

HCL OneDB 2.0.1

OneDB ODBC Driver Programmer's Manual



Contents

Chapter 1. HCL OneDB™ ODBC Driver Guide.....	3
Overview of HCL OneDB™ ODBC Driver.....	3
What is HCL OneDB™ ODBC Driver?.....	3
ODBC component overview.....	5
HCL OneDB™ ODBC Driver with the DMR.....	7
HCL OneDB™ ODBC Driver components.....	8
The HCL OneDB™ ODBC Driver API.....	11
Connection pooling.....	17
Global Language Support.....	25
X/Open standard interface.....	28
External authentication.....	29
Bypass ODBC parsing.....	32
BufferLength in character for SQLGetDiagRecW.....	33
HCL OneDB™ and ISAM error descriptions in SQLGetDiagRec.....	33
Improved performance for single-threaded applications.....	34
Partially supported and unsupported ODBC features.....	34
Configure data sources.....	37
Configure a DSN on UNIX™.....	37
Configuring a DSN in Windows™.....	48
Connection string keywords that make a connection.....	57
DSN migration tool.....	58
Data types.....	59
Data types.....	59
SQL data types.....	60
C data types.....	72
Report standard ODBC types.....	75
Convert data.....	79
Smart large objects.....	104
Data structures for smart large objects.....	104
Storage of smart large objects.....	105
Example of creating a smart large object.....	109
Transfer smart-large-object data.....	118
Access a smart large object.....	118
Retrieve the status of a smart large object.....	133
Read or write a smart large object to or from a file.....	142
Rows and collections.....	143
Allocating and binding a row or collection buffer.....	143
Example of creating a row and a list on the client.....	153
Modify a row or collection.....	160
Retrieve information about a row or collection.....	161
Client functions.....	161
Call a client function.....	161
Functions for smart large objects.....	163
Functions for rows and collections.....	183
Improve application performance.....	192
Error checking during data transfer.....	192
Enable delimited identifiers in ODBC.....	192
Connection level optimizations.....	193
Optimizing query execution.....	194
Insert multiple rows.....	194
Automatically freeing a cursor.....	194
Delay execution of the SQL PREPARE statement.....	195
Set the fetch array size for simple-large-object data.....	196
The SPL output parameter feature.....	197
OUT and INOUT parameters.....	198
Asynchronous execution.....	200
Update data with positioned updates and deletes.....	201
BIGINT and BIGSERIAL data types.....	202
Message transfer optimization.....	202
Error messages.....	205
Diagnostic SQLSTATE values.....	205
Map SQLSTATE values to HCL OneDB™ error messages.....	205
Map HCL OneDB™ error messages to SQLSTATE values.....	218
SQLParamOptions (core and level two only).....	244
Unicode.....	254
Overview of Unicode.....	254
Unicode in an ODBC application.....	255
Unicode in an ODBC application.....	256
Supported Unicode functions.....	257
ODBC Smart trigger.....	258
Index.....	272

Chapter 1. HCL OneDB™ ODBC Driver Guide

The *HCL OneDB™ ODBC Driver Programmer's Manual* serves as a user guide and reference for HCL OneDB™ ODBC Driver, which is the HCL OneDB™ implementation of the Microsoft™ Open Database Connectivity (ODBC) interface, Version 3.0.

These topics explain how to use the HCL OneDB™ ODBC Driver application programming interface (API) to access the HCL OneDB™ database and interact with the HCL OneDB™ database server.

These topics are written for C programmers who use HCL OneDB™ ODBC Driver to access HCL OneDB™ databases.

These topics assume that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational or object-relational databases, or exposure to relational database concepts
- C programming language

For information about software compatibility, see the HCL OneDB™ Client SDK release notes.

Overview of HCL OneDB™ ODBC Driver

These topics introduce the HCL OneDB™ ODBC Driver and describe its advantages and architecture. The topics also describe conformance, isolation and lock levels, libraries, and environment variables.

What is HCL OneDB™ ODBC Driver?

Open Database Connectivity (ODBC) is a specification for a database application programming interface (API).

Microsoft™ ODBC, Version 3.0, is based on the Call Level Interface specifications from X/Open and the International Standards Organization/International Electrotechnical Commission (ISO/IEC). ODBC supports SQL statements with a library of C functions. An application calls these functions to implement ODBC functionality.

ODBC applications enable you to perform the following operations:

- Connect to and disconnect from data sources
- Retrieve information about data sources
- Retrieve information about HCL OneDB™ ODBC Driver
- Set and retrieve HCL OneDB™ ODBC Driver options
- Prepare and send SQL statements
- Retrieve SQL results and process the results dynamically
- Retrieve information about SQL results and process the information dynamically

ODBC lets you allocate storage for results before or after the results are available. This feature lets you determine the results and the action to take without the limitations that predefined data structures impose.

ODBC does not require a preprocessor to compile an application program.

HCL OneDB™ ODBC Driver features

HCL OneDB™ ODBC Driver implements the Microsoft™ Open Database Connectivity (ODBC), Version 3.0 standard.

The HCL OneDB™ ODBC Driver product also provides the following features and functionality:

- Data Source Name (DSN) migration
- Driver Manager Replacement Module, which supports compatibility between ODBC 2.x applications and the ODBC driver, version 3.0.
- Microsoft™ Transaction Server (MTS), which is an environment that lets you develop, run, and manage scalable, component-based Internet and intranet server applications. MTS performs the following tasks:
 - Manages system resources, including processes, threads, and database connections, so that your application can scale to many simultaneous users
 - Manages server component creation, execution, and deletion
 - Automatically initiates and controls transactions to make your application reliable
 - Implements security so that unauthorized users cannot access your application
 - Provides tools for configuration, management, and deployment



Important: If you want to use distributed transactions managed by MTS with the HCL OneDB™ ODBC Driver, you must have connection pooling enabled.

- Extended data types, including rows and collections
- Long identifiers
- Limited support of bookmarks
- GLS data types
- Extensive error detection
- Unicode support
- XA support
- Internet Protocol Version 6 support for internet protocols of 128 bits. (For more information, see *HCL OneDB™ Administrator's Guide*.)

Support for extended data types

HCL OneDB™ ODBC Driver supports the extended data types.

HCL OneDB™ ODBC Driver supports the following extended data types:

- Collection (LIST, MULTISET, SET)
- DISTINCT
- OPAQUE (fixed, unnamed)
- Row (named, unnamed)
- Smart large object (BLOB, CLOB)
- Client functions to support some of the extended data types

Support for GLS data types

HCL OneDB™ ODBC Driver supports the GLS data types.

HCL OneDB™ ODBC Driver supports the following GLS data types:

- NCHAR
- NVARCHAR

Extended error detection

HCL OneDB™ ODBC Driver detects the ISM and XA types of errors.

Additional values for some ODBC function arguments

HCL OneDB™ ODBC Driver supports additional values for some ODBC function arguments.

These additional values for some ODBC function arguments include:

- *fDescType* values for SQLColAttributes
 - SQL_INFX_ATTR_FLAGS
 - SQL_INFX_ATTR_EXTENDED_TYPE_ALIGNMENT
 - SQL_INFX_ATTR_EXTENDED_TYPE_CODE
 - SQL_INFX_ATTR_EXTENDED_TYPE_NAME
 - SQL_INFX_ATTR_EXTENDED_TYPE_OWNER
 - SQL_INFX_ATTR_SOURCE_TYPE_CODE
- *fInfoType* return value for SQLGetInfo
 - SQL_INFX_LO_PTR_LENGTH
 - SQL_INFX_LO_SPEC_LENGTH
- SQL_INFX_LO_STAT_LENGTH
- *fOption* value for SQLGetConnectOption and SQLSetConnectOption: SQL_INFX_OPT_LONGID
- *fOption* value for SQLGetConnectOption and SQLSetConnectOption: SQL_ATTR_ENLIST_IN_DTC

ODBC component overview

ODBC with the HCL OneDB™ ODBC Driver includes several components.

ODBC with the HCL OneDB™ ODBC Driver can include the following components:

- Driver manager

An application can link to a driver manager, which links to the driver specified by the data source. The driver manager also checks parameters and transitions. On most UNIX™ platforms, the ODBC Driver Manager can be purchased from a third-party vendor.

On Microsoft™ Windows™ platforms, the ODBC Driver Manager is a part of the OS.

- HCL OneDB™ ODBC Driver

The driver provides an interface to HCL OneDB™ database server. Applications can use the driver in the following configurations:

- to link to the ODBC driver manager
- to link to the Driver Manager Replacement & the driver
- to link to the driver directly

- Data sources

The driver provides access to the following data sources:

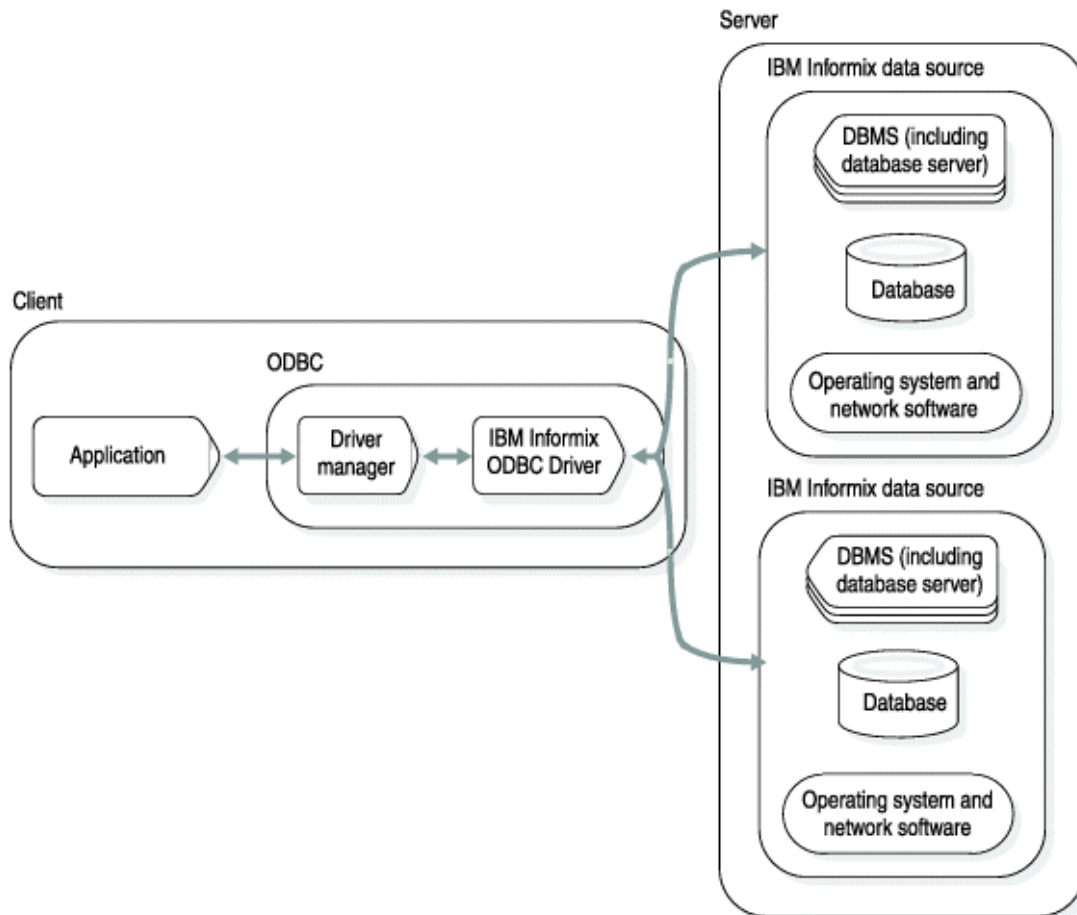
- database management systems (DBMS), including a database server
- databases
- operating systems and network software required for accessing the database

HCL OneDB™ ODBC Driver with a driver manager

There is software architecture when a driver manager is included in the system.

The following figure shows the software architecture when a driver manager is included in the system. In such a system, the driver and driver manager act like a single unit that processes function calls.

Figure 1. HCL OneDB™ ODBC Driver with a driver manager

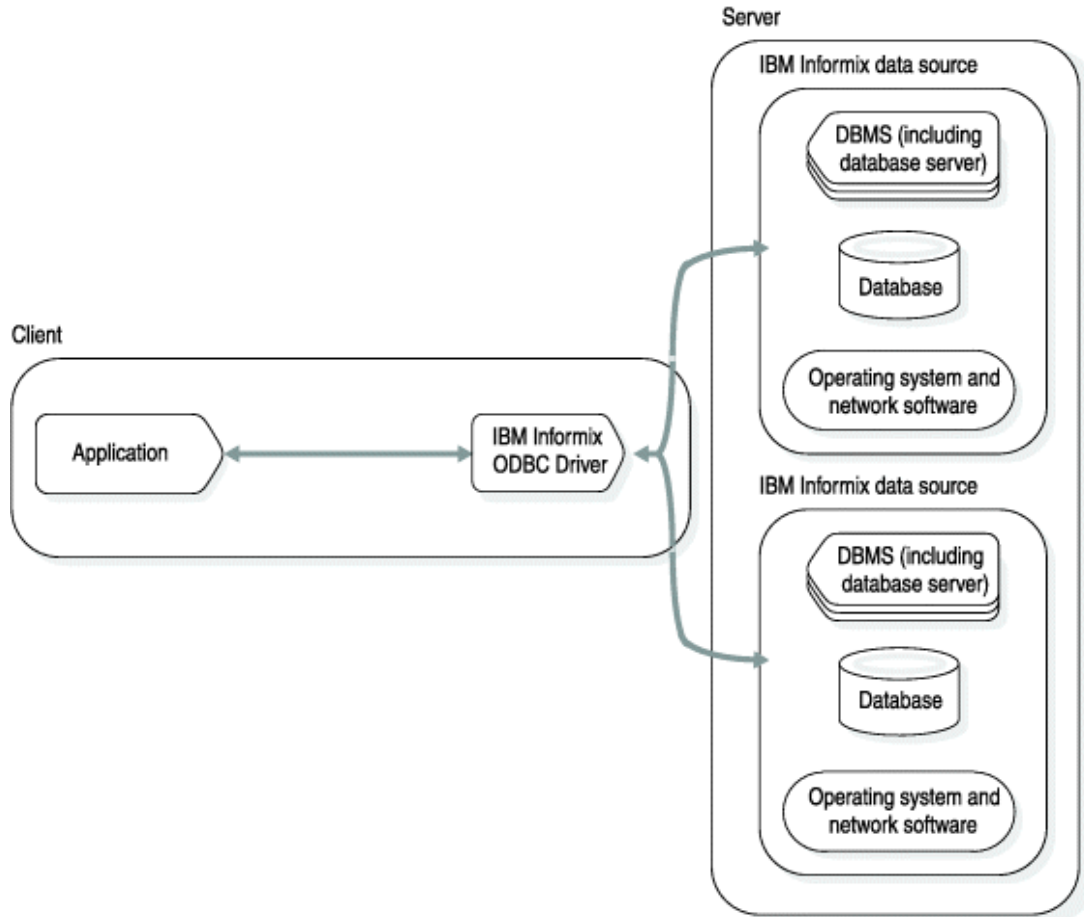


HCL OneDB™ ODBC Driver without a driver manager (UNIX™)

There is software architecture when a driver manager is not included in the system.

The following figure shows an application that uses HCL OneDB™ ODBC Driver without a driver manager. In this case, the application must link to the HCL OneDB™ ODBC Driver library.

Figure 2. HCL OneDB™ ODBC Driver without a driver manager



HCL OneDB™ ODBC Driver with the DMR

HCL OneDB™ ODBC Driver includes a Driver Manager Replacement (DMR) library. The DMR replaces the driver manager on platforms where no driver manager is available.

The following figure shows an ODBC configuration with the DMR.

Figure 3. Architecture of the driver manager replacement module



Applications that are linked directly to the ODBC Version 4.10 driver and the DMR do not require the ODBC Driver Manager.

In addition to supporting ODBC Version 4.10 features, the DMR supports compatibility between ODBC 2.x applications and Version 3.00 of the HCL OneDB™ ODBC Driver. To be compatible with ODBC 2.x applications, the application must link to Version 3.00 of HCL OneDB™ ODBC Driver through the DMR or through the ODBC Version 4.10 driver manager.

You cannot use the HCL® OneDB® DMR to connect to non-HCL OneDB™ data sources. The DMR does not support connection pooling. The DMR does not map between Unicode and ANSI APIs.

HCL OneDB™ ODBC Driver components

HCL OneDB™ ODBC Driver includes the four components.

HCL OneDB™ ODBC Driver includes the following components:

- Environment variables
- Header files
- Data types
- Libraries

Environment variables

There are four environment variables that you must set for the driver.

The following list describes environment variables that you must set for the driver. For more information about environment variables, see the *HCL OneDB™ Guide to SQL: Reference*.

ONEDB_HOME

Full path of the directory where the HCL OneDB™ Client Software Development Kit is installed.

On Windows™ platforms, **ONEDB_HOME** is a registry setting rather than an environment variable. It is set during installation.

PATH

Directories that are searched for executable programs. Your **PATH** setting must include the path to your `$ONEDB_HOME/bin` directory.

DBCENTURY (optional)

Controls the setting of year values. **DBCENTURY** affects a client program when a user issues a statement that contains a date or datetime string that specifies only the last two digits of the year. For example:

```
insert into datetable (datecol) values ('01/01/01');
```

The database server stores the date specified in this statement as either 01-01-1901 or 01-01-2001, depending on the **DBCENTURY** value on the client.

GL_DATE (optional)

GL_DATE controls the interpretation of dates. For example, you can specify whether the date format is `mm-dd-
yyyy` or `yyyy-mm-dd`.

Set environment variables on UNIX™

If you set the environment variables at the command line, you must reset them whenever you log on to your system. If you set the environment variables in a file, they are set automatically when you log on to your system.

HCL OneDB™ ODBC Driver provides a sample setup file called `setup.odbc` in `$ONEDB_HOME/etc`. You can use this file to set environment variables for the driver. The following list describes the environment variables that are in `setup.odbc`.

ONEDB_HOME

Full path of the directory where HCL OneDB™ Client Software Development Kit is installed.

ONEDB_SQLHOSTS

This value is optional. It specifies the directory that contains `sqlhosts`. By default, `sqlhosts` is in `$ONEDB_HOME/etc`. Set **ONEDB_SQLHOSTS** if you want `sqlhosts` to be in a different directory.

ODBCINI

This value is optional. You can use it to specify an alternative location for the `odbc.ini` file. The default location is your home directory.

Set environment variables in Windows™

If you set the environment variables at the command line, you must reset them whenever you log in to your Windows™ environment. If you set them in the Windows™ registry, however, they are set automatically when you log in.

HCL OneDB™ ODBC Driver stores environment variables in the following location in the Windows™ registry:

```
\HKEY_CURRENT_USERS\Software\Informix\Environment
```

In a Windows™ environment you must use `setnet32.exe`, or a tool that updates the registry correctly, to set environment variables that HCL OneDB™ dynamic link libraries (DLLs), such as `iclit09b.dll`, use. The Setnet utility can only be used to set HCL OneDB™ environment variables.

You can use environment variables as required by your development environment. For example, the compiler needs to know where to find the include files. To specify the location of the include files, set the environment variable **ONEDB_HOME** (or some other environment variable) and then set the include path to `ONEDB_HOME\incl\cli`.

The options for setting environment variables have the following precedence:

1. Setnet utility
2. Command line
3. Windows™ registry

Header files

You can use the `sql.h` and `sqlext.h` header files, which are part of the Microsoft™ compiler, to run HCL OneDB™ ODBC Driver.

To run HCL OneDB™ extensions, include the `infxcli.h` file, which is installed in `ONEDB_HOME/incl/cli`. This file defines HCL OneDB™ ODBC Driver constants and types, and provides function prototypes for the HCL OneDB™ ODBC Driver functions. If you include the `infxcli.h` file, it automatically includes the `sql.h` and `sqlext.h` files.

The `sql.h` and `sqlext.h` header files contain definitions of the C data types.

Include the `xa.h` header file in XA ODBC applications. ODBC applications on Windows™ require the HCL OneDB™ Client Software Development Kit to compile. Existing applications that use the ODBC driver might need to include the location of the Client SDK in the **PATH** environment variable before they are recompiled.

Data types

A column of data stored on a data source has an SQL data type.

HCL OneDB™ ODBC Driver maps HCL OneDB™-specific SQL data types to ODBC SQL data types, which are defined in the ODBC SQL grammar. (The driver returns these mappings through `SQLGetTypeInfo`. It also uses the ODBC SQL data types to describe the data types of columns and parameters in `SQLColAttributes` and `SQLDescribeCol`).

Each SQL data type corresponds to an ODBC C data type. By default, the driver assumes that the C data type of a storage location corresponds to the SQL data type of the column or parameter to which the location is bound. If the C data type of a storage location is not the *default* C data type, the application can specify the correct C data type with the *TargetType* argument for `SQLBindCol`, the *fCType* argument for `SQLGetData`, and the *ValueType* argument in `SQLBindParameter`. Before the driver returns data from the data source, it converts the data to the specified C data type. Before the driver sends data to the data source, it converts the data from the specified C data type to the SQL data type.

The HCL OneDB™ data type names differ from the Microsoft™ ODBC data type names. For information about these differences, see the appendix about data types in the *HCL OneDB™ ODBC Driver Programmer's Manual*.

Libraries

There is an installation procedure that installs libraries for UNIX™ and Windows™.

UNIX™

The installation procedure installs the following libraries into `ONEDB_HOME/lib/cli`. In each data source specification section in the `odbc.ini` file, set the driver value indicating the full path to one of the following library file names.

libifcli.a or libcli.a

Static version for single (nonthreaded) library

libifcli.so or iclis09b.so

Shared version for single (nonthreaded) library

libthcli.a

Static version for multithreaded library

libthcli.so or iclit09b.so

Shared version for multithreaded library

libifdrm.so or idmrs09a.so

Shared library for DMR (thread safe)

If you do not use a driver manager, your application needs to link to either the static or the shared version of the HCL OneDB™ ODBC Driver libraries.

The following compile command links an application to the thread-safe version of the HCL OneDB™ ODBC Driver libraries:

```
cc ... -L$ONEDB_HOME/lib/cli -lifdrm - lthcli
```

Windows™

The installation procedure installs the following libraries into `ONEDB_HOME\lib`.

iclit09b.lib

Enables linking directly to the driver without the use of a driver manager

iregt07b.lib

Allows linking directly to `iregt07b.dll`

The following compile command links an application to the thread-safe version of the HCL OneDB™ ODBC Driver libraries:

```
cl ... -L$ONEDB_HOME/lib/cli iclit09b.lib
```

If you use a driver manager, you must link your application to the driver manager library only, as the following example shows:

```
cl odb32.lib
```

The HCL OneDB™ ODBC Driver API

An application uses the HCL OneDB™ ODBC Driver API to make a connection to a data source, send SQL statements to a data source, process result data dynamically, and terminate a connection.

The driver enables your application to perform the following steps:

1. Connect to the data source.

You can connect to the data source through a DSN connection, or you can use DSN-less connection strings. Specify the data-source name and any additional information needed to complete the connection.

2. Process one or more SQL statements:

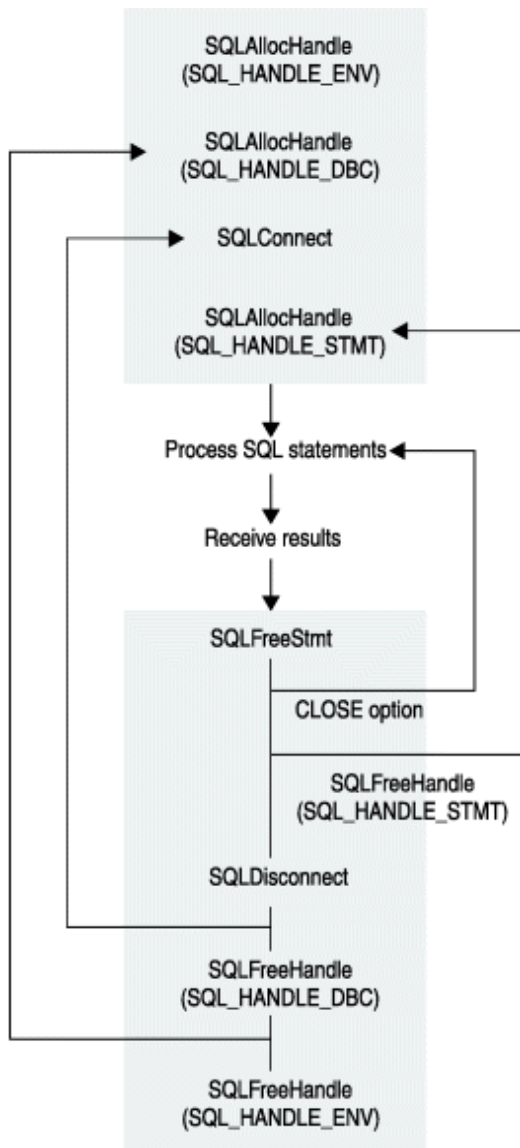
- a. Place the SQL text string in a buffer. If the statement includes parameter markers, set the parameter values.
- b. If the statement returns a result set, either assign a cursor name for the statement or let the driver assign one.
- c. Either prepare the statement or submit it for immediate execution.

- d. If the statement creates a result set, you can inquire about the attributes of the result set, such as the number of columns and the name and type of a specific column. For each column in the result set, assign storage and fetch the results.
 - e. If the statement causes an error, retrieve error information from the driver and take the appropriate action.
3. End any transaction by committing it or rolling it back.
 4. Terminate the connection when the application finishes interacting with the data source.

Every HCL OneDB™ ODBC Driver function name starts with the prefix SQL. Each function accepts one or more arguments. Arguments are defined as input (to the driver) or output (from the driver).

The following figure shows the basic function calls that an application makes even though an application generally calls other functions also.

Figure 4. Sample listing of function calls that the HCL OneDB™ ODBC Driver application makes



Environment, connection, and statement handles

When an application requests it, the driver and the driver manager allocate storage for information about the environment, each connection, and each SQL statement.

The driver returns a handle for each of these allocations to the application, which uses one or more handles in each call to a function.

The HCL OneDB™ ODBC Driver API uses the following types of handles:

Environment handles

Environment handles identify memory storage for global information, including the valid connection handles and the current active connection handle. The environment handle is an *henv* variable type. An application uses one environment handle. It must request this handle before it connects to a data source.

Connection handles

Connection handles identify memory storage for information about particular connections. A connection handle is an *hdbc* variable type. An application must request a connection handle before it connects to a data source. Each connection handle is associated with the environment handle. However, the environment handle can be associated with multiple connection handles.

Statement handles

Statement handles identify memory storage for information about SQL statements. A statement handle is an *hstmt* variable type. An application must request a statement handle before it submits SQL requests. Each statement handle is associated with exactly one connection handle. However, each connection handle can be associated with multiple statement handles.

Buffers

An application passes data to the driver in an input buffer. The driver returns data to the application in an output buffer.

The application must allocate memory for both input and output buffers. If the application uses the buffer to retrieve string data, the buffer must contain space for the null termination byte.

Some functions accept pointers to buffers that are used later by other functions. The application must ensure that these pointers remain valid until all applicable functions have used them. For example, the argument *rgbValue* in `SQLBindCol` points to an output buffer where `SQLFetch` returns the data for a column.

Input buffers

An application passes the address and length of an input buffer to the driver.

The length of the buffer must be one of the following values:

- A length greater than or equal to zero

This value is the actual length of the data in the input buffer. For character data, a length of zero indicates that the data is an empty (zero length) string. A length of zero is different from a null pointer. If the application specifies the length of character data, the character data does not need to be null-terminated.

- SQL_NTS

This value specifies that a character data value is null-terminated.

- SQL_NULL_DATA

This value tells the driver to ignore the value in the input buffer and use a NULL data value instead. It is valid only when the input buffer provides the value of a parameter in an SQL statement.

For character data that contains embedded null characters, the operation of HCL OneDB™ ODBC Driver functions is undefined; for maximum interoperability, it is better not to use them. HCL OneDB™ database servers treat null characters as end-of-string markers or as indicators that no more data exists.

Unless it is prohibited in a function description, the address of an input buffer can be a null pointer. In such cases, the value of the corresponding buffer-length argument is ignored.

Output buffers

An application passes arguments to the driver so that the driver can return data in an output buffer.

These arguments are:

- The address of the output buffer, to which the driver returns the data

Unless it is prohibited in a function description, the address of an output buffer can be a null pointer. In such cases, the driver does not return anything in the buffer and, in the absence of other errors, returns SQL_SUCCESS.

If necessary, the driver converts data before returning it. The driver always null-terminates character data before returning it.

- The length of the buffer

The driver ignores this value if the returned data has a fixed length in C, as with an integer, real number, or date structure.

- The address of a variable in which the driver returns the length of the data (the length buffer)

The returned length of the data is SQL_NULL_DATA if the data is a null value in a result set. Otherwise, the returned length of the data is the number of bytes of data that are available to return. If the driver converts the data, the returned length of the data is the number of bytes that remain after the conversion; for character data, it does not include the null-termination byte that the driver adds.

If the output buffer is too small, the driver attempts to truncate the data. If the truncation does not cause a loss of significant data, the driver returns the truncated data in the output buffer, returns the length of the available data (as opposed to the length of the truncated data) in the length buffer, and returns SQL_SUCCESS_WITH_INFO. If the truncation causes a loss of significant data, the driver leaves the output and length buffers untouched and returns SQL_ERROR. The application calls SQLGetDiagRec to retrieve information about the truncation or the error.

SQLGetInfo argument implementation

HCL OneDB™ implements the SQLGetInfo arguments for HCL OneDB™ ODBC Driver.

The following table describes the HCL OneDB™ implementation of SQLGetInfo arguments for HCL OneDB™ ODBC Driver.

Argument name	HCL OneDB™ implementation
SQL_ACTIVE_ENVIRONMENTS	HCL® OneDB® driver does not have a limit on number of active environments. Zero is always returned.
SQLAggregate_FUNCTIONS	HCL® OneDB® driver returns all aggregate functions that the database server supports.
SQL_ASYNC_MODE	HCL® OneDB® driver returns SQL_AM_NONE.
SQL_ATTR_METADATA_ID	Supported for GetInfo and PutInfo
SQL_BATCH_ROW_COUNT	HCL® OneDB® driver returns bitmask zero.
SQL_BATCH_SUPPORT	HCL® OneDB® driver returns bitmask zero.
SQL_CA1_POS_DELETE	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POS_POSITION	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POS_REFRESH	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POS_UPDATE	Operation arguments supported in a call to SQLSetPos
SQL_CA1_POSITIONED_DELETE	A DELETE WHERE CURRENT OF SQL statement is supported when the cursor is a forward-only cursor. (An SQL-92 entry-level-conforming driver always return this option as supported.)
SQL_CA1_POSITIONED_UPDATE	An UPDATE WHERE CURRENT OF SQL statement is supported when the cursor is a static-only cursor. (An SQL-92 entry-level-conforming driver always return this option as supported.)
SQL_CA1_LOCK_NO_CHANGE	A LockType argument of SQL_LOCK_NO_CHANGE is supported in a call to SQLSetPos when the cursor is a static-only cursor.
SQL_CA1_SELECT_FOR_UPDATE	A SELECT FOR UPDATE SQL statement is supported when the cursor is a forward-only cursor. (An SQL-92

Argument name	HCL OneDB™ implementation
	entry-level-conforming driver always return this option as supported.)
SQL_CATALOG_NAME	HCL® OneDB® driver returns 'Y'
SQL_COLLATION_SEQ	INTERSOLV DataDirect ODBC Driver returns InfoValuePtr (unmodified)
SQL_DDL_INDEX	Returns bitmask SQL_DL_CREATE_INDEX SQL_DL_DROP_INDEX
SQL_DESCRIBE_PARAMETER	Returns N; parameters cannot be described. (This is because the latest HCL OneDB™ database servers support function overloading such that multiple functions with the same name can accept different parameter types.)
SQL_DIAG_DYNAMIC_FUNCTION	Returns empty string
SQL_DROP_TABLE	Returns bitmask SQL_DT_DROP_TABLE SQL_DT_CASCADE SQL_DT_RESTRICT
SQL_DROP_VIEW	Returns bitmask SQL_DV_DROP_TABLE SQL_DV_CASCADE SQL_DV_RESTRICT
SQL_INDEX_KEYWORDS_	SQL_LK_ASC SQL_LK_DESC
SQL_INSERT_STATEMENT	Returns bitmask SQL_IS_INSERT_LITERALS SQL_INSERT_SEARCHED SQL_IS_SELECT_INT0
SQL_MAX_DRIVER_CONNECTIONS	Returns zero
SQL_MAX_IDENTIFIER_LEN	Returns different values, depending on database server capability
SQL_ODBC_INTERFACE_CONFORMANCE	Returns SQL_OIC_CORE
SQL_PARAM_ARRAY_ROW_COUNTS	Returns SQL_PARC_NO_BATCH
SQL_PARAM_ARRAY_SELECTS	Returns SQL_PAS_NO_SELECT
SQL_SQL_CONFORMANCE	Returns SQL_OSC_CORE
SQL_SQL92_FOREIGN_KEY_DELETE_RULE	Returns bitmask zero
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	Returns bitmask zero
SQL_SQL92_GRANT	Returns bitmask zero
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	Returns bitmask zero
SQL_SQL92_PREDICATES	Returns bitmask zero

Argument name	HCL OneDB™ implementation
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	Returns bitmask zero
SQL_SQL92_REVOKE	SQL_SR_CASCADE SQL_SR_RESTRICT
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	Returns bitmask zero
SQL_SQL92_STRING_FUNCTIONS	Returns bitmask zero
SQL_SQL92_VALUE_EXPRESSIONS	Returns bitmask zero
SQL_STANDARD_CLI_CONFORMANCE	Returns bitmask SQL_SCC_XOPEN_CLI_VERSION1 SQL_SCC_ISO92_CLI
SQL_STATIC_CURSOR_ATTRIBUTES1	Scrollable only
SQL_STATIC_CURSOR_ATTRIBUTES2	Scrollable only
SQL_XOPEN_CLI_YEAR	Returns string 1995

Connection pooling

OneDB ODBC driver also supports Connection Pooling capabilities. One may decide to use either Connection Pooling capabilities provided by ODBC Driver Manager or newly added capabilities in OneDB ODBC Driver

The main advantage of connection pooling will be for the applications (including open source drivers) directly using the OneDB ODBC driver (not via ODBC Driver Manager) and wants to optimize the connection resource in highly OLTP nature of applications where number of connections opened and closed are in large numbers.



Note: Driver Manager and Driver's connection pooling feature cannot be used together. Application should use only one of the features.

You can find the header file *infxcli.h*, which is located under `$ONEDB_HOME/incl/cli` directory. The file has the following additional macros:

```
#define SQL_INFX_ATTR_CP_TIMEOUT          2292
#define SQL_INFX_ATTR_CONNECTION_POOLING 2293
#define SQL_INFX_ATTR_CP_MATCH           2294
#define SQL_INFX_ATTR_CP_TOTAL_CONNECTIONS 2295
#define SQL_INFX_ATTR_CP_TOTAL_ACTIVE    2296
#define SQL_INFX_ATTR_CP_TOTAL_IDLE      2297

/* Connection pooling value parameters */
#define SQL_INFX_CP_STRICT_MATCH         1
#define SQL_INFX_CP_RELAXED_MATCH       2
#define SQL_INFX_CP_OFF                  1
#define SQL_INFX_CP_ON                   2
```

Enabling/Disabling connection pooling

There are two ways to enable/disable the connection pooling in ODBC:

- On Unix/Linux, user could use following attributes in `odbc.ini` file:

- Set 0 to disable, 1 to enable. By default its disabled(0).

```
InformixPooling=0
```

- Timeout for the connection in seconds range 5 to 60000. Default 60. Beyond range will be reset to default value without any error/warning.

```
InformixCTimeout=10
```

- Set 0 for strict match and 1 for relaxed match. By default, it is strict match(0). In strict match, more number of parameters are compared to find out the match from the available connection from the pool. It is recommended to use "strict match?".

```
InformixCPMatch=0
```

- On Unix/Linux or Windows, user can also set/get above values programatically using `SQLSetEnvAttr/SQLGetEnvAttr` APIs.

By default, its connection pooling is disabled, "`InformixPooling=0`" or `SQL_INFX_ATTR_CONNECTION_POOLING` set to `SQL_INFX_CP_OFF`.



Note: APIs used in program will have higher precedence over `odbc.ini`. For example: In `odbc.ini` file, pooling is disabled but in the application using `SQLSetEnvAttr(SQL_INFX_ATTR_CONNECTION_POOLING)`, if user enables the pooling, then application will take higher precedence. Similarly other way around. Also for Timeout and Match parameters.

The connection timing will be guided by Timeout parameter value (5 to 60000 seconds) set in "`InformixCTimeout`" parameter of `odbc.ini` file and/or `SQLSetEnvAttr(SQL_INFX_ATTR_CP_TIMEOUT)` value in the program. However this value is not guaranteed to be exact. Whenever the user calls `SQLDisconnect()` at that point of time, all the "not in use" connection will be iterated and whichever connections timeout has elapsed (difference between time of disconnect to current time), if this difference time is more than the timeout set by the user then such connections will be physically disconnected and all resources will be freed.

If connection pooling is enabled in `odbc.ini`, then it will be across all applications of ODBC. However, one can always change(enable/disable) at each ODBC Environment (`SQLAllocHandle(ENV)`) scope. For example:If it is enabled in the `odbc.ini` file and application has two Environments (`SQLAllocHandle(ENV1 and ENV2)`), then user can disable in one of the Environments by using `SQLSetEnvAttr(SQL_INFX_ATTR_CONNECTION_POOLING = DISABLED)`, the other Environment will be enabled from the effect of `odbc.ini` file. Hence, pooling object is managed at each Environment level.

The simple parameter comparisons across parameters (STRICT vs RELAX mode) is used to decide the pooling strategy. Depending on the mode used, all parameters must exactly match in order to hand over the available connection to the newly asked connection request.

You can use `SQLGetEnvAttr (SQL_INFX_ATTR_CONNECTION_POOLING / SQL_INFX_ATTR_CP_TIMEOUT / SQL_INFX_ATTR_CP_MATCH)` to list the values being set and if pooling is not enabled, you will get default parameters values or values (MODE and TIMEOUT) mentioned in the `odbc.ini` file.

The default value of SQL_INFX_ATTR_CP_TIMEOUT is 60 seconds, range is 5 to 60000 seconds, beyond this range, it will be reset to 60 seconds.

To track the number of connections using ODBC API, use SQLGetEnvAttr() API with below parameters:

```
SQLGetEnvAttr(SQL_INFX_ATTR_CP_TOTAL_CONNECTIONS) => This will return total number of connections.
               At any point of time, this will be equal to "Active +
               Idle" number of connections.
SQLGetEnvAttr(SQL_INFX_ATTR_CP_TOTAL_ACTIVE) => This will return connection count which are currently in use.
SQLGetEnvAttr(SQL_INFX_ATTR_CP_TOTAL_IDLE) => This will return connection count which are currently idle (not
               in use)and
               ready to be assigned for matching incoming connection
               requests.The idle number of connections
               are actually connected to the database server.
```



Note: If connection pooling is not enabled and calling these APIs would return default/unknown value -1 for each of the parameters.

Sample Output

```
Total connections = -1, active = -1, idle = -1
*** Connection pooling enabled ***
*** Connection pooling set to STRICT mode ***
*** Connection pooling timeout set to 10 seconds
Connected, label = connStrIn
Total connections = 1, active = 1, idle = 0
Disconnected, label = connStrIn
Total connections = 1, active = 0, idle = 1
Connected, label = connStrIn
Total connections = 1, active = 1, idle = 0
Disconnected, label = connStrIn
Total connections = 1, active = 0, idle = 1
Connected, label = connStrIn1
Total connections = 2, active = 1, idle = 1
Connected, label = connStrIn
Total connections = 2, active = 2, idle = 0
Disconnected, label = connStrIn
Total connections = 2, active = 1, idle = 1
Connected, label = connStrIn1
Total connections = 3, active = 2, idle = 1
Disconnected, label = connStrIn1
Total connections = 3, active = 1, idle = 2
Press enter to Exit, you may run 'userid informix onstat -g ses' to
see number of connections still opened due to connection pooling effect :

Statement handle freed successfully
Disconnected, label = connStrIn1
Total connections = 3, active = 0, idle = 3
Environment handle freed successfully
Press enter to Exit, you may run 'userid informix onstat -g ses' to see
there should be no connection, all connections must have closed by now :
sh-3.2$
```

```
/******
```

```

*      Licensed Materials - Property of HCL Technologies
*
*      "Restricted Materials of HCL"
*
*      HCL OneDB ODBC Application
*
*      Copyright HCL 2019 All rights reserved.
*
* Title:          ConnectionPooling.c
*
* Description:    Connection Pooling Sample ODBC Program
*
* Author   : User 1
*
* Compile/link options on Linux/Unix :
* gcc -g -c -fsigned-char -DNO_WIN32 -O -I$ONEDB_HOME/incl/cli ConnectionPooling.c
* gcc -g -o ConnectionPooling ConnectionPooling.o -L$ONEDB_HOME/lib/cli -L$ONEDB_HOME/lib/esql -lthcli -lifdmc
-lifgls -lifglx -lm -lnsl
*****
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#define __REENTRANT
#include <signal.h>
#ifdef NO_WIN32
#include <sys/wait.h>
#include <pthread.h>
#endif
#include <time.h>
#include <infxcli.h>

#define ERRMSG_LEN      200
#define NAMELEN        300
#define NUM_OF_INSTANCE 2

SQLHDBC      hdbc0;
SQLHDBC      hdbc1;
SQLHDBC      hdbc2;
SQLHDBC      hdbc3;
SQLHDBC      hdbc4;
SQLHENV      henv;
SQLHSTMT     hstmt;
SQLCHAR      connStrIn[NAMELEN];
SQLCHAR      connStrIn1[NAMELEN];
short        totalConn=0, totalActive=0, totalIdle=0;
SQLCHAR      connStrOut[NAMELEN];
SQLSMALLINT  connStrOutLen;

SQLINTEGER checkError (SQLRETURN      rc,

```

```

        SQLSMALLINT    handleType,
        SQLHANDLE      handle,
        SQLCHAR*       errmsg)
{
    SQLRETURN          retcode = SQL_SUCCESS;

    SQLSMALLINT        errNum = 1;
    SQLCHAR            sqlState[6];
    SQLINTEGER         nativeError;
    SQLCHAR            errMsg[ERRMSG_LEN];
    SQLSMALLINT        textLengthPtr;

    if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
    {
        while (retcode != SQL_NO_DATA)
        {
            retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState, &nativeError, errMsg, ERRMSG_LEN,
&textLengthPtr);

            if (retcode == SQL_INVALID_HANDLE)
            {
                fprintf (stderr, "CheckError function was called with an invalid handle!!\n");
                return 1;
            }

            if ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO))
                fprintf (stderr, "ERROR: %d: %s : %s \n", nativeError, sqlState, errMsg);

            errNum++;
        }

        fprintf (stderr, "%s\n", errMsg);
        return 1; /* all errors on this handle have been reported */
    }
    else
        return 0; /* no errors to report */
}

int GetConnectionPoolingAttributes()
{
    SQLRETURN rc = SQL_SUCCESS;

    rc = SQLGetEnvAttr (henv, SQL_INFX_ATTR_CP_TOTAL_CONNECTIONS , (void *) &totalConn, SQL_NTS, NULL);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection pooling Total Connections failed \nExiting!!");
        exit (-1);
    }
    rc = SQLGetEnvAttr (henv, SQL_INFX_ATTR_CP_TOTAL_ACTIVE , (void *) &totalActive, SQL_NTS, NULL);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection pooling Total Active failed \nExiting!!");
        exit (-1);
    }
    rc = SQLGetEnvAttr (henv, SQL_INFX_ATTR_CP_TOTAL_IDLE , (void *) &totalIdle, SQL_NTS, NULL);
    if (rc != SQL_SUCCESS)

```

```

    {
        fprintf (stdout, "Connection pooling Total Idle failed \nExiting!!");
        exit (-1);
    }

    printf("\nTotal connections = %d, active = %d, idle = %d",totalConn,totalActive, totalIdle );

return 0;
}

void SetConnectionString()
{
    memset(connStrIn, 0, sizeof(connStrIn));
    memset(connStrIn1, 0, sizeof(connStrIn1));
#ifdef NO_WIN32
    //sprintf((char *) connStrIn, "DRIVER={IBM INFORMIX ODBC DRIVER};

HOST=x.x.x.x;SERVER=ol_informix1210_2;SERVICE=8573;PROTOCOL=onsoctcp;DATABASE=sysadmin;UID=informix;PWD=xxxxx
");
    sprintf((char *) connStrIn, "DSN=SmartTrigger");
    sprintf((char *) connStrIn1, "DSN=odbc_demo");
#else
    //sprintf((char *) connStrIn, "DRIVER={IBM INFORMIX ODBC DRIVER};HOST=x.x.x.x;SERVER=ol_informix1210_1;
SERVICE=20195;PROTOCOL=onsoctcp;DATABASE=sysadmin;UID=informix;PWD=xxxxx");
    sprintf((char *) connStrIn, "DSN=SmartTrigger");
    sprintf((char *)connStrIn1, "DSN=odbc_demo");
#endif
    return;
}

int Connect(SQLHDBC *hdbc, SQLCHAR connStrIn[], SQLCHAR label[])
{
    SQLRETURN rc = SQL_SUCCESS;
    SQLHDBC tmpHdbc = NULL;
    /* Allocate the connection handle */
    rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &tmpHdbc);
    if (checkError (rc, SQL_HANDLE_ENV, henv, (SQLCHAR *) "Error(main) in Step 2 -- Connection Handle
Allocation failed\nExiting!!"))
        exit (-1);

    //printf("\nConnection handle BEFORE connection = %p", tmpHdbc);
    /* Establish the database connection */
    rc = SQLDriverConnect (tmpHdbc, NULL, connStrIn, SQL_NTS, connStrOut, NAMELEN, &connStrOutLen,
SQL_DRIVER_NOPROMPT);
    if (checkError (rc, SQL_HANDLE_DBC, tmpHdbc, (SQLCHAR *) "Error(main) in Step 3 -- SQLDriverConnect
failed\nExiting!!"))
        exit (-1);
    printf("\nConnected, label = %s", label);
    //printf("\nConnection handle AFTER connection = %p", tmpHdbc);

    *hdbc = tmpHdbc;

    return rc;
}

int DisconnectAndFree(SQLHDBC *hdbc, SQLCHAR label[])
{

```

```

SQLRETURN rc = SQL_SUCCESS;
SQLHDBC tmphdbc = *hdbc;
/* Disconnect from the data source */
rc = SQLDisconnect (tmphdbc);
printf("\nDisconnected, label = %s", label);

/* Free the environment handle and the database connection handle */
rc = SQLFreeHandle (SQL_HANDLE_DBC, tmphdbc);
//printf("\nDatabase handle freed successfully");

return rc;
}

int main (long      argc,
          char*    argv[])
{
/* Miscellaneous variables */
SQLRETURN      rc = 0;
BOOL           poolEnabled = 0;
BOOL           cpMode = 0;
SQLINTEGER     timeOut = -1;

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
    exit (-1);
}

/* Set the ODBC version to 3.0 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, (SQLCHAR *) "Error(main) in Step 1 -- SQLSetEnvAttr
failed\nExiting!!"))
    exit (-1);

GetConnectionPoolingAttributes();

//rc = SQLSetEnvAttr (henv, SQL_INFX_ATTR_CONNECTION_POOLING , (SQLPOINTER) SQL_INFX_CP_OFF, 0);
rc = SQLSetEnvAttr (henv, SQL_INFX_ATTR_CONNECTION_POOLING , (SQLPOINTER) SQL_INFX_CP_ON, 0);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Connection pooling call failed \nExiting!!");
    exit (-1);
}

rc = SQLGetEnvAttr (henv, SQL_INFX_ATTR_CONNECTION_POOLING , (void *) &poolEnabled, SQL_NTS, NULL);
if (rc != SQL_SUCCESS)
{
    fprintf (stdout, "Connection pooling call failed \nExiting!!");
    exit (-1);
}
if(SQL_INFX_CP_OFF == poolEnabled)
    printf("\n*** Connection pooling disabled ***");
else if(SQL_INFX_CP_ON == poolEnabled)
    printf("\n*** Connection pooling enabled ***");
else

```

```

    printf("\n*** What's going with Connection pooling!!!");

/*
    rc = SQLSetEnvAttr (henv, SQL_INFX_ATTR_CP_MATCH, (SQLPOINTER)SQL_INFX_CP_RELAXED_MATCH, 0);
    //rc = SQLSetEnvAttr (henv, SQL_INFX_ATTR_CP_MATCH, (SQLPOINTER)SQL_INFX_CP_STRICT_MATCH, 0);
    if (checkError (rc, SQL_HANDLE_ENV, henv, (SQLCHAR *) "Error(main) in Step 1 -- SQLSetEnvAttr
failed\nExiting!!!"))
        exit (-1);
*/
*/
    rc = SQLGetEnvAttr (henv, SQL_INFX_ATTR_CP_MATCH , (void *) &cpMode, SQL_NTS, NULL);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection pooling call failed \nExiting!!!");
        exit (-1);
    }
    if(SQL_INFX_CP_RELAXED_MATCH == cpMode)
        printf("\n*** Connection pooling set to RELAX mode ***");
    else if(SQL_INFX_CP_STRICT_MATCH == cpMode)
        printf("\n*** Connection pooling set to STRICT mode ***");
/*
    rc = SQLSetEnvAttr (henv, SQL_INFX_ATTR_CP_TIMEOUT, (SQLPOINTER)5, 0);
    if (checkError (rc, SQL_HANDLE_ENV, henv, (SQLCHAR *) "Error(main) in Step 1 -- SQLSetEnvAttr
failed\nExiting!!!"))
        exit (-1);
*/
*/
    rc = SQLGetEnvAttr (henv, SQL_INFX_ATTR_CP_TIMEOUT , (void *) &timeOut, SQL_NTS, NULL);
    if (rc != SQL_SUCCESS)
    {
        fprintf (stdout, "Connection pooling call failed \nExiting!!!");
        exit (-1);
    }
    printf("\n*** Connection pooling timeout set to %d seconds", timeOut);

    SetConnectionString();
    Connect(&hdbc0, connStrIn, "connStrIn");
    GetConnectionPoolingAttributes();
    DisconnectAndFree(&hdbc0, "connStrIn");
    GetConnectionPoolingAttributes();

    SetConnectionString();
    Connect(&hdbc1, connStrIn, "connStrIn");
    GetConnectionPoolingAttributes();
    DisconnectAndFree(&hdbc1, "connStrIn");
    GetConnectionPoolingAttributes();

    SetConnectionString();
    Connect(&hdbc2, connStrIn1, "connStrIn1");
    GetConnectionPoolingAttributes();
    rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc2, &hstmt );
    if (checkError (rc, SQL_HANDLE_DBC, hdbc2, (SQLCHAR *) "Error(main) in Step 4 -- Statement Handle
Allocation failed\nExiting!!!"))
        exit (-1);

    SetConnectionString();
    Connect(&hdbc3, connStrIn, "connStrIn");
    GetConnectionPoolingAttributes();
    DisconnectAndFree(&hdbc3, "connStrIn");

```



```

GetConnectionPoolingAttributes();

SetConnectionString();
Connect(&hdbc4, connStrIn1, "connStrIn1");
GetConnectionPoolingAttributes();
DisconnectAndFree(&hdbc4, "connStrIn1");
GetConnectionPoolingAttributes();

printf("\nPress enter to Exit, you may run 'userid informix onstat -g ses' to see
number of connections still opened due to connection pooling effect : ");
char c = getchar();

Exit:
/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);
printf("\nStatement handle freed successfully");

DisconnectAndFree(&hdbc2, "connStrIn1");
GetConnectionPoolingAttributes();

SQLFreeHandle (SQL_HANDLE_ENV, henv);
printf("\nEnvironment handle freed successfully");

printf("\nPress enter to Exit, you may run 'userid informix onstat -g ses' to see there
should be no connection, all connections must have closed by now : ");
c = getchar();

return (rc);
}

```



Global Language Support


HCL OneDB™ products can support many languages, cultures, and code sets. Global Language Support (GLS) provides support for all language- and culture-specific information.

The following table describes how to set the GLS options depending on your platform.

Platform	How to set GLS options
UNIX™	Specify the GLS options in the <code>odbc.ini</code> file.
Windows™	Specify the GLS options in the HCL OneDB™ ODBC Driver DSN Setup dialog box.

The following table describes the GLS options for HCL OneDB™ ODBC Driver.

GLS option	Description
Client locale	<p>Description: Locale and code set that the application runs in</p> <p>Format: locale.codeset@modifier</p> <p>odbc.ini field for UNIX™: CLIENT_LOCALE</p> <p>Default value for UNIX™: en_us.8859-1</p> <p>Default value for Windows™: en_us.1252</p> <p> Important: The setting of the CLIENT_LOCALE environment variable in the operating system environment and in Setnet32 are ignored by HCL OneDB™ ODBC Driver. To change the client locale, you must use this GLS option.</p>
Database locale	<p>Description: Locale and code set that the database was created in</p> <p>Format: locale.codeset@modifier</p> <p>odbc.ini field for UNIX™: DB_LOCALE</p> <p>Default value for UNIX™: en_us.8859-1</p> <p>Default value for Windows™: en_us.1252</p> <p> Important: The setting of the DB_LOCALE environment variable in the operating system environment and in Setnet32 are ignored by HCL OneDB™ ODBC Driver. To change the database locale, you must use this GLS option.</p>
Translation	<p>Description: Performs the code set conversion</p>

Option	Description
library	<p>Format:</p> <p>Path to the file for the library. The translation DLL must follow the ODBC standard for translation libraries. For more information, see the <i>HCL OneDB™ ODBC Driver Programmer's Manual</i>.</p> <p>odbc.ini field for UNIX™:</p> <p>TRANSLATIONDLL</p> <p>Default value for UNIX™:</p> <p>\$ONEDB_HOME/lib/esql/igo4a304.xx where xx is platform-specific extension for shared library</p> <p>Default value for Windows™:</p> <p>igo4n304.dll</p>
Translation option	<p>Description:</p> <p>Option for a non-HCL® OneDB® translation library</p> <p>Format:</p> <p>Determined by the vendor</p> <p>odbc.ini field for UNIX™:</p> <p>TRANSLATION_OPTION</p> <p>Default value for Windows™:</p> <p>Determined by the vendor</p> <p> Restriction: Do not set this option for the HCL OneDB™ translation library. The HCL OneDB™ translation library determines the translation option based on the client locale and database locale values.</p>
VMB character	<p>Description:</p> <p>Varying multibyte character length reporting option that specifies how to set <i>pcbValue</i> when <i>rgbValue</i> (the output area) is not large enough for the code-set-converted data. The possible values are:</p> <p>Estimate</p> <p>HCL OneDB™ ODBC Driver makes a worst-case estimate of the storage space needed to return the data.</p> <p>Exact</p> <p>HCL OneDB™ ODBC Driver writes the code-set-converted data to disk until all the data is converted. Because this option can degrade performance, it is recommended that you do not use this option unless your application does not work with Estimate.</p>

GLS option	Description
Possible values for UNIX™:	When you use a multibyte code set (in which characters vary in length from 1 to 4 bytes) for either the database or client locale, the length of a character string or simple large object (TEXT) in the database locale does not indicate the length of the string after it is converted to the client locale.
0 = Estimate	
1 = Exact	
Possible values for Windows™:	
Estimate	
Exact	
odbc.ini field for UNIX™:	
VMBCHARLENEXACT	
Default value for UNIX™:	
Estimate	
Default value for Windows™:	
Estimate	

For more information about GLS and locales, see the *HCL OneDB™ GLS User's Guide*.

X/Open standard interface

In addition to the standard ODBC functions, the HCL OneDB™ ODBC Driver also supports the additional functions.

The following functions are supported by HCL OneDB™ ODBC Driver

IFMX_SQLGetXaHenv

Function for obtaining the environment handle associated with an XA Connection

IFMX_SQLGetXaHdbc

Function for obtaining the database handle associated with an XA Connection

xa_open

Function takes an xa_info parameter. The HCL OneDB™ ODBC Driver uses this xa_info to establish a XA connection

The format of xa_info is as follows:

```
<applicationtoken>|<DSN name>
```

The application token is a unique number the application generates for each xa_open request. It must use the same application token as parameter to IFMX_SQLGetXaHenv and IFMX_SQLGetXaHdbc to get the associated environment and database handles.

External authentication

For HCL OneDB™ Version 10.0 and later, you can implement external authentication through the HCL OneDB™ ODBC Driver.

There are two external authentication modules available to use with the HCL OneDB™ ODBC Driver. The Pluggable Authentication Module (PAM), works on UNIX™ and Linux™ servers and the LDAP Authentication is supported on Microsoft™ Windows™ operating systems.

Pluggable Authentication Module (PAM) on UNIX™ and Linux™

You can use Pluggable Authentication Module (PAM) with the HCL OneDB™ ODBC Driver on the UNIX™ and Linux™ operating systems that support PAM.

PAM enables system administrators to implement different authentication mechanisms for different applications. For example, the needs of a system like the UNIX™ login program might be different from an application that accesses sensitive information from a database. PAM allows for many such scenarios in a single machine, because the authentication services are attached at the application level.

LDAP Authentication on Windows™

You can use LDAP Authentication with the HCL OneDB™ ODBC Driver on Windows™ operating systems. LDAP Authentication is similar to the Pluggable Authentication Module.

Use the LDAP Authentication Support module when you want to use an LDAP server to authenticate your system users. The module contains source code that you can modify for your specific LDAP Authentication Support module. For information about installing and customizing the LDAP Authentication Support module, see the *HCL OneDB™ Security Guide*.

Connection pooling and authentication

In ODBC, the driver manager controls connection pooling.

An application programmer must be aware of the effects of connection pooling when using authentication. The driver manager does not control when its connections are placed in the pool or when a connection is pulled from the pool. If the application connects and disconnects without the knowledge of the user, the performance benefits of connection pooling are maintained and the user does not receive any unexpected authentication challenges. If the application does make the user aware they are re-establishing a connection, there is still no authentication challenge because the connection between the driver manager and the server was never closed. For more information about connection pooling, see the *Microsoft™ Data Access SDK* documentation.

The SQLSetConnectAttr() function with authentication

Use the SQLSetConnectAttr() function to specify the callback function used by the server.

SQLSetConnectAttr() is also used to specify what parameters are used by the callback function. Parameter attributes are passed back to the callback function exactly as they are specified to the driver.

The following attributes are HCL OneDB™-specific extensions to the ODBC standard:

Parameter	Type	Description
SQL_INFX_ATTR_PAM_FUNCTION	void *	A pointer to the callback function.
SQL_INFX_ATTR_PAM_RESPONSE_BUF	void *	A generic pointer to a buffer containing the response to an authentication challenge.
SQL_INFX_ATTR_PAM_RESPONSE_LEN	int	The length of the response buffer in bytes.
SQL_INFX_ATTR_PAM_RESPONSE_LEN_PTR	int *	The address which stores the number of bytes in the response.
SQL_INFX_ATTR_PAM_CHALLENGE_BUF	void *	A generic pointer to a buffer containing the authentication challenge. The driver stores any challenge received from the server into this buffer. If the buffer is not large enough to contain the challenge, the challenge is truncated. The callback function can detect this challenge by comparing the buffer length with the number of bytes in the challenge. It is up to the application developer to detect this situation and handle it correctly.
SQL_INFX_ATTR_PAM_CHALLENGE_BUF_LEN	int	The length of the challenge buffer in bytes.
SQL_INFX_ATTR_PAM_CHALLENGE_LEN_PTR	int *	The address that stores the number of bytes in the challenge.

The challenge and response buffer pointers can be null. If the authentication server requires the information that would be stored in these buffers, a connection failure results due to an authentication failure. The challenge length information is returned whether the connection is successful or not. If the message type does not require a response, the response buffer might be null (default) or it might contain an empty string.

The attributes in the previous table can be set at any time and in any order. However, they are only valid for connections established with subsequent calls to one of the driver's connect functions.

You can set the isolation level with the SQLSetConnectAttr() API by using one of the following connection attributes:

- SQL_TXN_READ_UNCOMMITTED = Read Uncommitted
- SQL_TXN_READ_COMMITTED = Read Committed
- SQL_TXN_SERIALIZABLE = Serializable
- SQL_TXN_REPEATABLE_READ = Repeatable Read
- SQL_TXN_LAST_COMMITTED = Last Committed
- SQL_TXN_TRANSACTION = Transaction

If you use the `SQL_TXN_LAST_COMMITTED` or `SQL_TXN_TRANSACTION` attributes with the `SQLSetConnectAttr()` API, then your applications must link directly to the HCL OneDB™ ODBC Driver instead of to the ODBC Driver Manager. However, if the attribute is specified in the `odbc.ini` file or the Data Source Administrator, the application can be linked with ODBC Driver Manager.

If you use the `SQL_TXN_TRANSACTION` attribute, then the isolation level set in the DTC application is propagated to the server. This option should be used only in Windows™ DTC applications.

The default behavior of the ODBC driver is to trim blank characters from the end of `VARCHAR` column results. To preserve trailing spaces, set the **`SQL_INFX_ATTR_LEAVE_TRAILING_SPACES`** attribute:

```
SQLSetConnectAttr( hdbc, SQL_INFX_ATTR_LEAVE_TRAILING_SPACES,
(SQLPOINTER)SQL_TRUE, SQL_IS_INTEGER );
```

To trim trailing spaces, change `SQL_TRUE` to `SQL_FALSE`.

The behavior is limited to the connection.

Connect functions

Any ODBC function which establishes a connection, `SQLConnect()`, `SQLDriverConnect()`, or `SQLBrowseConnect()`, can be used with authentication modules.

Consider the following when using these functions.

The `SQLConnect()` function

The `DriverCompletion` parameter to the `SQLConnect()` function can take the following values

- `SQL_DRIVER_PROMPT`
- `SQL_DRIVER_COMPLETE`
- `SQL_DRIVER_COMPLETE_REQUIRED`
- `SQL_DRIVER_NOPROMPT`

If an authentication challenge is expected, it is recommended that you use **`SQL_DRIVER_NOPROMPT`**. Using other values might result in the user being presented with multiple requests for authentication information.

The `SQLBrowseConnect()` function

The `SQLBrowseConnect()` function is designed to be used iteratively where the driver provides guidance to the application on how to complete the connection string and the application prompts the user for the required values. This can create situations where the user is presented with multiple prompts between connection string completion and authentication.

Additionally, it is typical for the driver to present a choice of databases to the application as part of the connection string completion process. However, the driver is not able to query the server for a list of databases until after the user is authenticated. Depending on application logic, whether it provides a database name in the original connection string, and whether a challenge is going to be received from the authentication server, it might not be possible to use `SQLBrowseConnect()` when the server uses authentication.

Third-party applications or intermediate code

When using authentication, it is the responsibility of the application to handle any challenges that originate from the authentication server.

To handle the challenges, the application programmer must be able to register a callback function with the driver. Because there are no attributes defined in the ODBC standard that are used to accomplish this, the attributes used are HCL OneDB™ extensions.

Many applications are written with ADO layer of Microsoft™ to abstract the ODBC calls from the developer. Most Visual Basic applications are written with ADO objects. These applications and third-party applications in general are not aware of the HCL OneDB™ extensions and are not able to handle an authentication challenge.

The ODBC Data Source Administrator on Windows™ also falls under the class of third-party applications. Not all features are available when configuring a UNIX™ data source. For example, the **Apply and Test Connection** button and the **User Server Database Locale** toggle does not work if a challenge is received because those features require the ability to connect to the server.

Bypass ODBC parsing

You can bypass ODBC parsing by using several options.

Sometimes you might want to improve performance by bypassing ODBC parsing. Do not bypass ODBC parsing if these conditions exist:

- You intend to use ODBC escape sequences in your query.
- You intend to call any catalog functions (for example, SQLColumns, SQLProcedureColumns, or SQLTables) after running your SQL query.

You can bypass ODBC parsing in the following ways:

- Set SKIPPARSING to 1 in the connection string. The connection string is used in a SQLDriverConnect call. For example:

```
connString="DB=xxx;UID=xxx;...;SKIPPARSING=1;"
```

- Include SQL_INFX_ATTR_SKIP_PARSING in a SQLSetConnectAttr call, for example:

```
SQLSetConnectAttr (hdbc, SQL_INFX_ATTR_SKIP_PARSING,
                  (SQLPOINTER)SQL_TRUE, SQL_IS_USMALLINT);
```

Use this call after the connection is completed. To restore ODBC parsing, change SQL_TRUE to SQL_FALSE. After this value is enabled at the connection level, all statement handles that are allocated with the connection inherit this property.

- In a SQLSetStmtAttr call, include SQL_TRUE. To restore ODBC parsing, change SQL_TRUE to SQL_FALSE.

```
SQLSetStmtAttr (hstmt, SQL_INFX_ATTR_SKIP_PARSING,
               (SQLPOINTER)SQL_TRUE, SQL_IS_USMALLINT);
```

- On UNIX™ systems, in .odbc.ini set SKIPPARSING=1. To restore ODBC parsing, reset the value to SKIPPARSING=0.

The precedence of bypassing ODBC parsing is as follows:

- If ODBC parsing is bypassed or reset in the `odbc.ini` file (on UNIX™ systems) and also in the application with the `SQLDriverConnect`, `SQLSetConnectAttr`, or the `SQLSetStmtAttr` APIs, the API setting takes precedence.
- If ODBC parsing is bypassed or reset in the application with the `SQLDriverConnect` API and also in the `SQLSetConnectAttr` or `SQLSetStmtAttr` APIs, the latter takes precedence.

BufferLength in character for SQLGetDiagRecW

The `SQLGetDiagRecW` API returns diagnostic information in the output buffer, where the `BufferLength` parameter is the length of buffer allocated.

The default for `BufferLength` is the number of bytes allocated. After setting the `SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW` attribute to `TRUE`, the `BufferLength` is treated as a specific number of characters. As a Widechar API, one character=`sizeof(SQLWCHAR)` bytes.

Set the attribute in the following ways:

- ```
SQLSetEnvAttr (henv, SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW,
 (SQLPOINTER)SQL_TRUE, SQL_IS_UINTEGER);
```
- ```
SQLSetConnectAttr (hdbc, SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW,
                    (SQLPOINTER)SQL_TRUE, SQL_IS_UINTEGER);
```
- ```
SQLSetStmtAttr (hstmt, SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW,
 (SQLPOINTER)SQL_TRUE, SQL_IS_UINTEGER);
```
- Set the `LENGTHINCHARFORDIAGRECW=1` in the connection string.
- On UNIX™ systems, in `odbc.ini` set `LENGTHINCHARFORDIAGRECW=1`

The precedence of setting `SQL_INFX_ATTR_LENGTHINCHARFORDIAGRECW` is:

- Setting the `SQLSetEnvAttr` attribute reflects to the `henv`, `hdbc`, and `hstmt` handles.
- Resetting the `hdbc` and `hstmt` handles through
  - Setting `SQLSetConnectAttr`
  - Passing the attribute in connection string
  - Enabling the **Length in Chars for SQLGetDiagRecW** option in the DSN
- If the `hstmt` handle is set or not set by the previously mentioned methods, setting `SQLSetStmtAttr` resets it.

## HCL OneDB™ and ISAM error descriptions in SQLGetDiagRec

The `SQLGetDiagRec` API returns diagnostic information in the output buffer, where the error description is for the HCL OneDB™ error message.

When the HCL OneDB™ server encounters an error, it returns the HCL OneDB™ error code and the associated error description. There is an additional error code, the ISAM error code, which provides information that is necessary to understand the circumstances that caused the HCL OneDB™ error code.

If you do not set an attribute for the `SQLSetConnectAttr` API, the `SQLGetDiagRec` API returns the HCL OneDB™ error message.

If you set the `SQL_DIAG_ISAM_ERROR` attribute for the `SQLGetDiagField` API, the `SQLGetDiagField` API returns the ISAM error message.

If you set the `SQL_INFX_ATTR_IDSISAMERRMSG` attribute for the `SQLSetConnectAttr` API, the `SQLGetDiagRec` API returns both the HCL OneDB™ error message and the ISAM error message.

Set the `SQL_INFX_ATTR_IDSISAMERRMSG` attribute in the following way:

```
SQLSetConnectAttr (hdbc, SQL_INFX_ATTR_IDSISAMERRMSG,
 (SQLPOINTER)SQL_TRUE, SQL_IS_UINTEGER);
```

## Improved performance for single-threaded applications

You are likely to improve the performance of single-threaded applications by using the `SINGLETHREADED` connection parameter. The value is off by default.

Do not use this parameter in an XA/MSDTC environment. You can set the `SINGLETHREADED` connection parameter in a connection string as the following example shows:

```
DSN=xxx;Uid=xxx;Pwd=xxx;SINGLETHREADED=1;"
```

## Partially supported and unsupported ODBC features

HCL OneDB™ ODBC Driver supports partial implementation of several ODBC features.

These ODBC features are

- Transaction processing
- ODBC cursors
- ODBC bookmarks
- **SQLBulkOperations**

## Transaction processing

HCL OneDB™ ODBC Driver implementation of transaction isolation levels and transaction modes is slightly different from the Microsoft™ ODBC implementation of these features.

The following topics describe the implementation of transaction isolation levels and transaction modes in HCL OneDB™ ODBC Driver.

## Transaction isolation levels

HCL OneDB™ ODBC Driver supports three transaction isolation levels for the HCL OneDB™ database server.

The following table lists the transaction isolation levels that HCL OneDB™ ODBC Driver supports for the HCL OneDB™ database server.

| Database servers | Transaction isolation levels                                                                                                                   |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| HCL OneDB™       | <ul style="list-style-type: none"> <li>• SQL_TXN_READ_COMMITTED</li> <li>• SQL_TXN_READ_UNCOMMITTED</li> <li>• SQL_TXN_SERIALIZABLE</li> </ul> |

The default transaction isolation level is `SQL_TXN_READ_COMMITTED`. To change the transaction isolation level, call `SQLSetConnectOption()` with an *fOption* value of `SQL_TXN_ISOLATION`.

For more information about transaction isolation levels, see the `SQL_DEFAULT_TXN_ISOLATION` and `SQL_TXN_ISOLATION_OPTION` descriptions in the *HCL OneDB™ ODBC Driver Programmer's Manual*.

## Changing the transaction mode

You can change the transaction mode from its default of auto-commit to manual commit.

### About this task

To change the transaction mode to manual commit:

1. Enable transaction logging for your database server.

For information about transaction logging, see your *HCL OneDB™ Administrator's Guide*.

2. Call `SQLSetConnectOption()` with `SQL_AUTOCOMMIT` set to `SQL_AUTOCOMMIT_OFF`.

## ODBC cursors

HCL OneDB™ ODBC Driver supports static and forward cursors but not dynamic and keyset-driven cursors.

For more information about cursors, see the *HCL OneDB™ ODBC Driver Programmer's Manual*.

## ODBC bookmarks

A *bookmark* is a value that identifies a row of data.

HCL OneDB™ ODBC Driver supports bookmarks with **SQLFetchScroll** and `SQLExtendedFetch` and does not support them with **SQLBulkOperations**. HCL OneDB™ ODBC Driver supports bookmarks to the following extent:

- Uses only variable length bookmarks.
- `SQL_DESC_OCTET_LENGTH` is set to 4 for bookmark columns.
- A bookmark is an integer that contains the row number within the row set, starting with 1.
- Bookmarks persist only if the cursor remains open.
- **SQLFetchScroll**, using `SQL_FETCH_BOOKMARK` for the fetch orientation argument, is fully supported.
- **SQLBulkOperations** does not update the bookmark column for `SQL_ADD`.

For more information about ODBC bookmarks, see the *HCL OneDB™ ODBC Driver Programmer's Manual*.

## SQLBulkOperations

HCL OneDB™ ODBC Driver supports only the SQL\_ADD argument of **SQLBulkOperations**.

## SQLDescribeParam

SQLDescribeParam is an ODBC API which returns metadata for the parameters of a query.

In earlier releases of the HCL OneDB™ ODBC Driver, the SQLDescribeParam API returned SQL\_UNKNOWN if the API was called to get information about an expression value or a parameter that was embedded inside another routine. This restriction no longer applies to values of BOOLEAN, LVARCHAR, or of built-in non-opaque HCL OneDB™ data types that are returned by the following expressions in other UDRs:

- Binary arithmetic expressions
  - Addition ( + )
  - Subtraction ( - )
  - Multiplication ( \* )
  - Division ( / )
- Relational operator expressions
  - Less than ( < )
  - Less than or equal to ( <= )
  - Equal to ( =, == )
  - Greater than or equal to ( >= )
  - Greater than ( > )
  - Not equal to ( <>, != )
- The following string operations
  - Concatenation ( || )
  - MATCHES
  - LIKE
- BETWEEN ... AND conditional expressions

For example, if the column **tab1.c1** is an INT data type, SQLDescribeParam() returns type int for the input host variable of the following query:

```
select c1, c2 from tab1 where ABS(c1) > ?;
```

The UDR from the other side of the expression can be a column expression or a built-in routine, but it cannot be a user-defined routine. In earlier releases, the SQLDescribeParam API returns SQL\_UNKNOWN for expression values and parameters that are embedded in another procedure in the following cases:

- The value on the other side of the expression is a user-defined routine.
- Another operand of the same expression is a user-defined routine.
- The data type of any operand of the expression is not a BOOLEAN, LVARCHAR, or a built-in non-opaque data type.

## Unsupported Microsoft™ ODBC driver features

HCL OneDB™ ODBC Driver does not support implementation of the certain Microsoft™ ODBC driver features.

The unsupported Microsoft™ ODBC driver features are:

- Asynchronous communication mode
- Concurrency checking
  - SQL\_CA2\_OPT\_ROWVER\_CONCURRENCY
  - SQL\_CA2\_OPT\_VALUES\_CONCURRENCY
- CONVERT scalar functions
- Cursor simulation features:
  - SQL\_CA2\_CRC\_APPROXIMATE
  - SQL\_CA2\_CRC\_EXACT
  - SQL\_CA2\_SIMULATE\_NON\_UNIQUE
  - SQL\_CA2\_SIMULATE\_TRY\_UNIQUE
  - SQL\_CA2\_SIMULATES\_UNIQUE
- Dynamic cursor attributes
- Installer DLL

## Configure data sources

These topics explain how to configure a data source (DSN) on UNIX™ and Windows™ for HCL OneDB™ ODBC Driver.

After you install the driver, you must configure your DSN before you can connect to it.

### Configure a DSN on UNIX™

The configuration files provide information, such as driver attributes, that the driver uses to connect to DSNs.

This section provides information about driver specifications and DSN specifications on UNIX™, and describes the following DSN configuration files:

- `sqlhosts`
- `odbcinst.ini`
- `odbc.ini`

To modify these files, use a text editor. The section also provides examples of driver and DSN specifications.

If you are enabling single-sign on (SSO), additional steps are in "Configuring ESQL/C and ODBC Drivers for SSO" in *HCL OneDB™ Security Guide*.

### The odbcinst.ini file

The `odbcinst.ini` file has entries for all the installed drivers on your computer.

Installed ODBC drivers use the `odbcinst.ini` sample file, which is located in `$ONEDB_HOME/etc/odbcinst.ini`. To create your `odbcinst.ini` file, copy the `odbcinst.ini` sample file to your home directory as `$HOME/.odbcinst.ini` (note the added dot at the beginning of the file name). Update this file when you install a new driver or a new version of a driver. The following table describes section items in the `$HOME/.odbcinst.ini` file.

| Section                    | Description                                     | Status   |
|----------------------------|-------------------------------------------------|----------|
| ODBC drivers               | List of names of all the installed ODBC drivers | Optional |
| ODBC driver specifications | List of driver attributes and values            | Optional |

## ODBC drivers

Use examples to obtain information about ODBC drivers.

The following example illustrates information about drivers:

```
[ODBC Drivers]
driver_name1=Installed
driver_name2=Installed
```

The following example illustrates information about installed drivers:

```
[ODBC Drivers]
IBM INFORMIX ODBC DRIVER=Installed
```

## Driver specifications

Each installed driver has a properties section under the name of the driver.

The following example illustrates a driver-specification format:

```
[driver name1]
Driver=driver_library_path
Setup=setup/driver_library_path
APILevel=api_level_supported
ConnectFunctions=connectfunctions
DriverODBCVer=odbc_version
FileUsage=file_usage
SQLLevel=sq_level
```

The following example illustrates information about driver specifications:

```
[IBM INFORMIX ODBC DRIVER]
Driver=/vobs/tristarm/odbc/iclis09b.so
Setup=/vobs/tristarm/odbc/iclis09b.so
APILevel=1
ConnectFunctions=YYY
DriverODBCVer=03.50
FileUsage=0
SQLLevel=1
```

The following table describes the keywords that are in the driver-specification section.

| Keywords                | Description                                                                                                           | Status   |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------|----------|
| <b>API Level</b>        | ODBC interface conformance level that the driver supports<br><br>0=None<br>1=Level 1 supported<br>2=Level 2 supported | Required |
| <b>ConnectFunctions</b> | Three-character string that indicates whether the driver supports SQLConnect, SQLDriverConnect, and SQLBrowseConnect  | Required |
| <b>DriverODBCVer</b>    | Character string with the version of ODBC that the driver supports                                                    | Required |
| <b>Driver</b>           | Driver library path                                                                                                   | Required |
| <b>FileUsage</b>        | Number that indicates how a file-based driver directly treats files in a DSN                                          | Required |
| <b>Setup</b>            | Setup library                                                                                                         | Required |
| <b>SQLLevel</b>         | Number that indicates the SQL-92 grammar that the driver supports                                                     | Required |

For a detailed description of the Driver Specification section, see the *HCL OneDB™ ODBC Driver Programmer's Manual*.

## The odbc.ini file

The `odbc.ini` file is a sample data-source configuration information file.

For the location of the `odbc.ini` file, see the release notes. To create this file, copy `odbc.ini` to your home directory as `$HOME/.odbc.ini` (note the added dot at the beginning of the file name). Every DSN to which your application connects must have an entry in this file. The following table describes the sections in `$HOME/.odbc.ini`.

| Section                   | Description                                                                                                                                                                     | Status   |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| ODBC Data Sources         | This section lists the DSNs and associates them with the name of the driver. You need to provide this section only if you use an ODBC driver manager from a third-party vendor. | Required |
| Data Source Specification | Each DSN listed in the ODBC Data Sources section has a Data-Source Specification section that describes the DSN.                                                                | Required |
| ODBC                      | This section lists ODBC tracing options.                                                                                                                                        | Optional |

Follow these rules to include comments in the `odbc.ini` file on UNIX™ systems:

- Begin a comment with a semicolon (;) or number sign (#) in the first position of the first line.
- If a comment includes multiple lines, you can begin following comment lines with a space or tab character (\t).
- You can include blank lines in comments.

## ODBC Data Sources

Each entry in the ODBC Data Sources section lists a DSN and the driver name.

The *data\_source\_name* value is any name that you choose. It is like an envelope that contains all relevant connection information about the DSN.

The following example illustrates an ODBC data-source format:

```
[ODBC Data Sources]
data_source_name=OneDB ODBC DRIVER
```

The following example defines two DSNs called EmpInfo and CustInfo:

```
[ODBC Data Sources]
EmpInfo=OneDB ODBC DRIVER
CustInfo=OneDB ODBC DRIVER
```

## Data-source specification

Each DSN in the data sources section has a data-source specification section.

The following example illustrates a data-source specification format:

```
[data_source_name]
Driver=driver_path
Description=data_source_description
Database=database_name
LogonID=user_id
pwd=user_password
Server=database_server
CLIENT_LOCALE=application_locale
DB_LOCALE=database_locale
TRANSLATIONDLL=translation_path
CURSORBEHAVIOR=cursor_behavior
DefaultUDTFetchType=default_UDT_Fetch_type
ENABLESCROLLABLECURSORS=enable_scroll_cursors
ENABLEINSERTCURSORS=enable_insert_cursors
OPTIMIZEAUTOCOMMIT=optimize_auto_commit
NEEDODBCYPESONLY=need_odbc_types_only
OPTOFC=open_fetch_close_optimization
REPORTKEYSETCURSORS=report_keyset_cursors
FETCHBUFFERSIZE=fetchbuffer_size
DESCRIBEDECIMALFLOATPOINT=describe_decimal_as_float
USESERVERDBLOCALE=use_server_dblocale
DONOTUSELVARCHAR=do_not_use_lvarchar
REPORTCHARCOLASWIDECHARCOL=char_col_as_widechar_col
UPDATE_DESCRIBE=update_describe
[ODBC]
 UNICODE=unicode_type
 LENGTHINCHARFORDIAGRECW=bufferlength_as_number_of_characters
 LEAVE_TRAILING_SPACES=leave_trailing_spaces
```


The following table describes the keywords that are in the data-source specification section and the order that they appear in each section.



| Keywords                         | Description                                                                                                                                                                                                            | Status   |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| <b>data_source_name</b>          | Data source specified in the Data Sources section                                                                                                                                                                      | Required |
| <b>Driver</b>                    | Path for the driver<br><br>Set this value to the complete path name for the driver library. For more information about the library directory and file names, see the release notes.                                    | Required |
| <b>Description</b>               | Description of the DSN<br><br>Configured for a single user or for system users.                                                                                                                                        | Optional |
| <b>Database</b>                  | Database to which the DSN connects by default                                                                                                                                                                          | Required |
| <b>LogonID</b>                   | User identification or account name for access to the DSN                                                                                                                                                              | Optional |
| <b>pwd</b>                       | Password for access to the DSN                                                                                                                                                                                         | Optional |
| <b>Server</b>                    | HCL OneDB™ database server on which <i>database_name</i> is in                                                                                                                                                         | Required |
| <b>CLIENT_LOCALE</b> (GLS only)  | Client locale. Default value: <b>en_us.8859-1</b>                                                                                                                                                                      | Optional |
| <b>DB_LOCALE</b> (GLS only)      | Database locale. Default value: <b>en_us.8859-1</b>                                                                                                                                                                    | Optional |
| <b>TRANSLATIONDLL</b> (GLS only) | DLL that performs code-set conversion; default value: <code>\$ONEDB_HOME/lib/esql/ig04a304.xx</code> where <i>xx</i> represents a platform-specific file extension                                                     | Optional |
| <b>CURSORBEHAVIOR</b>            | Flag for cursor behavior when a commit or rollback transaction is called.<br><br>Possible values are: <ul style="list-style-type: none"> <li>• 0=close cursor</li> <li>• 1=preserve cursor</li> </ul> Default value: 0 | Optional |
| <b>DefaultUDTFetchType</b>       | Default UDT fetch type.<br><br>Default value: SQL_C_BINARY<br><br>Possible values are: <ul style="list-style-type: none"> <li>• SQL_C_BINARY</li> <li>• SQL_C_CHAR</li> </ul>                                          | Optional |
| <b>ENABLESCROLLABLECURSORS</b>   | If this option is activated, the HCL OneDB™ ODBC Driver supports only scrollable, static cursors.                                                                                                                      | Optional |

| Keywords                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                     | Status   |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
|                            | <p>Available only as a connection option:</p> <pre>SQL_INFX_ATTR_ENABLE_SCROLL_CURSORS</pre> <p>or as a connection attribute string:</p> <pre>EnableScrollableCursors</pre> <p>Default value is: 0 (disabled)</p>                                                                                                                                                                                                                               |          |
| <b>ENABLEINSERTCURSORS</b> | <p>Reduces the number of network messages sent to and from the server by buffering the inserted rows used with arrays of parameters and insert statements. This option improves the performance of bulk insert operations.</p> <p>Available as both a connection and statement option:</p> <pre>SQL_INFX_ATTR_ENABLE_INSERT_CURSORS</pre> <p>or as a connection attribute string:</p> <pre>EnableInsertCursors</pre> <p>Default value is: 0</p> | Optional |
| <b>OPTIMIZEAUTOCOMMIT</b>  | <p>Defers automatic commit operations while cursors remain open. This option can reduce database communication when the application is using non-ANSI logging databases.</p> <p>Available as a connection option:</p> <pre>SQL_INFX_ATTR_OPTIMIZE_AUTOCOMMIT</pre> <p>or as a connection attribute string:</p> <pre>OptimizeAutoCommit</pre> <p>Default value is: 1 (enabled)</p>                                                               | Optional |
| <b>OPTOFC</b>              | <p>Causes the driver to buffer the open, fetch, and close cursor messages to the server. This option eliminates one or more message cycles when you use SQLPrepare, SQLExecute, and SQLFetch statements to fetch data with a cursor.</p> <p>Only available as a connection option:</p> <pre>SQL_INFX_ATTR_OPTOFC</pre> <p>or as a connection attribute string:</p> <pre>OPTOFC</pre>                                                            | Optional |

| Keywords                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Status   |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
|                                  | Default is: 0 (disabled)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |          |
| <b>REPORTKEYSETCURSORS</b>       | <p>Causes the driver to report (through SQLGetInfo) that it supports forward-only, static, and keyset-driver cursors even though the driver only supports forward-only and static cursors. This option is used to enable dynaset-type functions, such as Microsoft™ Visual Basic, which require drivers that support keyset-driven cursors.</p> <p>Also available as connection option:</p> <pre>SQL_INFX_ATTR_REPORT_KEYSET_CURSORS</pre> <p>or as a connection attribute string:</p> <pre>ReportKeysetCursors</pre> <p>Default is: 0 (disabled)</p>                                                                                                                                                                                                                                                        | Optional |
| <b>FETCHBUFFERSIZE</b>           | <p>Size of a fetch buffer in bytes.</p> <p>Available as connection attribute string:</p> <pre>FETCHBUFFERSIZE</pre> <p>The maximum size of the fetch buffer is 2 GB.</p> <p>Default is: 32767</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Optional |
| <b>DESCRIBEDECIMALFLOATPOINT</b> | <p>Describes all floating-point decimal columns as:</p> <ul style="list-style-type: none"> <li>• Float(SQL_REAL) or</li> <li>• Float(SQL_DOUBLE)</li> </ul> <p>A floating-point decimal column is a column that was created without a scale, for example DECIMAL(12). Some prepackaged applications such as Visual Basic cannot properly format Decimal columns that do not have a fixed scale. To use these applications, you must enable this option or redefine the column with a fixed scale.</p> <p>Enabling this option has the disadvantage that SQL_DECIMAL is an exact numeric data type while SQL_REAL and SQL_DOUBLE are approximate numeric data types. SQL_DECIMAL with a precision of 8 or less are described as SQL_REAL. With a precision greater than 8, it is described as SQL_DOUBLE.</p> | Optional |

| Keywords                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Status   |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
|                                    | <p>Available as connection attribute string:</p> <pre>DESCRIBEDECIMALFLOATPOINT</pre> <p>Default is: 0 (disabled)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |          |
| <b>USESERVERDBLOCALE</b>           | <p>Users server database locale.</p> <p>Available as a connection attribute string:</p> <pre>USESERVERDBLOCALE</pre> <p>Default is: 0 (disabled)</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Optional |
| <b>DONOTUSELVARCHAR</b>            | <p>If enabled, the SQLGetTypeInfo does not report LVARCHAR as a supported type (DATA_TYPE) of SQL_VARCHAR. Some applications use LVARCHAR instead of VARCHAR, even in columns that are less than 256 bytes. The minimum number of bytes transmitted for LVARCHAR is higher than VARCHAR. Many LVARCHAR columns can result in the rowset size exceeding the maximum.</p> <p> <b>Important:</b> Enable this option only if your SQL_VARCHAR columns are less than 256 bytes.</p> <p>Available as a connection attribute string:</p> <pre>DONOTUSELVARCHAR</pre> <p>Default is: 0 (disabled)</p> | Optional |
| <b>REPORTCHARCOLASWIDECHARCOL</b>  | <p>Causes SQLDescribeCol to report character columns as wide character columns as follows:</p> <ul style="list-style-type: none"> <li>• SQL_CHAR is reported as SQL_WCHAR</li> <li>• SQL_VARCHAR is reported as SQL_WVARCHAR</li> <li>• SQL_LONGVARCHAR is reported as SQL_WLONGVARCHAR</li> </ul> <p>Available as a connection attribute string:</p> <pre>REPORTCHARCOLASWIDECHARCOL</pre> <p>Default is: 0 (disabled)</p>                                                                                                                                                                                                                                                      | Optional |
| <b>UPDATE_DESCRIBE OR UPD_DESC</b> | <p>This is required particularly for BLOB/CLOB data types. If enabled, server will send the description of these data types</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Optional |

| Keywords                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Status   |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
|                                | <p>which will be used by ODBC Driver. This option should only be enabled when needed. It should not be turned on all the time as that would cause more round trips between client and server.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• 0</li> <li>• 1</li> </ul> <p>Default value is: 0</p>                                                                                                                                                                                                                                                                                                                                                                                     |          |
| <b>UNICODE</b>                 | <p>Indicates the type of Unicode used by an application. This attribute applies to UNIX™ applications only and is set in the ODBC section of the <code>odbc.ini</code> file. The following considerations apply:</p> <ul style="list-style-type: none"> <li>• Applications on UNIX™ not linking to Data Direct ODBC driver manager should set this to UCS-4</li> <li>• Applications on IBM AIX® with version lower than 5L should set this attribute to UCS-2.</li> <li>• Applications using Data Direct driver manager do not need to set this attribute.</li> </ul> <p>Default is: UTF-8</p> <p>For more information about using Unicode in an ODBC application, see <a href="#">Unicode on page 254</a>.</p> | Required |
| <b>LENGTHINCHARFORDIAGRECW</b> | <p>If enabled, the SQLGetDiagRecW API treats the BufferLength parameter as the number of characters.</p> <p>Default is: FALSE (disabled)</p> <p>For more information about using the BufferLength parameter see <a href="#">BufferLength in character for SQLGetDiagRecW on page 33</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                    |          |
| <b>LEAVE_TRAILING_SPACES</b>   | <p>If enabled, the driver preserves blank characters at the end of VARCHAR/LVARCHAR column results.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• 0 (trim trailing spaces)</li> <li>• 1 (preserve trailing spaces)</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                        |          |

| Keywords | Description         | Status |
|----------|---------------------|--------|
|          | Default value is: 0 |        |

The following example shows the configuration for a DSN called EmpInfo:

```
[EmpInfo]
Driver=/informix/lib/cli/iclis09b.so
Description=Demo data source
Database=odbc_demo
LogonID=admin
pwd=tiger
Server=ifmx_91
CLIENT_LOCALE=en_us.8859-1
DB_LOCALE=en_us.8859-1
TRANSLATIONDLL=/opt/informix/lib/esql/igo4a304.so
```

The following example shows the configuration for a DSN called HCL OneDB™ 9:

```
[Informix9]
Driver=/work/informix/lib/cli/iclis09b.so
Description=Informix 9.x ODBC Driver
LogonID=user1
pwd=tigress4
Database=odbc_demo
ServerName=my_server
```

If you specify a null `LogonID` or `pwd`, the following error occurs:

```
Insufficient connect information supplied
```



**Tip:** You can establish a connection to a DSN with null values for `LogonID` and `pwd` if the local HCL OneDB™ database server is on the same computer where the client is located. In this case, the current user is considered a trusted user.

A sample data source, with no `LogonID` and `pwd`, where the server and client are on the same computer, might look like the following example:

```
Driver=/work/informix/lib/cli/iclis09b.so
Description=Informix 9.x ODBC Driver
LogonID=
pwd=tiger
Database=odbc_demo
ServerName=ifmx_server
```

## Set the isolation level (UNIX™ only)

Set the isolation level in the `odbc.ini` file by using the `ISOLATIONLEVEL` and `SQL_TXN_LAST_COMMITTED` keywords.

To specify the isolation level in the `odbc.ini` file, use the following keyword and values:

- `ISOLATIONLEVEL` = level
- `SQL_TXN_LAST_COMMITTED` = last committed

where level is a number from 0 to 5:

- 0 = Automatically considers the default based on database type
- 1 = Read Uncommitted
- 2 = Read Committed (default for non-ANSI databases)
- 3 = Repeatable Read (default for ANSI databases)
- 4 = Serializable
- 5 = Last Committed

If an application calls `SQLSetConnectAttr` with the `SQL_ATTR_TXN_ISOLATION` attribute and sets the value before connecting, and later sets `ISOLATIONLEVEL` or `ISOLVL` in the connection string, the connection string is the final value to be used.

The `SQL_TXN_TRANSACTION` isolation level is not supported on UNIX™ platforms.

## ODBC section

The values in the ODBC section of `odbc.ini` specify ODBC tracing options.

With tracing, you can find the log of calls made and also the return codes for each call. These options are set through the Tracing tab of the ODBC Data Source Administrator dialog box on Windows™.

The following table describes the tracing options in the ODBC section:

**Table 1. Tracing options for ODBC section of `odbc.ini`**

| Option    | Details                                                 |
|-----------|---------------------------------------------------------|
| TRACE=1   | Tracing enabled                                         |
| TRACE=0   | Tracing disabled                                        |
| TRACEFILE | Set to where you want to driver to write the call logs. |
| TRACEDLL  | Always <code>idmrs09a.so</code>                         |

The following example illustrates an ODBC section specification format:

```
[ODBC]
TRACE=1
TRACEFILE=/WORK/ODBC/ODBC.LOG
TRACEDLL=idmrs09a.so
UNICODE=UCS-4
```

You must set the `TRACEFILE` to where you want the driver to write all of the call logs. Keep in mind that `TRACE=1` means that tracing is enabled. `TRACE=0` disables tracing options.

## Set the `$ODBCINI` environment variable

Set the `$ODBCINI` environment variable to provide access to your DSN by system users

By default, HCL OneDB™ ODBC Driver uses configuration information found in the `$HOME/.odbc.ini` file. If you want to provide access to your DSN by system users, modify the path in the `$ODBCINI` environment variable to point to

another configuration file that also contains the configuration information found in the `$HOME/.odbc.ini` file. Then change the configuration file permissions to allow read access for system users. Do not change the permissions to the `$HOME/.odbc.ini` file.

In the following example, the configuration file name is `myodbc.ini`:

```
setenv ODBCINI /work/myodbc.ini
```

## The .netrc file

The `.netrc` file contains data for logging in to a remote database server over the network.

Create the `.netrc` file in the home directory where the client computer initiates the connection. Set the `.netrc` file permissions for the user to deny read access by the group and others.

To connect to a remote database server, create entries in the `.netrc` file for the `LogonID` and `pwd` required to autoconnect to the data source. To establish a connection to a remote data source, the ODBC driver first reads the `LogonID` and `pwd` from the data source entry in the `$HOME/.odbc.ini` file. If the `$HOME/.odbc.ini` file does not specify a `LogonID` and `pwd`, the ODBC driver searches the `$HOME/.netrc` file.

For example, to allow an autologin to the computer called **ray** by using the login name `log8in` with password `mypassword`, your `.netrc` file contains the following line:

```
machine ray login log8in password mypassword
```

For information about the `.netrc` file, see the UNIX™ man pages.

## Configuring a DSN in Windows™

In Windows™ environments, HCL OneDB™ ODBC Driver provides a GUI to configure DSNs.

### About this task


To configure a DSN:

- Choose a procedure to modify your DSN:
  - Choose from:**
    - Choose the User DSN option to restrict access to one user.
    - Choose the System DSN option to restrict access to system users.
    - Choose the File DSN option to allow access to all users on a network.
- Enter DSN-configuration values to create a DSN, such as the data-source name, the database server name, and the database locale.

### What to do next

For a description of values, see the following two tables. Values are shown in the order that they appear in each section. You can also use Microsoft™ ODBC, Version 2.5 or later, to configure a DSN.



 **Tip:** To find out what DSN you have, click the **About** tab and read the contents of the **Description** text box.

 **Important:** To configure a DSN on the Windows™ 64-bit platform, you must use the 32-bit ODBC Data Source Administrator:

```
C:\WINDOWS\SysWOW64\odbcad32.exe
```

You must specify the user and password. If you are enabling single-sign on (SSO), additional steps are in "Configuring ESQ/C and ODBC Drivers for SSO" in *HCL OneDB™ Security Guide*.

**Table 2. Required DSN values**

| Required values         | Description                                                                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Data Source Name</b> | DSN to access<br><br>This value is any name that you choose. <b>Data Source Name</b> is like an envelope that contains all relevant connection information about the DSN. |
| <b>Database Name</b>    | Name of the database to which the DSN connects by default                                                                                                                 |
| <b>Host Name</b>        | Computer on which <b>Server</b> is in                                                                                                                                     |
| <b>Protocol</b>         | Protocol used to communicate with <b>Server</b><br><br>After you have added a DSN, the menu will display the available choices.                                           |
| <b>Server Name</b>      | HCL® OneDB® database server on which <b>Database</b> is in                                                                                                                |
| <b>Service</b>          | HCL® OneDB® database server process that runs on your <b>Host</b> computer<br><br>Confirm the service name with your system administrator or database administrator.      |

**Table 3. Optional DSN values**

| Optional values        | Description                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Client Locale</b>   | Default value: <b>en_us.1252</b>                                                                                                                                                         |
| <b>Database Locale</b> | Default value: <b>en_us.1252</b>                                                                                                                                                         |
| <b>Description</b>     | Any information, such as version number and service                                                                                                                                      |
| <b>Options</b>         | General information, such as password settings<br><br>For more information about this value, see the <code>sqlhosts</code> information in your <i>HCL OneDB™ Administrator's Guide</i> . |
| <b>Password</b>        | Password for access to the DSN                                                                                                                                                           |

**Table 3. Optional DSN values (continued)**

| Optional values            | Description                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Translation Library</b> | Dynamic linked library (DLL) that performs code-set conversion; default value: <code>\$ONEDB_HOME\bin\ig04n304.dll</code>                                                                                                                                                                                                                                                                 |
| <b>User ID</b>             | User identification or account name for access to the DSN                                                                                                                                                                                                                                                                                                                                 |
| <b>Translation Option</b>  | <p>Option for a non-HCL® OneDB® translation library</p> <p>Varying multibyte character length reporting option that specifies how to set <i>pcbValue</i> when <i>rgbValue</i> (the output area) is not large enough for the code-set-converted data</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>• 0=Estimate</li> <li>• 1=Exact</li> </ul> <p>Default value: 0</p> |
| <b>Cursor Behavior</b>     | <p>Flag for cursor behavior when a commit or rollback transaction is called</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>• 0=close cursor</li> <li>• 1=preserve cursor</li> </ul> <p>Default value: 0</p>                                                                                                                                                       |

After you complete these steps, you will connect to the DSN.

## Configuring a new user DSN or system DSN

Access the ODBC Data Source Administrator dialog box to configure a new user DSN or system DSN.

### About this task

To configure a new user DSN or system DSN:

1. Choose **Start > Settings > Control Panel**.
2. Double-click ODBC to open the ODBC Data Source Administrator dialog box.

#### Choose from:

- To configure a user DSN, go to step [3 on page 50](#).
- To configure a system DSN, click the **System DSN** tab and go to step [3 on page 50](#).

All subsequent steps are the same to configure either a user DSN or a system DSN.

3. Click **Add**.

The Create New Data Source dialog box opens.

4. Double-click **OneDB ODBC driver** on the Create New Data Source wizard.

The **General** page for the HCL OneDB™ ODBC Driver Setup dialog box opens.

5. Enter the values in the **General** page, as the following example shows:

**Example**

- Data Source Name: `odbc33int`
- Description: `file DSN 3.81 on turbo`

For a description of the values, see [Table 2: Required DSN values on page 49](#) and [Table 3: Optional DSN values on page 49](#).



**Restriction:** Do not click **OK** after you enter the values on this page. If you click **OK** before you enter all the values, you get an error message.

6. Click the **Connection** tab to display the **Connection** page and enter the values, as the following example shows:

**Example**

- Server Name: `ol_clipper` (or use the menu to choose a server that is on the sqlhosts registry. If you use the menu, the ODBC application sets the Host Name, Service, Protocol, and Options values.)
- Host Name: `clipper`
- Service: `turbo`
- Protocol: `onsoctcp` (or use the menu to choose a protocol)
- Database Name: `odbc_demo` (or use the menu to find a database name)
- User ID: `myname`
- Password: `*****`

To save the values you chose and verify that your DSN connects successfully, click **Apply & Test Connection**. An ODBC Message dialog box opens. The box tells you if your connection was successful or, if it was not, tells you which Connection-tab value is incorrect.

7. Click the **Environment** tab to display the **Environment** page and enter the values, as the following example shows:

**Example**

- Client Locale: `en_US.CP1252`
- Database Locale: `en_US.CP1252`
- Use Server Database Locale: if check box is checked, database locale value is set to the server locale. If the check box is cleared, the database locale is set to the default locale, `en_US.CP1252`.
- Translation Library: `ONEDB_HOME\lib\esql\ig04n304.dll`
- Translation Option: `0`
- Cursor Behavior: `0 - Close`
- VMB Character: `0 - Estimate`

- Fetch Buffer Size: 4096
- Isolation Level: 0 - Default will be considered, Read Committed (non-ANSI databases) or Repeatable Read (ANSI databases)

8. Click the **Advanced** tab to display the **Advanced** page and click all applicable boxes.

| Option                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Auto commit optimization</b>      | <p>This option defers automatic commit operations while cursors remain open and can reduce database communication when the application is using non-ANSI logging databases. This option is available only as a connection option:</p> <p style="background-color: #f0f0f0; padding: 2px;"><code>SQL_INFX_ATTR_OPTIMIZE_AUTOCOMMIT</code></p> <p>or as a connection attribute string: "OptimizeAutoCommit"</p> <p>The default is: 1 (enabled).</p>                                                                                          |
| <b>Open-Fetch-Close optimization</b> | <p>This option causes the driver to buffer the open, fetch, and close cursor messages to the server. In addition, this option eliminates one or more message round trips when you use SQLPrepare, SQLExecute, and SQLFetch statements to fetch data with a cursor. This option is available only as a connection option:</p> <p style="background-color: #f0f0f0; padding: 2px;"><code>SQL_INFX_ATTR_OPTOFC</code></p> <p>or as a connection attribute string: "OPTOFC"</p> <p>The default is: 0 (disabled).</p>                           |
| <b>Insert cursors</b>                | <p>This option reduces the number of network messages sent to and from the server by buffering the inserted rows that are used with arrays of parameters and insert statements. This option can greatly improve the performance of bulk insert operations, and is available as both connection and statement options:</p> <p style="background-color: #f0f0f0; padding: 2px;"><code>SQL_INFX_ATTR_ENABLE_INSERT_CURSORS.</code></p> <p>or as a connection attribute string: "EnableInsertCursors"</p> <p>The default is: 0 (disabled).</p> |
| <b>Scrollable cursor</b>             | <p>If this option is activated, HCL OneDB™ ODBC Driver, Version 2.90 and later, supports only scrollable, static cursors. This option is available only as a connection option:</p> <p style="background-color: #f0f0f0; padding: 2px;"><code>SQL_INFX_ATTR_ENABLE_SCROLL_CURSORS</code></p> <p>or as a connection attribute string: "EnableScrollableCursors"</p> <p>The default is: 0 (disabled).</p>                                                                                                                                    |
| <b>Report KeySet cursors</b>         | <p>This option causes the driver to report (through SQLGetInfo) that it supports forward-only, static, and keyset-driven cursor types, although the driver only supports forward-only and static cursors. When you set this option, the driver enables dynaset-type functions, such as functions for Microsoft™</p>                                                                                                                                                                                                                        |

| Option                                                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                        | <p>Visual Basic. These functions require drivers that support keyset-driven cursor types. This option is also available as a connection attribute:</p> <pre data-bbox="410 352 821 380">SQL_INFX_ATTR_REPORT_KEYSET_CURSORS</pre> <p>or as a connection attribute string: "ReportKeysetCursors"</p> <p>The default is: <input type="checkbox"/> (disabled).</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <p><b>Report standard ODBC types only</b></p>                          | <p>If you activate this feature, the driver causes SQLGetTypeInfo to map all occurrences of user-defined types (UDTs) as follows:</p> <p><b>Blob</b></p> <p style="padding-left: 40px;"><b>SQL_LONGVARBINARY</b></p> <p><b>Clob</b></p> <p style="padding-left: 40px;"><b>SQL_LONGVARBINARY</b></p> <p><b>Multiset</b></p> <p style="padding-left: 40px;"><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p><b>Set</b></p> <p style="padding-left: 40px;"><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p><b>List</b></p> <p style="padding-left: 40px;"><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p><b>Row</b></p> <p style="padding-left: 40px;"><b>SQL_C_CHAR/SQL_C_BINARY</b></p> <p>The driver maps multiset, set, row, and list data types to <b>SQL_C_CHAR</b> or <b>SQL_C_BINARY</b>, which is the default UDT</p> <p>FetchType to <b>SQL_C_CHAR</b> features.</p> <p>The default is: <input type="checkbox"/> (disabled).</p> |
| <p><b>Describe decimal floating point as SQL_REAL / SQL_DOUBLE</b></p> | <p>This option describes all floating-point decimal columns as Float (SQL_REAL or SQL_DOUBLE). A floating-point decimal column is a column that was created without a scale, ex: DECIMAL(12). Some prepackaged applications such as Visual Basic cannot properly format Decimal columns that do not have a fixed scale. To use these applications you must enable this option or redefine the column with a fixed scale.</p> <p>There is a disadvantage to enabling this option however, SQL_DECIMAL is an exact numeric data type while SQL_REAL and SQL_DOUBLE are approximate numeric data types. A SQL_DECIMAL with a precision of 8 or less are described as SQL_REAL, with a precision greater than 8 it is SQL_DOUBLE.</p>                                                                                                                                                                          |

| Option                                              | Description                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                     | The default is: 0 (disabled).                                                                                                                                                                                                                                                                                                                                                   |
| <b>Do not use LVARCHAR</b>                          | Causes SQLGetTypeInfo to not report LVARCHAR as a supported type of DATA_TYPE of SQL_VARCHAR.                                                                                                                                                                                                                                                                                   |
|                                                     | Some applications such as MS Access97 use LVARCHAR instead of VARCHAR even for columns that are less than 256 bytes long. The minimum number of bytes transmitted for LVARCHAR is higher than for VARCHAR and many LVARCHAR columns can result in the rowset size exceeding the maximum. Enable this option only if your SQL_VARCHAR columns are less than 256 bytes in length. |
|                                                     | The default is: 0 (disabled).                                                                                                                                                                                                                                                                                                                                                   |
| <b>Report CHAR columns as wide CHAR columns</b>     | Causes SQLDescribeCol to report char columns as wide char columns. SQL_CHAR column is reported as SQL_WCHAR, SQL_VARCHAR as SQL_WVARCHAR and SQL_LONGVARCHAR column as SQL_WLONGVARCHAR                                                                                                                                                                                         |
|                                                     | The default is: 0 (disabled).                                                                                                                                                                                                                                                                                                                                                   |
| <b>Length in Chars for SQLGetDiag RecW</b>          | If enabled, the SQLGetDiagRecW API treats the BufferLength Parameter as the number of characters.<br>The default is: FALSE (disabled).                                                                                                                                                                                                                                          |
| <b>Leave Trailing Spaces</b>                        | If enabled, the driver preserves blank characters at the end of VARCHAR/LVARCHAR column results.<br>The default is: 0 (disabled).                                                                                                                                                                                                                                               |
| <b>Describe input parameters for SQL statements</b> | This is required particularly for BLOB/CLOB data types. If enabled, server will send the description of these data types which will be used by ODBC Driver. This option should only be enabled when needed. It should not be turned on all the time as that would cause more round trips between client and server.<br>The default is: 0 (disabled).                            |

9. To check your connection to the database server, click **Test Connection**.
10. Click **OK** to return to the ODBC Data Source Administrator dialog box and to update the DSN information in the appropriate files.

**Results**

When your application connects to this DSN, the values that you entered are the default entries for the DSN connection.

## Removing a DSN

Access the ODBC Data Source Administrator dialog box to remove a DSN.

### About this task

To remove a DSN:

1. Follow steps [1 on page 50](#) and [2 on page 50](#) from [Configuring a new user DSN or system DSN on page 50](#).
2. Click **Remove** in the ODBC Data Source Administrator dialog box.

The 32-bit ODBC Administrator dialog box opens.

3. Click **Yes** to remove the DSN and return to the ODBC Data Source Administrator dialog box.

## Reconfiguring an existing DSN

Access the ODBC Data Source Administrator dialog box to reconfigure an exiting user DSN.

### About this task

To reconfigure an existing DSN:

1. Follow steps [1 on page 50](#) and [2 on page 50](#) from [Configuring a new user DSN or system DSN on page 50](#)
2. Click **Configure** to display the HCL OneDB™ ODBC Driver Setup dialog box.

Enter the new configuration values in the corresponding text boxes and click **OK** to return to the ODBC Data Source Administrator dialog box.

### Results

After you complete these steps, you will connect to the DSN.

## Configuring a file DSN

Access the ODBC Data Source Administrator dialog box to configure a file DSN.

### About this task

To configure a file DSN:

1. Choose **Start > Settings > Control Panel**.
2. Double-click the ODBC icon to open the ODBC Data Source Administrator dialog box.
3. Click the **File DSN** tab to display the **File DSN** page.

Choose the File DSN option to allow access to the DSN to all users on a network. For a description of values, see [Table 2: Required DSN values on page 49](#) and [Table 3: Optional DSN values on page 49](#).

4. Click **Add**.

The Create New Data Source wizard opens.

5. Select **IBM INFORMIX ODBC Driver** from the driver list and click **Next** to display the Create New Data Source Setup wizard, which contains a file data source text box.
6. If you know the name of the data source file, type the name into the text box, click **Next** to display the completed Create New Data Source wizard, and go to step [9 on page 56](#)

If you do not know the name of the file, click **Browse** to display the Save As dialog box and enter the values, as the following example shows:

- File Name: File\_DSN
- Save as type: ODBC File Data Sources

Select a file name or type a file name in the **File\_name** text box.

7. Click **Save** to display the Create New Data Source wizard, which displays information about the data source name.
8. Click **Next** to display the completed Create New Data Source wizard.
9. Click **Finish** to display the HCL® OneDB® Connect dialog box.

For a description of the values, see [Table 2: Required DSN values on page 49](#) and [Table 3: Optional DSN values on page 49](#). For **Advanced** tab values, see [Configuring a new user DSN or system DSN on page 50](#).

10. Click **OK** to save the values and display the ODBC Data Source Administrator dialog box.

The name of the data file that you chose or typed in step [6 on page 56](#) is displayed in the text box.

## Results

After you add or change DSN-configuration information, the driver updates the appropriate Windows™ registry to reflect the specified values. To be compatible with other HCL® OneDB® connectivity products, the driver stores the DSN-configuration information in the Windows™ registry.

## Creating logs of calls to the drivers

Access the **Tracing** page to create logs of calls to the drivers.

### About this task

To create logs of calls to the drivers:

1. Click the **Tracing** tab to display the **Tracing** page.
2. Select **Start Tracing Now** to turn on tracing.
3. To enter an existing log file, click **Browse** to display the Select ODBC Log File dialog box.
4. Enter the file name in the **File\_name** text box and click **Save** to return to the **Tracing** page.
5. To select a custom trace dynamic link library (DLL), click **Select DLL** to display the Select a custom trace dll dialog box, and enter the values, as the following example shows:

#### Example

- File name: test2\_dsn
- Files of type: Dynamic link libraries (\*.dll)



Choose a file or type a file name in the **File\_name** text box.

6. Click **Open** to display the Tracing page.
7. Click **OK** to save the changes.

## Connection string keywords that make a connection

Use connection string keywords to make a connection with or without DSN and with the DRIVER keywords.

The following table lists the connection string keywords that can be used in making a connection:

| Keyword                    | Short version |
|----------------------------|---------------|
| CLIENT_LOCALE              | CLOC          |
| CONNECTDATABASE            | CONDB         |
| CURSORBEHAVIOR             | CURB          |
| DATABASE                   | DB            |
| DB_LOCALE                  | DLOC          |
| DESCRIBEDECIMALFLOATPOINT  | DDFP          |
| DESCRIPTION                | DESC          |
| DONOTUSELVARCHAR           | DNL           |
| DRIVER                     | DRIVER        |
| DSN                        | DSN           |
| ENABLEINSERTCURSORS        | ICUR          |
| ENABLESCROLLABLECURSORS    | SCUR          |
| EXCLUSIVE                  | XCL           |
| FETCHBUFFERSIZE            | FBC           |
| FILEDSN                    | FILEDSN       |
| HOST                       | HOST          |
| NEEDODBCTYPESONLY          | ODTYP         |
| OPTIMIZEAUTOCOMMIT         | OAC           |
| OPTIONS                    | OPT           |
| OPTOFC                     | OPTOFC        |
| PWD                        | PWD           |
| REPORTCHARCOLASWIDECHARCOL | RCWC          |
| REPORTKEYSETCURSORS        | RKC           |

| Keyword           | Short version |
|-------------------|---------------|
| SAVEFILE          | SAVEFILE      |
| SERVER            | SRVR          |
| SERVICE           | SERV          |
| SINGLETHREADED    | SINGLETH      |
| SKIPPARSING       | SKIPP         |
| TRANSLATIONDLL    | TDLL          |
| TRANSLATIONOPTION | TOPT          |
| UID               | UID           |
| UPDATE_DESCRIBE   | UPDESC        |

## DSN migration tool

You can use the DSN migration tool by creating a text file with an `.ini` extension.

To use the DSN migration tool, `dsnigrate.exe`, that accompanies HCL OneDB™ ODBC Driver, create a text file with the extension `.ini`; and then type the names and values of the DSNs that you want to migrate or restore. The migration log file is located in `%ONEDB_HOME%\release\dsnMigr.log`. The restore information is located in `%ONEDB_HOME%\release\dsnMigr.sav`.

The following restrictions apply:

- A user DSN can be used or migrated only by the user who created that DSN.
- A system DSN can be used by all users of the system.
- A file DSN requires write privileges to the file.

## Setting up and using the DSN migration tool

Set up and use the DSN migration tool with a text editor to create a text file.

### About this task

To set up and use the DSN migration tool:

1. Open a text editor and create a text file with an `.ini` extension.
2. Create a section in the file for each type of DSN (user, system, and file) to be modified.
3. On a separate line in each section, specify your DSNs by using the following format:

```
DSNname=drivername
```

*drivername* must be `IBM INFORMIX ODBC DRIVER`

4. To run `dsnigrate.exe`, use the following command:

```
dsnMigrate -f filename
```

where *filename* is the name of the text file created in [step 1 on page 58](#)

## DSN migration tool examples

The DSN migration tool examples illustrate various DSNs migrated to the HCL OneDB™ ODBC Driver.

In the following example a DSN named **Test1** migrates to `IBM INFORMIX ODBC DRIVER`, and a DSN named **Test2** migrates to `IBM INFORMIX ODBC DRIVER`. Both DSNs are restricted to the user who created them.

```
[User DSN]
Test1=IBM INFORMIX ODBC DRIVER
Test2=IBM INFORMIX ODBC DRIVER
```

In the second example a DSN named **Test3** migrates to `IBM INFORMIX ODBC DRIVER`, and a DSN named **Test4** migrates to its original DSN. Both DSNs can be used by all users of the system. The user who migrates these system DSNs must have permission to modify ODBC system DSN registry entries.

```
[System DSN]
Test3=IBM INFORMIX ODBC DRIVER
Test4=restore
```

In the third example, two file DSNs named **test5.dsn** and **test6.dsn** migrate to `IBM INFORMIX ODBC DRIVER`.

```
[File DSN]
C:\Program Files\ODBC\Data Sources\test5.dsn=IBM INFORMIX ODBC DRIVER
C:\Program Files\ODBC\Data Sources\test6.dsn=IBM INFORMIX ODBC DRIVER
```

## Data types

These topics contain information about the data types that are supported by HCL OneDB™ ODBC Driver.

### Data types

HCL OneDB™ ODBC Driver supports five different data types.

The following table describes the data types that HCL OneDB™ ODBC Driver supports.

| Data type                            | Description                                                 | Example          |
|--------------------------------------|-------------------------------------------------------------|------------------|
| HCL OneDB™ SQL data type             | Data types that your HCL OneDB™ database server uses        | CHAR( <i>n</i> ) |
| HCL OneDB™ ODBC Driver SQL data type | Data types that correspond to the HCL OneDB™ SQL data types | SQL_CHAR         |
| Standard C data type                 | Data types that your C compiler defines                     | unsigned char    |
| HCL OneDB™ ODBC Driver typedef       | Typedefs that correspond to the standard C data types       | UCHAR            |
| HCL OneDB™ ODBC Driver C data type   | Data types that correspond to the standard C data types     | SQL_C_CHAR       |

## SQL data types


HCL OneDB™ database server uses SQL data types.


For detailed information about the HCL OneDB™ SQL data types, see *HCL OneDB™ Guide to SQL: Reference*, *HCL OneDB™ Guide to SQL: Tutorial*, and *HCL OneDB™ User-Defined Routines and Data Types Developer's Guide*.

## Standard SQL data types

Standard HCL OneDB™ SQL data types have corresponding HCL OneDB™ ODBC Driver data types.

The following table lists the standard HCL OneDB™ SQL data types and their corresponding HCL OneDB™ ODBC Driver data types.

| HCL OneDB™ SQL data type        | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                                                                                                             |
|---------------------------------|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BIGINT                          | SQL_INFX_BIGINT                                 | Signed numeric value with precision 10, scale 0, and range $n$<br>$((2^{63} - 1) n 2^{63} 1)$                                                                                           |
| BIGSERIAL                       | SQL_INFX_BIGINT                                 | Sequential positive integers to $2^{63} 1)$                                                                                                                                             |
| BOOLEAN                         | SQL_BIT                                         | t' or f'                                                                                                                                                                                |
| BYTE                            | SQL_LONGVARBINARY                               | Binary data of variable length                                                                                                                                                          |
| CHAR( $n$ ), CHARACTER( $n$ )   | SQL_CHAR                                        | Character string of fixed length $n$<br>$(1 n 32,767)$                                                                                                                                  |
| CHARACTER VARYING( $m, r$ )     | SQL_VARCHAR                                     | Character string of variable length with maximum length $m$ ( $1 m 255$ ) and minimum amount of reserved space $r$ ( $0 r < m$ )                                                        |
| DATE                            | SQL_DATE                                        | Calendar date                                                                                                                                                                           |
| DATETIME                        | SQL_TIMESTAMP                                   | Calendar date and time of day                                                                                                                                                           |
| DEC( $p,s$ ), DECIMAL( $p, s$ ) | SQL_DECIMAL                                     | Signed numeric value with precision $p$ and scale $s$ :<br>$(1 p 32; 0 s p)$<br> <b>Important:</b> |

| HCL OneDB™ SQL data type | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                          |                                                 |  <ul style="list-style-type: none"> <li>• If you use a value for scale &gt; 14, inconsistencies in rounding are possible.</li> <li>• When a DECIMAL column contains floating-point data, HCL OneDB™ ODBC reports the column's scale as 255. This is to differentiate from fixed-point data, which has a maximum scale of 32. The DataDirect ODBC Driver always returns a scale of zero.</li> </ul> |
| DOUBLE PRECISION         | SQL_DOUBLE                                      | Signed numeric value with the same characteristics as the standard C <b>double</b> data type                                                                                                                                                                                                                                                                                                                                                                                          |
| FLOAT                    | SQL_DOUBLE                                      | Signed numeric value with the same characteristics as the standard C <b>double</b> data type                                                                                                                                                                                                                                                                                                                                                                                          |
| IDSSECURITYLABEL         |                                                 | Built-in DISTINCT OF VARCHAR(128) data type; use is restricted to label-based access control                                                                                                                                                                                                                                                                                                                                                                                          |
| INT, INTEGER             | SQL_INTEGER                                     | Signed numeric value with precision 10, scale 0, and range $n$<br><br>$(-2,147,483,647 \text{ } n \text{ } 2,147,483,647)$                                                                                                                                                                                                                                                                                                                                                            |
| INT8                     | SQL_BIGINT                                      | Signed numeric value with precision 10, scale 0, and range $n$<br><br>$((2^{63} - 1) \text{ } n \text{ } 2^{63} - 1)$                                                                                                                                                                                                                                                                                                                                                                 |
| INTERVAL MONTH( $p$ )    | SQL_INTERVAL_MONTH                              | Number of months between two dates; $p$ is the interval leading precision.                                                                                                                                                                                                                                                                                                                                                                                                            |

| HCL OneDB™ SQL data type                        | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                                                                             |
|-------------------------------------------------|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| INTERVAL YEAR( <i>p</i> )                       | SQL_INTERVAL_YEAR                               | Number of years and months between two dates; <i>p</i> is the interval leading precision.                                                               |
| INTERVAL YEAR( <i>p</i> ) TO MONTH              | SQL_INTERVAL_YEAR_TO_MONTH                      | Number of years and months between two dates; <i>p</i> is the interval leading precision.                                                               |
| INTERVAL DAY( <i>p</i> )                        | SQL_INTERVAL_DAY                                | Number of days between two dates; <i>p</i> is the interval leading precision.                                                                           |
| INTERVAL HOUR( <i>p</i> )                       | SQL_INTERVAL_HOUR                               | Number of hours between two date times; <i>p</i> is the interval leading precision.                                                                     |
| INTERVAL MINUTE( <i>p</i> )                     | SQL_INTERVAL_MINUTE                             | Number of minutes between two date/times; <i>p</i> is the interval leading precision.                                                                   |
| INTERVAL SECOND( <i>p,q</i> )                   | SQL_INTERVAL_SECOND                             | Number of seconds between two date/times; <i>p</i> is the interval leading precision and <i>q</i> is the interval seconds precision.                    |
| INTERVAL DAY( <i>p</i> ) TO HOUR                | SQL_INTERVAL_DAY_TO_HOUR                        | Number of days/hours between two date/times; <i>p</i> is the interval leading precision.                                                                |
| INTERVAL DAY( <i>p</i> ) TO MINUTE              | SQL_INTERVAL_DAY_TO_MINUTE                      | Number of days/hours/minutes between two date/times; <i>p</i> is the interval leading precision.                                                        |
| INTERVAL DAY( <i>p</i> ) TO SECOND( <i>q</i> )  | SQL_INTERVAL_DAY_TO_SECOND                      | Number of days/hours/minutes/seconds between two date/times; <i>p</i> is the interval leading precision and <i>q</i> is the interval seconds precision. |
| INTERVAL HOUR ( <i>p</i> ) TO MINUTE            | SQL_INTERVAL_HOUR_TO_MINUTE                     | Number of hours/minutes between two date/times; <i>p</i> is the interval leading precision.                                                             |
| INTERVAL HOUR( <i>p</i> ) TO SECOND( <i>q</i> ) | SQL_INTERVAL_HOUR_TO_SECOND                     | Number of hours/minutes/seconds between two date/times; <i>p</i> is the interval leading precision and <i>q</i> is the interval seconds precision.      |

| HCL OneDB™ SQL data type                          | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INTERVAL MINUTE( <i>p</i> ) TO SECOND( <i>q</i> ) | SQL_INTERVAL_MINUTE_TO_SECOND                   | Number of minutes/seconds between two date/times; <i>p</i> is the interval leading precision and <i>q</i> is the interval seconds precision.                                                                                                                                                                                                                       |
| LVARCHAR                                          | SQL_VARCHAR                                     | Character string of variable length with length <i>l</i><br><br>(255 / 32,000)<br><br>When connecting to HCL OneDB™ 10.0 servers with the ODBC driver, the SQLDescribeCol, SQLColAttributes & SQLDescribeParam APIs report the length mentioned during creation of the LVARCHAR column. If no length was mentioned during creation, length defaults to 2048 bytes. |
| MONEY( <i>p</i> , <i>s</i> )                      | SQL_DECIMAL                                     | Signed numeric value with precision <i>p</i> and scale <i>s</i><br><br>(1 <i>p</i> 32; 0 <i>s</i> <i>p</i> )                                                                                                                                                                                                                                                       |
| NUMERIC                                           | SQL_NUMERIC                                     | Signed, exact, numeric value with precision <i>p</i> and scale <i>s</i><br><br>(1 <i>p</i> 15; 0 <i>s</i> <i>p</i> )                                                                                                                                                                                                                                               |
| REAL                                              | SQL_REAL                                        | Signed numeric value with the same characteristics as the standard C <b>float</b> data type                                                                                                                                                                                                                                                                        |
| SERIAL                                            | SQL_INTEGER                                     | Sequential INTEGER                                                                                                                                                                                                                                                                                                                                                 |
| SERIAL8                                           | SQL_BIGINT                                      | Sequential INT8                                                                                                                                                                                                                                                                                                                                                    |
| SMALLFLOAT                                        | SQL_REAL                                        | Signed numeric value with the same characteristics as the standard C <b>float</b> data type                                                                                                                                                                                                                                                                        |
| SMALLINT                                          | SQL_SMALLINT                                    | Signed numeric value with precision 5, scale 0, and range <i>n</i><br><br>(32,767 <i>n</i> 32,767)                                                                                                                                                                                                                                                                 |

| HCL OneDB™ SQL data type       | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                                                                              |
|--------------------------------|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| TEXT                           | SQL_LONGVARCHAR                                 | Character string of variable length                                                                                                                      |
| VARCHAR( <i>m</i> , <i>r</i> ) | SQL_VARCHAR                                     | Character string of variable length with maximum length <i>m</i> (1 <i>m</i> 255) and minimum amount of reserved space <i>r</i> (0 <i>r</i> < <i>m</i> ) |

## Visual Basic client-side cursors

When you use Visual Basic client-side cursors to perform rowset update related operations with using CHAR or LVARCHAR columns that have lengths greater than or equal to 16,385, the HCL OneDB™ ODBC Driver might return an error.

Visual Basic sends the SQL data type to SQLBindParameter as SQL\_LONGVARCHAR instead of SQL\_VARCHAR when the length is greater than or equal to 16,385. HCL OneDB™ ODBC Driver maps SQL\_LONGVARCHAR to TEXT data type. Therefore, applications might see the error:

```
[Informix][Informix ODBC Driver]No cast from text to lvarchar
```

or

```
[Informix][Informix ODBC Driver]Illegal attempt to use Text/Byte host variable.
```

## Additional SQL data types for GLS

Additional SQL data types for GLS have corresponding HCL OneDB™ ODBC Driver data types.

The following table lists the additional HCL OneDB™ SQL data types for GLS and their corresponding HCL OneDB™ ODBC Driver data types. HCL OneDB™ ODBC driver does not provide full GLS support. For more information about GLS, see the *HCL OneDB™ GLS User's Guide*.

| HCL OneDB™ SQL data type        | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                                                                                                            |
|---------------------------------|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NCHAR( <i>n</i> )               | SQL_CHAR                                        | Character string of fixed length <i>n</i> (1 <i>n</i> 32,767). Collation depends on locale.                                                                                            |
| NVARCHAR( <i>m</i> , <i>r</i> ) | SQL_VARCHAR                                     | Character string of variable length with maximum length <i>m</i> (1 <i>m</i> 255) and minimum amount of reserved space <i>r</i> (0 <i>r</i> < <i>m</i> ). Collation depends on locale. |

## Additional SQL data types for HCL OneDB™

Additional HCL OneDB™ SQL data types for HCL OneDB™ have corresponding HCL OneDB™ ODBC Driver data types.



The following table lists the additional HCL OneDB™ SQL data types for HCL OneDB™ and their corresponding HCL OneDB™ ODBC Driver data types. To use the HCL OneDB™ ODBC driver SQL data types for HCL OneDB™, include `infxcli.h`.

| HCL OneDB™ SQL data type          | HCL OneDB™ ODBC driver SQL data type (fSqlType) | Description                                                                                               |
|-----------------------------------|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| Collection (LIST, MULTISSET, SET) | Any HCL OneDB™ ODBC driver SQL data type        | Composite value that consists of one or more elements, where each element has the same data type.         |
| DISTINCT                          | Any HCL OneDB™ ODBC driver SQL data type        | UDT that is stored the same way as its source data type but has different casts and functions             |
| OPAQUE (fixed)                    | SQL_INFX_UDT_FIXED                              | Fixed-length UDT with an internal structure that has the same size for all possible values                |
| OPAQUE (varying)                  | SQL_INFX_UDT_VARYING                            | Variable-length UDT with an internal structure that can have a different size for each different value    |
| Row (Named row, unnamed row)      | Any HCL OneDB™ ODBC Driver SQL data type        | Composite value that consists of one or more elements, where each element can have a different data type. |
| Smart large object (BLOB or CLOB) | SQL_IFMX_UDT_BLOB<br>SQL_IFMX_UDT_CLOB          | Large object that is stored in an sbospace on disk and is recoverable.                                    |

## Precision, scale, length, and display size

The functions that get and set precision, scale, length, and display size for SQL values have size limitations for their input arguments.

Therefore, these values are limited to the size of an SDWORD that has a maximum value of 2,147,483,647. The following table describes these values.

| Value     | Description for a numeric data type                                                                                                                                                 | Description for a non-numeric data type            |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| Precision | Maximum number of digits.                                                                                                                                                           | Either the maximum length or the specified length. |
| Scale     | Maximum number of digits after the decimal point.<br>For floating point values, the scale is undefined because the number of digits to the right of the decimal point is not fixed. | Not applicable.                                    |

| Value        | Description for a numeric data type                                                                     | Description for a non-numeric data type                                                                                                                        |
|--------------|---------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Length       | Maximum number of bytes that a function returns when a value is transferred to its default C data type. | Maximum number of bytes that a function returns when a value is transferred to its default C data type. The length does not include the NULL termination byte. |
| Display size | Maximum number of bytes needed to display data in character form.                                       | Maximum number of bytes needed to display data in character form.                                                                                              |

## Standard SQL data types

View the values for the precision, scale, length, and display size for standard HCL OneDB™ ODBC Driver SQL data types.

The following table describes the precision, scale, length, and display size for the standard HCL OneDB™ ODBC Driver SQL data types.

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                     |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQL_BIGINT                                      | <p><b>Precision</b></p> <p>19. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>0. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>8 bytes</p> <p><b>Display size</b></p> <p>20 digits. One digit is for the sign.</p> |
| SQL_BIT                                         | <p><b>Precision</b></p> <p>1. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>0. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>1 byte</p> <p><b>Display size</b></p> <p>1 digit</p>                                 |
| SQL_CHAR                                        | <p><b>Precision</b></p> <p>Same as the length</p>                                                                                                                                                                                                                                                                               |

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                 | <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>The specified length. For example, the length of CHAR(10) is 10 bytes.</p> <p><b>Display size</b></p> <p>Same as the length.</p>                                                                                                                                                                                                                                              |
| SQL_DATE                                        | <p><b>Precision</b></p> <p>10. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>6 bytes</p> <p><b>Display size</b></p> <p>10 digits in the format <code>yyyy-mm-dd</code>.</p>                                                                                                                                                                   |
| SQL_DECIMAL                                     | <p><b>Precision</b></p> <p>The specified precision. For example, the precision of DECIMAL (12, 3) is 12.</p> <p><b>Scale</b></p> <p>The specified scale. For example, the scale of DECIMAL(12, 3) is 3.</p> <p><b>Length</b></p> <p>The specified precision plus 2. For example, the length of DECIMAL(12, 3) is 14 bytes. The two additional bytes are used for the sign and the decimal points because functions return this data type as a character string.</p> <p><b>Display size</b></p> <p>Same as the length.</p> |
| SQL_DOUBLE                                      | <p><b>Precision</b></p> <p>15. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p>                                                                                                                                                                                                                                                                                                                                                                                                              |

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                 | <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>8 bytes</p> <p><b>Display size</b></p> <p>22 digits. The digits are for a sign, 15 numeric characters, a decimal point, the letter E, another sign, and 2 more numeric characters.</p>                                                                                                           |
| SQL_INTEGER                                     | <p><b>Precision</b></p> <p>10. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>0. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>4 bytes</p> <p><b>Display size</b></p> <p>11 digits. One digit is for the sign.</p>                                                                                                              |
| SQL_LONGVARBINARY                               | <p><b>Precision</b></p> <p>Same as the length.</p> <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>The maximum length. If a function cannot determine the maximum length, it returns SQL_NO_TOTAL.</p> <p><b>Display size</b></p> <p>The maximum length times 2. If a function cannot determine the maximum length, it returns SQL_NO_TOTAL.</p> |
| SQL_LONGVARCHAR                                 | <p><b>Precision</b></p> <p>Same as the length.</p>                                                                                                                                                                                                                                                                                                                                                                                           |

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                 | <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>The maximum length. If a function cannot determine the maximum length, it returns SQL_NO_TOTAL.</p> <p><b>Display size</b></p> <p>Same as the length.</p>                                                                                                                                         |
| SQL_REAL                                        | <p><b>Precision</b></p> <p>7. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>4 bytes</p> <p><b>Display size</b></p> <p>13 digits. The digits are for a sign, 7 numeric characters, a decimal point, the letter E, another sign, and 2 more numeric characters.</p> |
| SQL_SMALLINT                                    | <p><b>Precision</b></p> <p>5. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>0. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>2 bytes</p> <p><b>Display size</b></p> <p>6 digits. One digit is for the sign.</p>                                                                                                                 |
| SQL_TIMESTAMP                                   | <p><b>Precision</b></p> <p>8. SQLBindParameter ignores the value of <i>cbColDef</i> for this data type.</p> <p><b>Scale</b></p> <p>The number of digits in the FRACTION field.</p>                                                                                                                                                                                                                                                            |

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                 | <p><b>Length</b></p> <p>16 bytes</p> <p><b>Display size</b></p> <p>19 or more digits:</p> <ul style="list-style-type: none"> <li>• If the scale of the time stamp is 0: 19 digits in the format <code>yyyy-mm-dd hh:mm:ss</code>.</li> <li>• If the scale of the time stamp exceeds 0: 20 digits plus digits for the FRACTION field in the format <code>yyyy-mm-dd hh:mm:ss.f . . .</code></li> </ul> |
| SQL_VARCHAR                                     | <p><b>Precision</b></p> <p>Same as the length.</p> <p><b>Scale</b></p> <p>Not applicable. SQLBindParameter ignores the value of <i>ibScale</i> for this data type.</p> <p><b>Length</b></p> <p>The specified length. For example, the length of VARCHAR(10) is 10 bytes.</p> <p><b>Display size</b></p> <p>Same as the length.</p>                                                                    |

### Additional SQL data types for HCL OneDB™

View the values for the precision, scale, length, and display size for additional HCL OneDB™ ODBC Driver SQL data types.

The following table describes the precision, scale, length, and display size for the HCL OneDB™ ODBC Driver SQL data types for HCL OneDB™.

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                               |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQL_IFMX_UDT_BLOB                               | <p><b>Precision</b></p> <p>Variable value. To determine this value, call a function that returns the precision for a column.</p> <p><b>Scale</b></p> <p>Not applicable. A function that returns the scale for a column returns -1 for this data type.</p> |

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                 | <p><b>Length</b></p> <p>Variable value. To determine this value, call a function that returns the length for a column.</p> <p><b>Display size</b></p> <p>Variable value. To determine this value, call a function that returns the display size for a column.</p>                                                                                                                                                                                                                                                           |
| SQL_IFMX_UDT_CLOB                               | <p><b>Precision</b></p> <p>Variable value. To determine this value, call a function that returns the precision for a column.</p> <p><b>Scale</b></p> <p>Not applicable. A function that returns the scale for a column returns -1 for this data type.</p> <p><b>Length</b></p> <p>Variable value. To determine this value, call a function that returns the length for a column.</p> <p><b>Display size</b></p> <p>Variable value. To determine this value, call a function that returns the display size for a column.</p> |
| SQL_INFX_UDT_FIXED                              | <p><b>Precision</b></p> <p>Variable value. To determine this value, call a function that returns the precision for a column.</p> <p><b>Scale</b></p> <p>Not applicable. A function that returns the scale for a column returns -1 for this data type.</p> <p><b>Length</b></p> <p>Variable value. To determine this value, call a function that returns the length for a column.</p> <p><b>Display size</b></p> <p>Variable value. To determine this value, call a function that returns the display size for a column.</p> |
| SQL_INFX_UDT_VARYING                            | <p><b>Precision</b></p> <p>Variable value. To determine this value, call a function that returns the precision for a column.</p>                                                                                                                                                                                                                                                                                                                                                                                            |

| HCL OneDB™ ODBC driver sql data type (fSqlType) | Description                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                 | <p><b>Scale</b></p> <p>Not applicable. A function that returns the scale for a column returns -1 for this data type.</p> <p><b>Length</b></p> <p>Variable value. To determine this value, call a function that returns the length for a column.</p> <p><b>Display size</b></p> <p>Variable value. To determine this value, call a function that returns the display size for a column.</p> |

## C data types

HCL OneDB™ ODBC Driver applications use C data types to store values that the application processes.

The following table describes the C data types that HCL OneDB™ ODBC Driver provides.



**Important:** String arguments in HCL OneDB™ ODBC driver functions are unsigned. Therefore, you need to cast a CString object as an unsigned string before you use it as an argument in the HCL OneDB™ ODBC driver function.

| Value          | HCL OneDB™ ODBC driver C data type (fCType) | HCL OneDB™ ODBC driver typedef | Standard C data type                                                    |
|----------------|---------------------------------------------|--------------------------------|-------------------------------------------------------------------------|
| Binary         | SQL_C_BINARY                                | UCHAR FAR *                    | unsigned char FAR *                                                     |
| Boolean        | SQL_C_BIT                                   | UCHAR                          | unsigned char                                                           |
| Character      | SQL_C_CHAR                                  | UCHAR FAR *                    | unsigned char FAR *                                                     |
| Wide Character | SQL_C_WCHAR                                 | WCHAR FAR *                    | wchar_t FAR *                                                           |
| Date           | SQL_C_DATE                                  | DATE_STRUCT                    | <pre>struct tagDATE_STRUCT{ SWORD year; UWORD month; UWORD day; }</pre> |
| Interval       | SQL_C_INTERVAL_YEAR                         | SQL_INTERVAL_STRUCT            | C Interval Structure                                                    |
|                | SQL_C_INTERVAL_MONTH                        | SQL_INTERVAL_STRUCT            | C Interval Structure                                                    |
|                | SQL_C_INTERVAL_DAY                          | SQL_INTERVAL_STRUCT            | C Interval Structure                                                    |
|                | SQL_C_INTERVAL_HOUR                         | SQL_INTERVAL_STRUCT            | C Interval Structure                                                    |
|                | SQL_C_INTERVAL_MINUTE                       | SQL_INTERVAL_STRUCT            | C Interval Structure                                                    |
|                | SQL_C_INTERVAL_SECOND                       | SQL_INTERVAL_STRUCT            | C Interval Structure                                                    |



| Value   | HCL OneDB™ ODBC driver C data type (fCType) | HCL OneDB™ ODBC driver typedef | Standard C data type                                                                                                                                    |
|---------|---------------------------------------------|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | SQL_C_INTERVAL_YEAR_TO_MONTH                | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
|         | SQL_C_INTERVAL_DAY_TO_HOUR                  | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
|         | SQL_C_INTERVAL_DAY_TO_MINUTE                | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
|         | SQL_C_INTERVAL_DAY_TO_SECOND                | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
|         | SQL_C_INTERVAL_HOUR_TO_MINUTE               | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
|         | SQL_C_INTERVAL_HOUR_TO_SECOND               | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
|         | SQL_C_INTERVAL_MINUTE_TO_SECOND             | SQL_INTERVAL_STRUCT            | C Interval Structure                                                                                                                                    |
| Numeric | SQL_C_DOUBLE                                | SDOUBLE                        | signed double                                                                                                                                           |
|         | SQL_C_FLOAT                                 | SFLOAT                         | signed float                                                                                                                                            |
|         | SQL_C_LONG                                  | SDWORD                         | signed long int                                                                                                                                         |
|         | SQL_C_NUMERIC                               | SQL_NUMERIC_STRUCT             | <pre> struct tag SQL_NUMERIC_STRUCT { SQLCHAR precision; SQLSCHAR scale; SQLCHAR sign; SQLCHAR val [SQL_MAX_ NUMERIC_LEN]; }SQL_NUMERIC_ STRUCT; </pre> |
|         | SQL_C_SHORT                                 | SWORD                          | signed short int                                                                                                                                        |
|         | SQL_C_SLONG                                 | SDWORD                         | signed long int                                                                                                                                         |
|         | SQL_C_SSHORT                                | SWORD                          | signed short int                                                                                                                                        |
|         | SQL_C_STINYINT                              | SCHAR                          | signed char                                                                                                                                             |
|         | SQL_C_TINYINT                               | SCHAR                          | signed char                                                                                                                                             |
|         | SQL_C_ULONG                                 | UDWORD                         | unsigned long int                                                                                                                                       |
|         | SQL_C_USHORT                                | UWORD                          | unsigned short int                                                                                                                                      |
|         | SQL_C_UTINYINT                              | UCHAR                          | unsigned char                                                                                                                                           |

| Value      | HCL OneDB™ ODBC driver C data type (fCType) | HCL OneDB™ ODBC driver typedef | Standard C data type                                                                                                                                               |
|------------|---------------------------------------------|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Time stamp | <b>SQL_C_TIMESTAMP</b>                      | TIMESTAMP_STRUCT               | <pre>struct tagTIMESTAMP_STRUCT {     SWORD year;     UWORD month;     UWORD day;     UWORD hour;     UWORD minute;     UWORD second;     UDWORD fraction; }</pre> |

## C interval structure

Specify the C data type for the SQL interval data type by using a C interval structure.

The following structures specify the C data type for the SQL interval data type:

```
typedef struct tagSQL_INTERVAL_STRUCT
{
 SQLINTERVAL interval_type;
 SQLSMALLINT interval_sign;
 union
 {
 SQL_YEAR_MONTH_STRUCT year_month;
 SQL_DAY_SECOND_STRUCT day_second;
 } intval;
}SQLINTERVAL_STRUCT;
```

```
typedef enum
{
 SQL_IS_YEAR=1,
 SQL_IS_MONTH=2,
 SQL_IS_DAY=3,
 SQL_IS_HOUR=4,
 SQL_IS_MINUTE=5,
 SQL_IS_SECOND=6,
 SQL_IS_YEAR_TO_MONTH=7,
 SQL_IS_DAY_TO_HOUR=8,
 SQL_IS_DAY_TO_MINUTE=9,
 SQL_IS_DAY_TO_SECOND=10,
 SQL_IS_HOUR_TO_MINUTE=11,
 SQL_IS_HOUR_TO_SECOND=12,
 SQL_IS_MINUTE_TO_SECOND=13,
}SQLINTERVAL;
```

```
typedef struct tagSQL_YEAR_MONTH
{
 SQLINTEGER year;
 SQLINTEGER month;
}SQL_YEAR_MONTH_STRUCT;
```

```
typedef struct tagSQL_DAY_SECOND
{
 SQLINTEGER day;
 SQLINTEGER hour;
 SQLINTEGER minute;
 SQLINTEGER second;
```

```
SQLINTEGER fraction;
}SQL_DAY_SECOND_STRUCT;
```

## Transfer data

Among data sources that use the same DBMS, you can safely transfer data in the internal form that a DBMS uses.

For a particular piece of data, the SQL data types must be the same in the source and target data sources. The C data type is **SQL\_C\_BINARY**.

When you call `SQLFetch`, `SQLExtendedFetch`, or `SQLGetData` to retrieve this data from a data source, HCL OneDB™ ODBC Driver retrieves the data and transfers it, without conversion, to a storage location of type **SQL\_C\_BINARY**. When you call `SQLExecute`, `SQLExecDirect`, or `SQLPutData` to send this data to a target data source, HCL OneDB™ ODBC Driver retrieves the data from the storage location and transfers it, without conversion, to the target data source.

The binary representation of `INT8`, `SERIAL8`, and `BIGSERIAL` data types is an array of two unsigned long integers followed by a short integer that indicates the sign field. The sign field is `1` for a positive value, `-1` for a negative value, or `0` for a null value.



**Important:** Applications that transfer any data (except binary data) in this manner are not interoperable among DBMSs.

## Report standard ODBC types

HCL OneDB™ ODBC Driver supports existing applications that support standard ODBC data types only. Check the DSN option **Report Standard ODBC Types** to turn on this behavior.

When an application sets this option, the driver sets the following behavior:

- Only Standard ODBC data types are reported for all the driver defined new data types.
- The data type access method for smart-large-object (LO) data can be accessed as `SQL_LONGVARCHAR` and `SQL_LONGVARBINARY`. In other words, `SQL_LONGVARCHAR` and `SQL_LONGVARBINARY` act like the simple large objects, byte, and text.
- The defaultUDTfetchtype is set to **SQL\_C\_CHAR**.

However, you can control each of the preceding behaviors individually as a connection or a statement level option. Use the following connection and statement level attributes:

- `SQL_INFX_ATTR_ODBC_TYPES_ONLY`
- `SQL_INFX_ATTR_LO_AUTOMATIC`
- `SQL_INFX_ATTR_DEFAULT_UDT_FETCH_TYPE`

Applications can use `SQLSetConnectAttr` and `SQLSetStmtAttr` to set and unset these values. (ODBC 2.x applications can use `SQLSetConnectOption` and `SQLSetStmtOption` equivalently.)

## SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY

Applications can set the SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY attribute to value SQL\_TRUE or SQL\_FALSE.

This attribute can be set and unset at connection and statement level. All the statements allocated under the same connection inherit this value. Alternatively each statement can change this attribute. By default this attribute is set to SQL\_FALSE.

An application can change the value of this attribute by using SQLSetConnectAttr and SQLSetStmtAttr (SQLSetConnectOption and SQLSetStmtOption in ODBC 2.x). Applications can retrieve the values set by using SQLGetConnectAttr and SQLGetStmtAttr (SQLGetConnectOption and SQLGetStmtOption in ODBC 2.x).

This attribute cannot be set to SQL\_TRUE when SQL\_INFX\_ATTR\_LO\_AUTOMATIC is set SQL\_FALSE. An error message is returned that reports the following message:

```
Attribute cannot be set. LoAutomatic should be ON to set this value
```

The application should first set the SQL\_INFX\_ATTR\_LO\_AUTOMATIC attribute to SQL\_TRUE and then set the attribute SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY to SQL\_TRUE.

## SQL\_INFX\_ATTR\_LO\_AUTOMATIC

Applications can set the SQL\_INFX\_ATTR\_LO\_AUTOMATIC attribute to value SQL\_TRUE or SQL\_FALSE.

This attribute can be set and unset at connection and statement level. All the statements allocated under the same connection inherit this value. Alternatively each statement can change this attribute. By default this attribute is set to SQL\_FALSE.

An application can change the value of this attribute by using SQLSetConnectAttr and SQLSetStmtAttr (SQLSetConnectOption and SQLSetStmtOption in ODBC 2.x). Applications can retrieve the values set by using SQLGetConnectAttr and SQLGetStmtAttr (SQLGetConnectOption and SQLGetStmtOption in ODBC 2.x).

The attribute SQL\_INFX\_ATTR\_LO\_AUTOMATIC cannot be set to SQL\_FALSE when SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY is set to SQL\_TRUE. An error message is returned that reports the following message:

```
Attribute cannot be set. ODBC types only should be OFF to set this value
```

Applications should first set the attribute SQL\_INFX\_ODBC\_TYPES\_ONLY to SQL\_FALSE and then set the attribute SQL\_INFX\_ATTR\_LO\_AUTOMATIC to SQL\_FALSE.

Applications would like to set attribute SQL\_INFX\_ATTR\_UPDATE\_DESCRIBE (this option can also be enabled using DSN and/or connection string. For more information, see [Connection string keywords that make a connection on page 57](#)) using SQLSetConnectAttr() API. By enabling this option, server will send the description of BLOB/CLOB data types which will be used by ODBC Driver. This option should only be enabled when needed. It should not be enabled all the time, otherwise it would cause more round trips between client and server.

## SQL\_INFX\_ATTR\_DEFAULT\_UDT\_FETCH\_TYPE

Applications can set the SQL\_INFX\_ATTR\_DEFAULT\_UDT\_FETCH\_TYPE attribute to **SQL\_C\_CHAR** or **SQL\_C\_BINARY** to set the default fetch type for UDTs.

The default value of this attribute is set depending on the following conditions:

- If the DSN setting for **Report Standard ODBC Types** is ON, the value of DefaultUDTFetchType is set to **SQL\_C\_CHAR**.
- If the DSN setting for **Report Standard ODBC Types** is OFF, the value of DefaultUDTFetchType is set to **SQL\_C\_BINARY**.
- If a user has set a registry key, the value of DefaultUDTFetchType is set to the value in the registry provided **Report Standard ODBC Types** is not set.

An application can change the value of this attribute by using SQLSetConnectAttr and SQLSetStmtAttr (SQLSetConnectOption and SQLSetStmtOption in ODBC 2.x). Applications can retrieve the values set by using SQLGetConnectAttr and SQLGetStmtAttr (SQLGetConnectOption and SQLGetStmtOption in ODBC 2.x).

Setting the **Report Standard ODBC Types** to ON always overrides DefaultUDTFetchType to **SQL\_C\_CHAR**.

## Report wide character columns

HCL OneDB™ servers do not support wide character data types.

When an application sets the **Report Char Columns as Wide Char Columns** option, the driver sets the following behavior:

- SQLDescribeCol reports char columns as wide char columns
- SQL\_CHAR column is reported as SQL\_WCHAR
- SQL\_VARCHAR column is reported as SQL\_WVARCHAR
- SQL\_LONGVARCHAR column is reported as SQL\_WLONGVARCHAR
- The default is 0: (disabled)

After setting the **Report Char Columns as Wide Char Columns** option, calls to SQLBindParameter with SQL data types have the following behavior:

- SQL\_WCHAR is mapped to SQL\_CHAR
- SQL\_WVARCHAR is mapped to SQL\_VARCHAR
- SQL\_WLONGVARCHAR is mapped to SQL\_LONGVARCHAR

## DSN settings for report standard ODBC data types

For UNIX™ and Windows™, you can add the new DSN option **NeedODBCTypesOnly**.

For UNIX™, add a new DSN option **NeedODBCTypesOnly** under your DSN setting in your `odbc.ini` file [default is 0]. For example:

```
[Informix9]
Driver=/informix/lib/cli/libthcli.so
Description=IBM Informix ODBC Driver
```

```
...
NeedODBCTypesOnly=1
```

For Windows™, check this option under the **Advanced** tab of the ODBC Administration for HCL OneDB™ Driver DSN [default is 0].

The following table shows how the HCL OneDB™ data types map to the standard ODBC data types.

**Table 4. HCL OneDB™ and ODBC data type mapping**

| HCL OneDB™ | ODBC                          |
|------------|-------------------------------|
| Bigint     | SQL_BIGINT                    |
| Bigserial  | SQL_BIGINT                    |
| Blob       | SQL_LONGVARBINARY             |
| Boolean    | SQL_BIT                       |
| Clob       | SQL_LONGVARCHAR               |
| Int8       | SQL_BIGINT                    |
| Lvarchar   | SQL_VARCHAR                   |
| Serial8    | SQL_BIGINT                    |
| Multiset   | SQL_C_CHAR or<br>SQL_C_BINARY |
| Set        | SQL_C_CHAR or<br>SQL_C_BINARY |
| List       | SQL_C_CHAR or<br>SQL_C_BINARY |
| Row        | SQL_C_CHAR or<br>SQL_C_BINARY |



**Important:**



- For multiset, set, row, and list data types, the data type is mapped to the defaultUDTFetchType attribute set (**SQL\_C\_CHAR** or **SQL\_C\_BINARY**).
- To enable SQL\_BIGINT to work correctly with SQLBindCol and SQLBindParameter, you must use SQL\_C\_UBIGINT (which has a supported data range of 8 byte unsigned integer) and not SQL\_C\_LONG (which has a supported data range of 4 byte integer).

## Convert data

The word *convert* is used in this section in a broad sense; it includes the transfer of data from one storage location to another without a conversion in data type.

## Standard conversions

Standard conversions exist between the HCL OneDB™ SQL data types and the HCL OneDB™ ODBC Driver C data types.

Only HCL OneDB™ can convert data to **SQL\_C\_BIT**.

The HCL OneDB™ ODBC driver C data types, **SQL\_C\_BINARY**, **SQL\_C\_CHAR**, and **SQL\_C\_WCHAR**, support conversion between all HCL OneDB™ SQL data types listed in the following tables.

The following tables show the supported conversions between the HCL OneDB™ SQL data types and the HCL OneDB™ ODBC Driver C data types.

**Table 5. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

| SQL data type     | ODBC driver C data types (target type) |            |              |             |
|-------------------|----------------------------------------|------------|--------------|-------------|
|                   | SQL_C_BIT                              | SQL_C_DATE | SQL_C_DOUBLE | SQL_C_FLOAT |
| BOOLEAN           | yes                                    | no         | no           | no          |
| CHAR, CHARACTER   | yes                                    | no         | yes          | yes         |
| CHARACTER VARYING | yes                                    | no         | yes          | yes         |
| DATE              | no                                     | yes        | no           | no          |
| DATETIME          | no                                     | yes        | no           | no          |
| DEC, DECIMAL      | yes                                    | no         | yes          | yes         |
| DOUBLE PRECISION  | no                                     | no         | yes          | yes         |
| FLOAT             | no                                     | no         | yes          | yes         |
| INT, INTEGER      | yes                                    | no         | yes          | yes         |

**Table 5. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

(continued)

| SQL data type | ODBC driver C data types (target type) |            |              |             |
|---------------|----------------------------------------|------------|--------------|-------------|
|               | SQL_C_BIT                              | SQL_C_DATE | SQL_C_DOUBLE | SQL_C_FLOAT |
| INT8          | no                                     | no         | no           | no          |
| LVARCHAR      | yes                                    | yes        | no           | yes         |
| MONEY         | no                                     | yes        | yes          | yes         |
| NUMERIC       | no                                     | yes        | yes          | yes         |
| REAL          | no                                     | yes        | yes          | yes         |
| SERIAL        | no                                     | yes        | yes          | yes         |
| SMALLFLOAT    | yes                                    | no         | yes          | yes         |
| SMALLINT      | yes                                    | no         | yes          | yes         |
| TEXT          | yes                                    | yes        | yes          | yes         |
| VARCHAR       | yes                                    | yes        | yes          | yes         |

**Table 6. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

| SQL data type     | ODBC driver C data types (target type) |               |             |             |
|-------------------|----------------------------------------|---------------|-------------|-------------|
|                   | SQL_C_LONG                             | SQL_C_NUMERIC | SQL_C_SHORT | SQL_C_SLONG |
| BIGINT            | yes                                    | yes           | no          | yes         |
| BIGSERIAL         | yes                                    | yes           | yes         | yes         |
| BYTE              | no                                     | no            | no          | no          |
| CHAR, CHARACTER   | yes                                    | yes           | yes         | yes         |
| CHARACTER VARYING | yes                                    | yes           | yes         | yes         |
| DEC, DECIMAL      | yes                                    | yes           | yes         | yes         |
| DOUBLE PRECISION  | yes                                    | yes           | yes         | yes         |
| FLOAT             | yes                                    | yes           | yes         | yes         |



**Table 6. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

(continued)

| SQL data type | ODBC driver C data types (target type) |               |             |             |
|---------------|----------------------------------------|---------------|-------------|-------------|
|               | SQL_C_LONG                             | SQL_C_NUMERIC | SQL_C_SHORT | SQL_C_SLONG |
| INT, INTEGER  | yes                                    | yes           | yes         | yes         |
| INT8          | yes                                    | yes           | no          | yes         |
| LVARCHAR      | yes                                    | no            | yes         | yes         |
| MONEY         | yes                                    | yes           | yes         | yes         |
| NUMERIC       | yes                                    | yes           | yes         | yes         |
| REAL          | yes                                    | yes           | yes         | yes         |
| SERIAL        | yes                                    | no            | yes         | yes         |
| SERIAL8       | yes                                    | yes           | yes         | yes         |
| SMALLFLOAT    | yes                                    | yes           | yes         | yes         |
| SMALLINT      | yes                                    | yes           | yes         | yes         |
| TEXT          | yes                                    | yes           | yes         | yes         |
| VARCHAR       | yes                                    | yes           | yes         | yes         |

**Table 7. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

| SQL data type     | ODBC driver C data types (target type) |                |                 |
|-------------------|----------------------------------------|----------------|-----------------|
|                   | SQL_C_SSHORT                           | SQL_C_STINYINT | SQL_C_TIMESTAMP |
| BIGINT            | yes                                    | no             | no              |
| BIGSERIAL         | yes                                    | no             | no              |
| CHAR, CHARACTER   | yes                                    | yes            | no              |
| CHARACTER VARYING | yes                                    | yes            | no              |
| DATE              | no                                     | no             | yes             |
| DATETIME          | no                                     | no             | yes             |
| DEC, DECIMAL      | yes                                    | yes            | no              |

**Table 7. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

(continued)

| SQL data type    | ODBC driver C data types (target type) |                |                 |
|------------------|----------------------------------------|----------------|-----------------|
|                  | SQL_C_SSHORT                           | SQL_C_STINYINT | SQL_C_TIMESTAMP |
| DOUBLE PRECISION | yes                                    | yes            | no              |
| FLOAT            | yes                                    | yes            | no              |
| INT, INTEGER     | yes                                    | yes            | no              |
| INT8             | yes                                    | no             | no              |
| LVARCHAR         | yes                                    | yes            | yes             |
| MONEY            | yes                                    | yes            | yes             |
| NUMERIC          | yes                                    | yes            | yes             |
| REAL             | yes                                    | yes            | yes             |
| SERIAL           | yes                                    | yes            | yes             |
| SERIAL8          | yes                                    | no             | no              |
| SMALLFLOAT       | yes                                    | yes            | no              |
| SMALLINT         | yes                                    | yes            | no              |
| TEXT             | yes                                    | yes            | yes             |
| VARCHAR          | yes                                    | yes            | yes             |

The ODBC driver C data type **SQL\_C\_ULONG** supports conversion between all the SQL data types listed in the following table.

**Table 8. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

| SQL data type        | ODBC driver C data types (target type) |              |                |
|----------------------|----------------------------------------|--------------|----------------|
|                      | SQL_C_TINYINT                          | SQL_C_USHORT | SQL_C_UTINYINT |
| BIGINT               | no                                     | no           | no             |
| BIGSERIAL            | no                                     | yes          | no             |
| CHAR, CHARACTER      | yes                                    | yes          | yes            |
| CHARACTER<br>VARYING | yes                                    | yes          | yes            |

**Table 8. Supported conversions between HCL OneDB™ SQL data types and ODBC Driver C data types**

(continued)

| SQL data type    | ODBC driver C data types (target type) |              |                |
|------------------|----------------------------------------|--------------|----------------|
|                  | SQL_C_TINYINT                          | SQL_C_USHORT | SQL_C_UTINYINT |
| DEC, DECIMAL     | yes                                    | yes          | yes            |
| DOUBLE PRECISION | yes                                    | yes          | yes            |
| FLOAT            | yes                                    | yes          | yes            |
| INT, INTEGER     | yes                                    | yes          | yes            |
| INT8             | no                                     | no           | no             |
| LVARCHAR         | yes                                    | yes          | yes            |
| MONEY            | yes                                    | yes          | yes            |
| NUMERIC          | yes                                    | yes          | yes            |
| REAL             | yes                                    | yes          | yes            |
| SERIAL           | yes                                    | yes          | yes            |
| SERIAL8          | no                                     | yes          | no             |
| SMALLFLOAT       | yes                                    | yes          | yes            |
| SMALLINT         | yes                                    | yes          | yes            |
| TEXT             | yes                                    | yes          | yes            |
| VARCHAR          | yes                                    | yes          | yes            |

## Additional conversions for GLS

There are supported conversions between the additional HCL OneDB™ SQL data types for GLS and the HCL OneDB™ ODBC Driver C data types.

Only HCL OneDB™ can convert data to **SQL\_C\_BIT**.

The HCL OneDB™ NCHAR and NVARCHAR SQL data types support conversion between the following ODBC driver C data types (fCType):

- **SQL\_C\_BINARY**
- **SQL\_C\_BIT**
- **SQL\_C\_CHAR**
- **SQL\_C\_DATE**
- **SQL\_C\_DOUBLE**

- **SQL\_C\_FLOAT**
- **SQL\_C\_LONG**
- **SQL\_C\_SHORT**
- **SQL\_C\_SLONG**
- **SQL\_C\_SSHORT**
- **SQL\_C\_STINYINT**
- **SQL\_C\_TIME STAMP**
- **SQL\_C\_TINYINT**
- **SQL\_C\_ULONG**
- **SQL\_C\_USHORT**
- **SQL\_C\_UTINYINT**

## Additional conversions for HCL OneDB™

There are supported conversions between the additional HCL OneDB™ SQL data types for HCL OneDB™ and the HCL OneDB™ ODBC Driver C data types.

The HCL OneDB™ SQL data types, Collection, DISTINCT, Row, and Smart large object, support conversions between the following HCL OneDB™ ODBC driver C data types (fCType):

- **SQL\_C\_BINARY**
- **SQL\_C\_BIT**
- **SQL\_C\_CHAR**
- **SQL\_C\_DATE**
- **SQL\_C\_DOUBLE**
- **SQL\_C\_FLOAT**
- **SQL\_C\_LONG**
- **SQL\_C\_SHORT**
- **SQL\_C\_SLONG**
- **SQL\_C\_SSHORT**
- **SQL\_C\_STINYINT**
- **SQL\_C\_TIMESTAMP**
- **SQL\_C\_TINYINT**
- **SQL\_C\_ULONG**
- **SQL\_C\_USHORT**
- **SQL\_C\_UTINYINT**

The HCL OneDB™ SQL data type OPAQUE supports conversion between the **SQL\_C\_BINARY** and **SQL\_C\_CHAR** ODBC driver C data types (fCType). Use **SQL\_C\_CHAR** to access an OPAQUE value in the external format as a string. Use **SQL\_C\_BINARY** to access an OPAQUE value in the internal binary format.

## Convert data from SQL to C

When you call `SQLExtendedFetch`, `SQLFetch`, or `SQLGetData`, HCL OneDB™ ODBC Driver retrieves data from a data source.

If necessary, HCL OneDB™ ODBC Driver converts the data from the source data type to the data type that the *TargetType* argument in SQLBindCol or the *fctype* argument in SQLGetData specifies. Finally, HCL OneDB™ ODBC Driver stores the data in the location pointed to by the *rgbValue* argument in SQLBindCol or SQLGetData.

The tables in the following sections describe how HCL OneDB™ ODBC Driver converts data that it retrieves from a data source. For a given HCL OneDB™ ODBC Driver SQL data type, the first column of the table lists the legal input values of the *TargetType* argument in SQLBindCol and the *fctype* argument in SQLGetData. The second column lists the outcomes of a test, often by using the *cbValueMax* argument specified in SQLBindCol or SQLGetData, which HCL OneDB™ ODBC Driver performs to determine whether it can convert the data. For each outcome, the third and fourth columns list the values of the *rgbValue* and *pcbValue* arguments specified in SQLBindCol or SQLGetData after HCL OneDB™ ODBC Driver tries to convert the data.

The last column lists the SQLSTATE returned for each outcome by SQLExtendedFetch, SQLFetch, or SQLGetData.

If the *TargetType* argument in SQLBindCol or the *fctype* argument in SQLGetData contains a value for the HCL OneDB™ ODBC Driver C data type that is not shown in the table for a given HCL OneDB™ ODBC Driver SQL data type, SQLExtendedFetch, SQLFetch, or SQLGetData returns SQLSTATE 07006 (Restricted data type attribute violation). If the *fctype* argument or the *TargetType* argument contains a value that specifies a conversion from a driver-specific SQL data type to the HCL OneDB™ ODBC Driver C data type and HCL OneDB™ ODBC Driver does not support this conversion, then SQLExtendedFetch, SQLFetch, or SQLGetData returns SQLSTATE S1C00 (Driver not capable).

Although the tables in this chapter do not show it, the *pcbValue* argument contains SQL\_NULL\_DATA when the SQL data value is null. When HCL OneDB™ ODBC Driver converts SQL data to character C data, the character count returned in *pcbValue* does not include the null-termination byte. If *rgbValue* is a null pointer, SQLBindCol or SQLGetData returns SQLSTATE S1009 (Invalid argument value).

The following terms and conventions are used in the tables:

**Length of data**

The number of bytes of C data that are available to return in *rgbValue*, regardless of whether the data is truncated before it returns to the application. For string data, this does not include the null-termination byte.

**Display size**

Total number of bytes that are needed to display the data in character format.

**Words in *italics***

Represent function arguments or elements of the HCL OneDB™ ODBC Driver SQL grammar.

## Default C data types

You can specify the **SQL\_C\_DEFAULT** for different functions so that HCL OneDB™ ODBC Driver uses the C data type.

If you specify **SQL\_C\_DEFAULT** for the *TargetType* argument in SQLBindCol, the *fctype* argument in SQLGetData, or the *ValueType* argument in SQLBindParameter, HCL OneDB™ ODBC Driver uses the C data type of the output or input buffer for the SQL data type of the column or parameter to which the buffer is bound.

## Standard default C data types

There is default C data type for each HCL OneDB™ ODBC Driver SQL data type.

For each HCL OneDB™ ODBC Driver SQL data type, the following table shows the default C data type.

| <b>HCL OneDB™ ODBC driver SQL data type (fSqlType)</b> | <b>Default HCL OneDB™ ODBC driver C data type (fCType)</b> |
|--------------------------------------------------------|------------------------------------------------------------|
| SQL_BIGINT                                             | SQL_C_CHAR                                                 |
| SQL_BIT                                                | SQL_C_BITS                                                 |
| SQL_CHAR                                               | SQL_C_CHAR                                                 |
| SQL_DATE                                               | SQL_C_DATE                                                 |
| SQL_DECIMAL                                            | SQL_C_CHAR                                                 |
| SQL_DOUBLE                                             | SQL_C_DOUBLE                                               |
| SQL_INTEGER                                            | SQL_C_SLONG                                                |
| SQL_LONGVARBINARY                                      | SQL_C_BINARY                                               |
| SQL_LONGVARCHAR                                        | SQL_C_CHAR                                                 |
| SQL_NUMERIC                                            | SQL_C_NUMERIC                                              |
| SQL_REAL                                               | SQL_C_FLOAT                                                |
| SQL_SMALLINT                                           | SQL_C_SSHORT                                               |
| SQL_TIMESTAMP                                          | SQL_C_TIMESTAMP                                            |
| SQL_VARCHAR                                            | SQL_C_CHARS                                                |

## Additional default C data types for HCL OneDB™

There is default C data type for each additional HCL OneDB™ ODBC Driver SQL data type.

For each additional HCL OneDB™ ODBC Driver SQL data type for HCL OneDB™, the following table shows the default C data type.

| <b>HCL OneDB™ ODBC driver SQL data type (fSqlType)</b> | <b>Default HCL OneDB™ ODBC driver C data type (fCType)</b> |
|--------------------------------------------------------|------------------------------------------------------------|
| SQL_IFMX_U                                             | SQL_C_BINARY                                               |
| DT_BLOB                                                |                                                            |
| SQL_IFMX_U                                             | SQL_C_BINARY                                               |
| DT_CLOB                                                |                                                            |

| HCL OneDB™<br>ODBC driver<br>SQL data type<br>(fSqlType) | Default HCL OneDB™ ODBC driver C data type (fCType)                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQL_INFX_UD<br>T_FIXED                                   | This HCL OneDB™ ODBC Driver SQL data type does not have a default HCL OneDB™ ODBC Driver C data type. Because this HCL OneDB™ ODBC driver SQL data type can contain binary data or character data, you must bind a variable for this HCL OneDB™ ODBC driver SQL data type before you fetch a corresponding value. The data type of the bound variable specifies the C data type for the value. |
| SQL_INFX_UD<br>T_VARYING                                 | This HCL OneDB™ ODBC Driver SQL data type does not have a default HCL OneDB™ ODBC Driver C data type. Because this HCL OneDB™ ODBC Driver SQL data type can contain binary data or character data, you must bind a variable for this HCL OneDB™ ODBC Driver SQL data type before you fetch a corresponding value. The data type of the bound variable specifies the C data type for the value. |

## SQL to C: Binary

The binary HCL OneDB™ ODBC Driver SQL data type is SQL\_LONGVARBINARY.

The following table shows the HCL OneDB™ ODBC Driver C data types to which binary SQL data can be converted.

| fCType       | Test                                          | rgbValue       | pcbValue       | SQLSTATE |
|--------------|-----------------------------------------------|----------------|----------------|----------|
| SQL_C_BINARY | Length of data $\leq$ <i>cbValueMax</i>       | Data           | Length of data | N/A      |
|              | Length of data > <i>cbValueMax</i>            | Truncated data | Length of data | 01004    |
| SQL_C_CHAR   | (Length of data) * 2 < <i>cbValueMax</i>      | Data           | Length of data | N/A      |
|              | (Length of data) * 2 $\geq$ <i>cbValueMax</i> | Truncated data | Length of data | 01004    |

When HCL OneDB™ ODBC Driver converts binary SQL data to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, HCL OneDB™ ODBC Driver converts binary 00000001 to 01 and binary 11111111 to FF.

HCL OneDB™ ODBC Driver converts individual bytes to pairs of hexadecimal digits and terminates the character string with a null byte. Because of this conversion, if *cbValueMax* is even and is less than the length of the converted data, the last byte of the *rgbValue* buffer is not used. (The converted data requires an even number of bytes, the next-to-last byte is a null byte, and the last byte cannot be used.)

## SQL to C: Boolean

The Boolean HCL OneDB™ ODBC Driver SQL data type is SQL\_BIT.

The following table shows the HCL OneDB™ ODBC Driver C data types to which Boolean SQL data can be converted. When HCL OneDB™ ODBC Driver converts Boolean SQL data to character C data, the possible values are 0 and 1.

| <b>fCType</b>       | <b>Test</b>                                                                                                                                                                 | <b>rgbValue</b> | <b>pcbValue</b>                                               | <b>SQLSTATE</b> |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|---------------------------------------------------------------|-----------------|
| <b>SQL_C_BINARY</b> | <i>cbValueMax</i> 1                                                                                                                                                         | Data            | 1                                                             | N/A             |
|                     | <i>cbValueMax</i> < 1                                                                                                                                                       | Untouched       | Untouched                                                     | 22003           |
| <b>SQL_C_BIT</b>    | HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type. | Data            | 1<br><br>(This is the size of the corresponding C data type.) | N/A             |
| <b>SQL_C_CHAR</b>   | <i>cbValueMax</i> > 1                                                                                                                                                       | Data            | 1                                                             | N/A             |
|                     | <i>cbValueMax</i> 1                                                                                                                                                         | Untouched       | Untouched                                                     | 22003           |

### SQL to C: Character

The character HCL OneDB™ ODBC Driver SQL data types are SQL\_CHAR, SQL\_LONGVARCHAR, and SQL\_VARCHAR.

The following table shows the HCL OneDB™ ODBC Driver C data types to which character SQL data can be converted. When HCL OneDB™ ODBC Driver converts character SQL data to numeric, date, or time stamp C data, it ignores leading and trailing spaces.

| <b>fCType</b>       | <b>Test</b>                                              | <b>rgbValue</b> | <b>pcbValue</b>                                                    | <b>SQLSTATE</b> |
|---------------------|----------------------------------------------------------|-----------------|--------------------------------------------------------------------|-----------------|
| <b>SQL_C_BINARY</b> | Length of data < <i>cbValueMax</i> .                     | Data            | Length of data                                                     | N/A             |
|                     | Length of data > <i>cbValueMax</i> .                     | Truncated data  | Length of data                                                     | 01004           |
| <b>SQL_C_BIT</b>    | Data is 0 or 1.                                          | Data            | 1                                                                  | N/A             |
|                     | Data is greater than 0, less than 2, and not equal to 1. | Truncated data  | 1                                                                  | 01004           |
|                     | Data is less than 0 or greater than or equal to 2.       | Untouched       | Untouched                                                          | 22003           |
|                     | Data is not a <i>numeric-literal</i> .                   | Untouched       | Untouched<br><br>(The size of the corresponding C data type is 1.) | 22005           |
| <b>SQL_C_CHAR</b>   | Length of data < <i>cbValueMax</i> .                     | Data            | Length of data                                                     | N/A             |
|                     | Length of data <i>cbValueMax</i> .                       | Truncated data  | Length of data                                                     | 01004           |



| fCType                                                                                                                                                                                                      | Test                                                                                                                                                                                                                                                                 | rgbValue       | pcbValue                                                           | SQLSTATE |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|--------------------------------------------------------------------|----------|
| <b>SQL_C_DATE</b>                                                                                                                                                                                           | Data value is a valid date-value.                                                                                                                                                                                                                                    | Data           | 6                                                                  | N/A      |
|                                                                                                                                                                                                             | Data value is a valid <i>timestamp-value</i> ; time portion is zero.                                                                                                                                                                                                 | Data           | 6                                                                  | N/A      |
|                                                                                                                                                                                                             | Data value is a valid <i>timestamp-value</i> ; time portion is non-zero.<br><br>(HCL OneDB™ ODBC Driver ignores the date portion of <i>timestamp-value</i> .)                                                                                                        | Truncated data | 6                                                                  | 01004    |
|                                                                                                                                                                                                             | Data value is not a valid <i>date-value</i> or <i>timestamp-value</i> .<br><br>(For all these conversions, HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> . HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.) | Untouched      | Untouched<br><br>(The size of the corresponding C data type is 6.) | 22008    |
| <b>SQL_C_DOUBLE</b><br><b>SQL_C_FLOAT</b>                                                                                                                                                                   | Data is within the range of the data type to which the number is being converted.                                                                                                                                                                                    | Data           | Size of the C data type                                            | N/A      |
|                                                                                                                                                                                                             | Data is outside the range of the data type to which the number is being converted.                                                                                                                                                                                   | Untouched      | Untouched                                                          | 22003    |
|                                                                                                                                                                                                             | Data is not a <i>numeric-literal</i> .<br><br>(For all these conversions, HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> . HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.)                                  | Untouched      | Untouched                                                          | 22005    |
| <b>SQL_C_LONG</b><br><b>SQL_C_SHORT</b><br><b>SQL_C_SLONG</b><br><b>SQL_C_SSHORT</b><br><b>SQL_C_STINYINT</b><br><b>SQL_C_TINYINT</b><br><b>SQL_C_ULONG</b><br><b>SQL_C_USHORT</b><br><b>SQL_C_UTINYINT</b> | Data converted without truncation.                                                                                                                                                                                                                                   | Data           | Size of the C data type                                            | N/A      |
|                                                                                                                                                                                                             | Data converted with truncation of fractional digits.                                                                                                                                                                                                                 | Truncated data | Size of the C data type                                            | 01004    |
|                                                                                                                                                                                                             | Conversion of data would result in loss of whole (as opposed to fractional) digits.                                                                                                                                                                                  | Untouched      | Untouched                                                          | 22003    |
|                                                                                                                                                                                                             | Data is not a <i>numeric-literal</i> .<br><br>(For all these conversions, HCL OneDB™ ODBC Driver ignores the value of                                                                                                                                                | Untouched      | Untouched                                                          | 22005    |

| fCType                 | Test                                                                                                                                                                                                                                                                                       | rgbValue                                                                                                   | pcbValue                                                            | SQLSTATE |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|----------|
|                        | <i>cbValueMax</i> . HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.)                                                                                                                                                                              |                                                                                                            |                                                                     |          |
| <b>SQL_C_TIMESTAMP</b> | Data value is a valid time stamp-value; fractional seconds portion not truncated.                                                                                                                                                                                                          | Data                                                                                                       | 16                                                                  | N/A      |
|                        | Data value is a valid <i>timestamp-value</i> ; fractional seconds portion truncated.                                                                                                                                                                                                       | Truncated data                                                                                             | 16                                                                  | N/A      |
|                        | Data value is a valid <i>date-value</i> .                                                                                                                                                                                                                                                  | Data<br><br>(HCL OneDB™ ODBC Driver sets the time fields of the time stamp structure to zero.)             | 16                                                                  | N/A      |
|                        | Data value is a valid <i>time-value</i> .                                                                                                                                                                                                                                                  | Data<br><br>(HCL OneDB™ ODBC Driver sets the date fields of the time stamp structure to the current date.) | 16                                                                  | N/A      |
|                        | Data value is not a valid <i>date-value</i> , <i>time-value</i> , or <i>timestamp-value</i> .<br><br>(For all these conversions, HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> . HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.) | Untouched                                                                                                  | Untouched<br><br>(The size of the corresponding C data type is 16.) | 22008    |

## SQL to C: Date

The date HCL OneDB™ ODBC Driver SQL data type is SQL\_DATE.

The following table shows the HCL OneDB™ ODBC Driver C data types to which date SQL data can be converted. When HCL OneDB™ ODBC Driver converts date SQL data to character C data, the resulting string is in the `yyyy-mm-dd` format.

| fCType          | Test                                                                                                                                                                        | rgbValue                                                                                       | pcbValue                                                       | SQLSTATE |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|----------------------------------------------------------------|----------|
| SQL_C_BINARY    | Length of data < <i>cbValueMax</i>                                                                                                                                          | Data                                                                                           | Length of data                                                 | N/A      |
|                 | Length of data > <i>cbValueMax</i>                                                                                                                                          | Untouched                                                                                      | Untouched                                                      | 22003    |
| SQL_C_CHAR      | <i>cbValueMax</i> 11                                                                                                                                                        | Data                                                                                           | 10                                                             | N/A      |
|                 | <i>cbValueMax</i> < 11                                                                                                                                                      | Untouched                                                                                      | Untouched                                                      | 22003    |
| SQL_C_DATE      | HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type. | Data                                                                                           | 6<br><br>(This is the size of the corresponding C data type.)  | N/A      |
| SQL_C_TIMESTAMP | HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type. | Data<br><br>(HCL OneDB™ ODBC Driver sets the time fields of the time stamp structure to zero.) | 16<br><br>(This is the size of the corresponding C data type.) | N/A      |

## SQL to C: Numeric

The numeric HCL OneDB™ ODBC Driver SQL data types are SQL\_DECIMAL, SQL\_DOUBLE, SQL\_INTEGER, SQL\_REAL, and SQL\_SMALLINT

The following table shows the HCL OneDB™ ODBC Driver C data types to which numeric SQL data can be converted.

| fCType       | Test                                                     | rgbValue       | pcbValue       | SQLSTATE |
|--------------|----------------------------------------------------------|----------------|----------------|----------|
| SQL_C_BINARY | Length of data < <i>cbValueMax</i> .                     | Data           | Length of data | N/A      |
|              | Length of data > <i>cbValueMax</i> .                     | Untouched      | Untouched      | 22003    |
| SQL_C_BIT    | Data is 0 or 1.                                          | Data           | 1              | N/A      |
|              | Data is greater than 0, less than 2, and not equal to 1. | Truncated data | 1              | 01004    |
|              | Data is less than 0 or greater than or equal to 2.       | Untouched      | Untouched      | 22003    |
|              | Data is not a <i>numeric-literal</i> .                   | Untouched      | Untouched      | 22005    |

| <b>fCType</b>                                                                                                       | <b>Test</b>                                                                                                                                                                                                                                                              | <b>rgbValue</b> | <b>pcbValue</b>                                   | <b>SQLSTATE</b> |
|---------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|---------------------------------------------------|-----------------|
|                                                                                                                     |                                                                                                                                                                                                                                                                          |                 | (The size of the corresponding C data type is 1.) |                 |
| <b>SQL_C_CHAR</b>                                                                                                   | Display size < <i>cbValueMax</i>                                                                                                                                                                                                                                         | Data            | Length of data                                    | N/A             |
|                                                                                                                     | Number of whole (as opposed to fractional) digits < <i>cbValueMax</i> .                                                                                                                                                                                                  | Truncated data  | Length of data                                    | 01004           |
|                                                                                                                     | Number of whole (as opposed to fractional) digits <i>cbValueMax</i> .                                                                                                                                                                                                    | Untouched       | Untouched                                         | 22003           |
| <b>SQL_C_DOUBLESQL_C_FLOAT</b>                                                                                      | Data is within the range of the data type to which the number is being converted.                                                                                                                                                                                        | Data            | Size of the C data type                           | N/A             |
|                                                                                                                     | Data is outside the range of the data type to which the number is being converted.<br><br>(HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.)  | Untouched       | Untouched                                         | 22003           |
| <b>SQL_C_LONGSQL_C_SHORTSQL_C_SLONGSQL_C_SSHORTSQL_C_STINYINTSQL_C_TINYINTSQL_C_ULONGSQL_C_USHORTSQL_C_UTINYINT</b> | Data converted without truncation.                                                                                                                                                                                                                                       | Data            | Size of the C data type                           | N/A             |
|                                                                                                                     | Data converted with truncation of fractional digits.                                                                                                                                                                                                                     | Truncated data  | Size of the C data type                           | 01004           |
|                                                                                                                     | Conversion of data would result in loss of whole (as opposed to fractional) digits.<br><br>(HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.) | Untouched       | Untouched                                         | 22003           |

## SQL to C: Time stamp

The time stamp HCL OneDB™ ODBC Driver SQL data type is SQL\_TIMESTAMP.

The following table shows the HCL OneDB™ ODBC Driver C data types to which time stamp SQL data can be converted.

| <b>fCType</b>          | <b>Test</b>                                                                                                                                                                                                                                 | <b>rgbValue</b>                                                                                            | <b>pcbValue</b>                                              | <b>SQLSTATE</b> |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|-----------------|
| <b>SQL_C_BINARY</b>    | Length of data < <i>cbValueMax</i> .                                                                                                                                                                                                        | Data                                                                                                       | Length of data                                               | N/A             |
|                        | Length of data > <i>cbValueMax</i> .                                                                                                                                                                                                        | Untouched                                                                                                  | Untouched                                                    | 22003           |
| <b>SQL_C_CHAR</b>      | <i>cbValueMax</i> > Display size.                                                                                                                                                                                                           | Data                                                                                                       | Length of data                                               | N/A             |
|                        | 20 < <i>cbValueMax</i> < Display size.                                                                                                                                                                                                      | Truncated data<br><br>(HCL OneDB™ ODBC Driver truncates the fractional seconds portion of the time stamp.) | Length of data                                               | 01004           |
|                        | <i>cbValueMax</i> < 20.                                                                                                                                                                                                                     | Untouched                                                                                                  | Untouched                                                    | 22003           |
| <b>SQL_C_DATE</b>      | Time portion of time stamp is zero.                                                                                                                                                                                                         | Data                                                                                                       | 6                                                            | N/A             |
|                        | Time portion of time stamp is nonzero.<br><br>(HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.)                 | Truncated data<br><br>(HCL OneDB™ ODBC Driver truncates the time portion of the time stamp.)               | 6<br><br>(The size of the corresponding C data type is 6.)   | 01004           |
| <b>SQL_C_TIMESTAMP</b> | Fractional seconds portion of time stamp is not truncated.                                                                                                                                                                                  | Data                                                                                                       | 16                                                           | N/A             |
|                        | Fractional seconds portion of time stamp is truncated.<br><br>(HCL OneDB™ ODBC Driver ignores the value of <i>cbValueMax</i> for this conversion. HCL OneDB™ ODBC Driver uses the size of <i>rgbValue</i> for the size of the C data type.) | Truncated data<br><br>(HCL OneDB™ ODBC Driver truncates the fractional seconds portion of the time stamp.) | 16<br><br>(The size of the corresponding C data type is 16.) | 01004           |

When HCL OneDB™ ODBC Driver converts time stamp SQL data to character C data, the resulting string is in the `yyyy-mm-dd hh:mm:ss[.f...]` format, where up to nine digits can be used for fractional seconds. Except for the decimal point and fractional seconds, the entire format must be used, regardless of the precision of the time stamp SQL data type.

## SQL-to-C data conversion examples

The examples show how HCL OneDB™ ODBC Driver converts SQL data to C data.

The following table illustrates how HCL OneDB™ ODBC Driver converts SQL data to C data. “\0” represents a null-termination byte (“\0” represents a wide null termination character when the C data type is `SQL_C_WCHAR`). HCL OneDB™ ODBC Driver always null-terminates `SQL_C_CHAR` and `SQL_C_WCHAR` data. For the combination of `SQL_DATE` and `SQL_C_TIMESTAMP`, HCL OneDB™ ODBC Driver stores the numbers that are in the `rgbValue` column in the fields of the `TIMESTAMP_STRUCT` structure.

| SQL data type | SQL data value | C data type    | cbValueMax | rgbValue     | SQLSTATE |
|---------------|----------------|----------------|------------|--------------|----------|
| SQL_CHAR      | tigers         | SQL_C_CHAR     | 7          | tigers\0     | N/A      |
| SQL_CHAR      | tigers         | SQL_C_CHAR     | 6          | tiger\0      | 01004    |
| SQL_CHAR      | tigers         | SQL_C_WCHAR    | 14         | tigers\0     | N/A      |
| SQL_CHAR      | tigers         | SQL_C_WCHAR    | 12         | tiger\0      | 01004    |
| SQL_DECIMAL   | 1234.56        | SQL_C_CHAR     | 8          | 1234.56\0    | N/A      |
| SQL_DECIMAL   | 1234.56        | SQL_C_CHAR     | 5          | 1234\0       | 01004    |
| SQL_DECIMAL   | 1234.56        | SQL_C_CHAR     | 4          | —            | 22003    |
| SQL_DECIMAL   | 1234.56        | SQL_C_WCHAR    | 16         | 1234.56\0    | N/A      |
| SQL_DECIMAL   | 1234.56        | SQL_C_WCHAR    | 10         | 1234\0       | 01004    |
| SQL_DECIMAL   | 1234.56        | SQL_C_WCHAR    | 8          | —            | 220023   |
| SQL_DECIMAL   | 1234.56        | SQL_C_FLOAT    | Ignored    | 1234.56      | N/A      |
| SQL_DECIMAL   | 1234.56        | SQL_C_SSHORT   | Ignored    | 1234         | 01004    |
| SQL_DECIMAL   | 1234.56        | SQL_C_STINYINT | Ignored    | —            | 22003    |
| SQL_DOUBLE    | 1.2345678      | SQL_C_DOUBLE   | Ignored    | 1.234567     | N/A      |
| SQL_DOUBLE    | 1.2345678      | SQL_C_FLOAT    | Ignored    | 1.234567     | N/A      |
| SQL_DOUBLE    | 1.2345678      | SQL_C_STINYINT | Ignored    | 1            | N/A      |
| SQL_DATE      | 1992-12-31     | SQL_C_CHAR     | 11         | 1992-12-31\0 | N/A      |
| SQL_DATE      | 1992-12-31     | SQL_C_CHAR     | 10         | —            | 22003    |

| SQL data type | SQL data value            | C data type     | cbValueMax | rgbValue                    | SQLSTATE |
|---------------|---------------------------|-----------------|------------|-----------------------------|----------|
| SQL_DATE      | 1992-12-31                | SQL_C_WCHAR     | 22         | 1992-12-31\0                | N/A      |
| SQL_DATE      | 1992-12-31                | SQL_C_WCHAR     | 20         | —                           | 22003    |
| SQL_DATE      | 1992-12-31                | SQL_C_TIMESTAMP | Ignored    | 1992,12,31,<br>0,0,0,0      | N/A      |
| SQL_TIMESTAMP | 1992-12-31<br>23:45:55.12 | SQL_C_CHAR      | 23         | 1992-12-31<br>23:45:55.12\0 | N/A      |
| SQL_TIMESTAMP | 1992-12-31<br>23:45:55.12 | SQL_C_CHAR      | 22         | 1992-12-31<br>23:45:55.1\0  | 01004    |
| SQL_TIMESTAMP | 1992-12-31<br>23:45:55.12 | SQL_C_CHAR      | 18         | —                           | 22003    |
| SQL_TIMESTAMP | 1992-12-31<br>23:45:55.12 | SQL_C_WCHAR     | 46         | 1992-12-31<br>23:45:55.12\0 | N/A      |
| SQL_TIMESTAMP | 1992-12-31<br>23:45:55.12 | SQL_C_WCHAR     | 44         | 1992-12-31<br>23:45:55.1\0  | 01004    |
| SQL_TIMESTAMP | 1992-12-31<br>23:45:55.12 | SQL_C_WCHAR     | 36         | —                           | 22003    |



**Important:** The size of a wide character (`wchar_t`) is platform dependent. The previous examples are applicable to Windows™ where the size of wide characters is 2 bytes. On most UNIX™ platforms, wide characters are 4 bytes. On IBM AIX® versions lower than AIX5L, it is 2 bytes.

## Convert data from C to SQL

When you call `SQLExecute` or `SQLExecDirect`, HCL OneDB™ ODBC Driver retrieves the data for parameters that are bound with `SQLBindParameter` from storage locations in the application.

For data-at-execution parameters, call `SQLPutData` to send the parameter data. If necessary, HCL OneDB™ ODBC Driver converts the data from the data type that the `ValueType` argument specifies in `SQLBindParameter` to the data type that the `fSqlType` argument specifies in the `SQLBindParameter`. Finally, HCL OneDB™ ODBC Driver sends the data to the data source.

If the *rgbValue* and *pcbValue* arguments specified in `SQLBindParameter` are both null pointers, then that function returns `SQLSTATE S1009` (Invalid argument value). To specify a null SQL data value, set the value that the *pcbValue* argument of `SQLBindParameter` points to or the value of the *cbValue* argument to `SQL_NULL_DATA`. To specify that the value in *rgbValue* is a null-terminated string, set these values to `SQL_NTS`.

The following terms are used in the tables:

**Length of data**

The number of bytes of SQL data that are available to send to the data source, regardless of whether the data is truncated before it goes to the data source. For string data, this does not include the null-termination byte.

**Column length and display size**

Defined for each SQL data type in [Precision, scale, length, and display size on page 65](#).

**Number of digits**

The number of characters that represent a number, including the minus sign, decimal point, and exponent (if needed).

**Words in italics**

Represent elements of the HCL OneDB™ ODBC Driver SQL syntax.

## C to SQL: Binary

The binary HCL OneDB™ ODBC Driver C data type is **SQL\_C\_BINARY**.

The following table shows the HCL OneDB™ ODBC Driver SQL data types to which binary C data can be converted. In the Test column, the SQL data length is the number of bytes needed to store the data on the data source. This length might be different from the column length, as defined in [Precision, scale, length, and display size on page 65](#).

| <b>fSqlType</b> | <b>Test</b>                            | <b>SQLSTATE</b> |
|-----------------|----------------------------------------|-----------------|
| SQL_BIGINT      | Length of data = SQL data length.      | N/A             |
|                 | Length of data $\neq$ SQL data length. | 22003           |
| SQL_BIT         | Length of data = SQL data length.      | N/A             |
|                 | Length of data $\neq$ SQL data length. | 22003           |
| SQL_CHAR        | Length of data < Column length.        | N/A             |
| SQL_LONGVARCHAR | Length of data > Column length.        | 01004           |
| SQL_VARCHAR     |                                        |                 |
| SQL_DATE        | Length of data = SQL data length.      | N/A             |
| SQL_TIMESTAMP   | Length of data $\neq$ SQL data length. | 22003           |



| <b>fSqlType</b>   | <b>Test</b>                            | <b>SQLSTATE</b> |
|-------------------|----------------------------------------|-----------------|
| SQL_DECIMAL       | Length of data = SQL data length.      | N/A             |
| SQL_DOUBLE        | Length of data $\neq$ SQL data length. | 22003           |
| SQL_INTEGER       |                                        |                 |
| SQL_REAL          |                                        |                 |
| SQL_SMALLINT      |                                        |                 |
| SQL_LONGVARBINARY | Length of data < Column length.        | N/A             |
|                   | Length of data > Column length.        | 01004           |

## C to SQL: Bit

The bit HCL OneDB™ ODBC Driver C data type is **SQL\_C\_BIT**.

The following table shows the HCL OneDB™ ODBC Driver SQL data types to which bit C data can be converted.

| <b>fSqlType</b> | <b>Test</b> | <b>SQLSTATE</b> |
|-----------------|-------------|-----------------|
| SQL_BIGINT      | None        | N/A             |
| SQL_DECIMAL     |             |                 |
| SQL_DOUBLE      |             |                 |
| SQL_INTEGER     |             |                 |
| SQL_REAL        |             |                 |
| SQL_SMALLINT    |             |                 |
| SQL_BIT         | None        | N/A             |
| SQL_CHAR        | None        | N/A             |
| SQL_LONGVARCHAR |             |                 |
| SQL_VARCHAR     |             |                 |

HCL OneDB™ ODBC Driver ignores the value that the *pcbValue* argument of `SQLBindParameter` points to and the value of the *cbValue* argument of `SQLPutData` when it converts data from the Boolean C data type. HCL OneDB™ ODBC Driver uses the size of *rgbValue* for the size of the Boolean C data type.

## C to SQL: Character

The character HCL OneDB™ ODBC Driver C data type is **SQL\_C\_CHAR**.

The following table shows the HCL OneDB™ ODBC Driver SQL data types to which C character data can be converted.

| fSqlType        | Test                                                                                                                                                                   | SQLSTATE |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| SQL_BIGINT      | Data converted without truncation.                                                                                                                                     | N/A      |
|                 | Data converted with truncation of fractional digits.                                                                                                                   | 01004    |
|                 | Conversion of data would result in loss of whole (as opposed to fractional) digits.                                                                                    | 22003    |
|                 | Data value is not a <i>numeric-literal</i> .                                                                                                                           | 22005    |
| SQL_BIT         | Data is 0 or 1.                                                                                                                                                        | N/A      |
|                 | Data is greater than 0, less than 2, and not equal to 1.                                                                                                               | 01004    |
|                 | Data is less than 0 or greater than or equal to 2.                                                                                                                     | 22003    |
|                 | Data is not a <i>numeric-literal</i> .                                                                                                                                 | 22005    |
| SQL_CHAR        | Length of data < Column length.                                                                                                                                        | N/A      |
| SQL_LONGVARCHAR | Length of data > Column length.                                                                                                                                        | 01004    |
| SQL_VARCHAR     |                                                                                                                                                                        |          |
| SQL_DATE        | Data value is a valid HCL OneDB™ ODBC driver <i>date-literal</i> .                                                                                                     | N/A      |
|                 | Data value is a valid HCL OneDB™ ODBC driver <i>timestamp-literal</i> ; time portion is zero.                                                                          | N/A      |
|                 | Data value is a valid HCL OneDB™ ODBC driver <i>timestamp-literal</i> ; time portion is non-zero. HCL OneDB™ ODBC driver truncates the time portion of the time stamp. | 01004    |
|                 | Data value is not a valid HCL OneDB™ ODBC driver <i>date-literal</i> or HCL OneDB™ ODBC driver <i>timestamp-literal</i> .                                              | 22008    |
| SQL_DECIMAL     | Data converted without truncation.                                                                                                                                     | N/A      |
| SQL_INTEGER     | Data converted with truncation of fractional digits.                                                                                                                   | 01004    |
| SQL_SMALLINT    | Conversion of data would result in loss of whole (as opposed to fractional) digits.                                                                                    | 22003    |
|                 | Data value is not a <i>numeric-literal</i> .                                                                                                                           | 22005    |
| SQL_DOUBLE      | Data is within the range of the data type to which the number is being converted.                                                                                      | N/A      |
| SQL_REAL        |                                                                                                                                                                        |          |

| fSqlType          | Test                                                                                                                                                                     | SQLSTATE |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
|                   | Data is outside the range of the data type to which the number is being converted.                                                                                       | 22003    |
|                   | Data value is not a <i>numeric-literal</i> .                                                                                                                             | 22005    |
| SQL_LONGVARBINARY | (Length of data) / 2 < Column length.                                                                                                                                    | N/A      |
|                   | (Length of data) / 2 > Column length.                                                                                                                                    | 01004    |
|                   | Data value is not a hexadecimal value.                                                                                                                                   | 22005    |
| SQL_TIMESTAMP     | Data value is a valid HCL OneDB™ ODBC driver <i>timestamp-literal</i> ; fractional seconds portion not truncated.                                                        | N/A      |
|                   | Data value is a valid HCL OneDB™ ODBC driver <i>timestamp-literal</i> ; fractional seconds portion truncated.                                                            | 01004    |
|                   | Data value is a valid HCL OneDB™ ODBC driver <i>date-literal</i> . HCL OneDB™ ODBC driver sets the time portion of the time stamp to zero.                               | N/A      |
|                   | Data value is a valid HCL OneDB™ ODBC driver <i>time-literal</i> . HCL OneDB™ ODBC driver sets the date portion of the time stamp to the current date.                   | N/A      |
|                   | Data value is not a valid HCL OneDB™ ODBC driver <i>date-literal</i> , HCL OneDB™ ODBC driver <i>time-literal</i> , or HCL OneDB™ ODBC driver <i>timestamp-literal</i> . | 22008    |

When HCL OneDB™ ODBC Driver converts character C data to numeric, date, or time stamp SQL data, it ignores leading and trailing blanks. When HCL OneDB™ ODBC Driver converts character C data to binary SQL data, it converts each two bytes of character data to one byte of binary data. Each two bytes of character data represent a number in hexadecimal form. For example, HCL OneDB™ ODBC Driver converts "01" to binary 00000001 and "FF" to binary 11111111.

HCL OneDB™ ODBC Driver always converts pairs of hexadecimal digits to individual bytes and ignores the null-termination byte. Because of this conversion, if the length of the character string is odd, the last byte of the string (excluding the null termination byte, if any) is not converted.

## C to SQL: Date

The date HCL OneDB™ ODBC Driver C data type is **SQL\_C\_DATE**.

The following table shows the HCL OneDB™ ODBC Driver SQL data types to which date C data can be converted.

| fSqlType        | Test                                                                                                | SQLSTATE |
|-----------------|-----------------------------------------------------------------------------------------------------|----------|
| SQL_CHAR        | Column length 10.                                                                                   | N/A      |
| SQL_LONGVARCHAR | Column length < 10.                                                                                 | 22003    |
| SQL_VARCHAR     | Data value is not a valid date.                                                                     | 22008    |
| SQL_DATE        | Data value is a valid date.                                                                         | N/A      |
|                 | Data value is not a valid date.                                                                     | 22008    |
| SQL_TIMESTAMP   | Data value is a valid date. HCL OneDB™ ODBC driver sets the time portion of the time stamp to zero. | N/A      |
|                 | Data value is not a valid date.                                                                     | 22008    |

When HCL OneDB™ ODBC Driver converts date C data to character SQL data, the resulting character data is in the `yyyy-mm-dd` format.

HCL OneDB™ ODBC Driver ignores the value that the *pcbValue* argument of `SQLBindParameter` points to and the value of the *cbValue* argument of `SQLPutData` when it converts data from the date C data type. HCL OneDB™ ODBC Driver uses the size of *rgbValue* for the size of the date C data type.

## C to SQL: Numeric

There are a total of ten HCL OneDB™ ODBC Driver C data types.

The numeric HCL OneDB™ ODBC Driver C data types are:

- **SQL\_C\_DOUBLE**
- **SQL\_C\_FLOAT**
- **SQL\_C\_LONG**
- **SQL\_C\_SHORT**
- **SQL\_C\_SLONG**
- **SQL\_C\_STINYINT**
- **SQL\_C\_TINYINT**
- **SQL\_C\_ULONG**
- **SQL\_C\_USHORT**
- **SQL\_C\_UTINYINT**

The following table shows the HCL OneDB™ ODBC Driver SQL data types to which numeric C data can be converted.

| fSqlType        | Test                                                                                | SQLSTATE |
|-----------------|-------------------------------------------------------------------------------------|----------|
| SQL_BIGINT      | Data converted without truncation.                                                  | N/A      |
|                 | Data converted with truncation of fractional digits.                                | 01004    |
|                 | Conversion of data would result in loss of whole (as opposed to fractional) digits. | 22003    |
| SQL_BIT         | Data is 0 or 1.                                                                     | N/A      |
|                 | Data is greater than 0, less than 2, and not equal to 1.                            | 01004    |
|                 | Data is less than 0 or greater than or equal to 2.                                  | 22003    |
| SQL_CHAR        | Number of digits < Column length.                                                   | N/A      |
| SQL_LONGVARCHAR | Number of whole (as opposed to fractional) digits < Column length.                  | 01004    |
| SQL_VARCHAR     | Number of whole (as opposed to fractional) digits > Column length.                  | 22003    |
| SQL_DECIMAL     | Data converted without truncation                                                   | N/A      |
| SQL_INTEGER     | Data converted with truncation of fractional digits.                                | 01004    |
| SQL_SMALLINT    | Conversion of data would result in loss of whole (as opposed to fractional) digits. | 22003    |
| SQL_DOUBLE      | Data is within the range of the data type to which the number is being converted.   | N/A      |
| SQL_REAL        | Data is outside the range of the data type to which the number is being converted.  | 22003    |

HCL OneDB™ ODBC Driver ignores the value that the *pcbValue* argument of SQLBindParameter points to and the value of the *cbValue* argument of SQLPutData when it converts data from the numeric C data types. HCL OneDB™ ODBC Driver uses the size of *rgbValue* for the size of the numeric C data type.

## C to SQL: Time stamp

The time stamp HCL OneDB™ ODBC Driver C data type is **SQL\_C\_TIMESTAMP**.

The following table shows the HCL OneDB™ ODBC Driver SQL data types to which time stamp C data can be converted.

| fSqlType | Test                        | SQLSTATE |
|----------|-----------------------------|----------|
| SQL_CHAR | Column length Display size. | N/A      |

| fSqlType                           | Test                                                                                                                                        | SQLSTATE |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|----------|
| SQL_LONGVARCHAR<br><br>SQL_VARCHAR | 19 < Column length < Display size.                                                                                                          | 01004    |
|                                    | HCL OneDB™ ODBC Driver truncates the fractional seconds of the time stamp.                                                                  |          |
|                                    | Column length < 19.                                                                                                                         | 22003    |
| SQL_DATE                           | Data value is not a valid date.                                                                                                             | 22008    |
|                                    | Time fields are zero.                                                                                                                       | N/A      |
|                                    | Time fields are non-zero.<br><br>HCL OneDB™ ODBC Driver truncates the time fields of the time stamp structure.                              | 01004    |
| SQL_TIMESTAMP                      | Data value does not contain a valid date.                                                                                                   | 22008    |
|                                    | Fractional seconds fields are not truncated.                                                                                                | N/A      |
|                                    | Fractional seconds fields are truncated.<br><br>HCL OneDB™ ODBC Driver truncates the fractional seconds fields of the time stamp structure. | 01004    |
| SQL_TIMESTAMP                      | Data value is not a valid time stamp.                                                                                                       | 22008    |

When HCL OneDB™ ODBC Driver converts time stamp C data to character SQL data, the resulting character data is in the `yyyy-mm-dd hh:mm:ss[.f...]` format.

HCL OneDB™ ODBC Driver ignores the value that the *pcbValue* argument of `SQLBindParameter` points to and the value of the *cbValue* argument of `SQLPutData` when it converts data from the time stamp C data type. HCL OneDB™ ODBC Driver uses the size of *rgbValue* for the size of the time stamp C data type.

## C-to-SQL data conversion examples

The examples show how HCL OneDB™ ODBC Driver converts C data to SQL data.

The following table illustrates how HCL OneDB™ ODBC Driver converts C data to SQL data. “\0” represents a null-termination byte. The null-termination byte is required only if the length of the data is `SQL_NTS`. For **SQL\_C\_DATE**, the numbers that are in the C Data Value column are the numbers that are stored in the fields of the `DATE_STRUCT` structure. For **SQL\_C\_TIMESTAMP**, the numbers that are in the C Data Value column are the numbers that are stored in the fields of the `TIMESTAMP_STRUCT` structure.

| C data type     | C data value                          | SQL data type | Column length                                                                                                   | SQL data value                | SQLSTATE |
|-----------------|---------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------|-------------------------------|----------|
| SQL_C_CHAR      | tigers\0                              | SQL_CHAR      | 6                                                                                                               | tigers                        | N/A      |
| SQL_C_CHAR      | tigers\0                              | SQL_CHAR      | 5                                                                                                               | tiger                         | 01004    |
| SQL_C_CHAR      | 1234.56\0                             | SQL_DECIMAL   | 8<br><br>(In addition to bytes for numbers, one byte is required for a sign and another for the decimal point.) | 1234.56                       | N/A      |
| SQL_C_CHAR      | 1234.56\0                             | SQL_DECIMAL   | 7<br><br>(In addition to bytes for numbers, one byte is required for a sign and another for the decimal point.) | 1234.5                        | 01004    |
| SQL_C_CHAR      | 1234.56\0                             | SQL_DECIMAL   | 4                                                                                                               | —                             | 22003    |
| SQL_C_FLOAT     | 1234.56                               | SQL_FLOAT     | not applicable                                                                                                  | 1234.56                       | N/A      |
| SQL_C_FLOAT     | 1234.56                               | SQL_INTEGER   | not applicable                                                                                                  | 1234                          | 01004    |
| SQL_C_FLOAT     | 1234.56                               | SQL_TINYINT   | not applicable                                                                                                  | —                             | 22003    |
| SQL_C_DATE      | 1992,12,31                            | SQL_CHAR      | 10                                                                                                              | 1992-12-31                    | N/A      |
| SQL_C_DATE      | 1992,12,31                            | SQL_CHAR      | 9                                                                                                               | —                             | 22003    |
| SQL_C_DATE      | 1992,12,31                            | SQL_TIMESTAMP | not applicable                                                                                                  | 1992-12-31<br><br>00:00:00.0  | N/A      |
| SQL_C_TIMESTAMP | 1992,12,31,<br>23,45,55,<br>120000000 | SQL_CHAR      | 22                                                                                                              | 1992-12-31<br><br>23:45:55.12 | N/A      |
| SQL_C_TIMESTAMP | 1992,12,31,<br>23,45,55,<br>120000000 | SQL_CHAR      | 21                                                                                                              | 1992-12-31<br><br>23:45:55.1  | 01004    |

| C data type     | C data value                          | SQL data type | Column length | SQL data value | SQLSTATE |
|-----------------|---------------------------------------|---------------|---------------|----------------|----------|
| SQL_C_TIMESTAMP | 1992,12,31,<br>23,45,55,<br>120000000 | SQL_CHAR      | 18            | —              | 22003    |

## Smart large objects

These topics describe how to store, create, and access a smart large object; how to transfer smart-large-object data; how to retrieve the status of a smart large object; and how to read or write a smart large object to or from a file.

The information in these topics apply only if your database server is .

A smart large object is a recoverable large object that is stored in an sbspace on disk. You can access a smart large object with read, write, and seek operations similar to an operating-system file. The two data types for smart large objects are *character large object* (CLOB) and *binary large object* (BLOB). A CLOB consists of text data and a BLOB consists of binary data in an undifferentiated byte stream.

For more information about smart-large-object data types, see the *HCL OneDB™ Guide to SQL: Reference*.

## Data structures for smart large objects

Because a smart large object can be huge, has two alternatives to store the content of a smart large object.

Therefore, instead of storing the content of a smart large object in a database table, does the following:

- Stores the content of the smart large object in an sbspace
- Stores a pointer to the smart large object in the database table

Because a smart large object can be huge, the HCL OneDB™ ODBC Driver application cannot receive a smart large object in a variable. Instead, the application sends or receives information about the smart large object in a data structure. The following table describes the data structures that HCL OneDB™ ODBC Driver uses for smart large objects.

| Data structure | Name                                 | Description                                                                                                                                                                                                                                                                                                                                                        |
|----------------|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lofd           | Smart-large-object file descriptor   | Provides access to a smart large object. Uses a file descriptor to access smart-large-object data as if it were in an operating-system file.                                                                                                                                                                                                                       |
| loptr          | Smart-large-object pointer structure | Provides security information and a pointer to a smart large object. This structure is the data that the database server stores in a database table for a smart large object. Therefore, SQL statements such as INSERT and SELECT accept a smart-large-object pointer structure as a value for a column or a parameter that has a data type of smart large object. |



| Data structure | Name                                       | Description                                                                                                                                                                                                                                                                                                                                                                    |
|----------------|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>lospec</b>  | Smart-large-object specification structure | Specifies the storage characteristics for a smart large object.                                                                                                                                                                                                                                                                                                                |
| <b>lostat</b>  | Smart-large-object status structure        | Stores status information for a smart large object. Normally you can fetch a user-defined data type (UDT) in either binary or character representation. However, it is not possible to convert a smart-large-object status structure to character representation. Therefore, you need to use <b>SQL_C_BINARY</b> as the HCL OneDB™ ODBC Driver C data type for <b>lostat</b> . |



**Restriction:** These data structures are opaque to HCL OneDB™ ODBC Driver applications and their internal structures might change. Therefore, do not access the internal structures directly. Use the smart-large-object client functions to manipulate the data structures.

The application is responsible for allocating space for these smart-large-object data structures.

## Working with a smart-large-object data structure

You can use this procedure to work with a smart-large-object data structure. An example is included.

### About this task

To work with a smart-large-object data structure:

1. Determine the size of the smart-large-object structure.
2. Use either a fixed size array or a dynamically allocated buffer that is at least the size of the data structure.
3. Free the array or buffer space when you are done with it.

### Results

The following code example illustrates these steps:

```
rc = SQLGetInfo(hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
 sizeof(lospec_size), NULL);
lospec_buffer = malloc(lospec_size);
;
free(lospec_buffer);
```

## Storage of smart large objects

The smart-large-object specification structure stores the disk-storage information and create-time flags for a smart large object.

### Disk-storage information

Disk-storage information helps HCL OneDB™ determine how to store the smart large object most efficiently on disk.

The following table describes the types of disk-storage information and the corresponding client functions. For most applications, it is recommended that you use the values for the disk-storage information that the database server determines.

| Disk-storage information | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Client functions                                       |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| Estimated size           | An estimate of the final size, in bytes, of the smart large object. The database server uses this value to determine the extents in which to store the smart large object. This value provides optimization information. If the value is grossly incorrect, it does not cause incorrect behavior. However, it does mean that the database server might not necessarily choose optimal extent sizes for the smart large object.                                                                                    | ifx_lo_specget_estbytes()<br>ifx_lo_specset_estbytes() |
| Maximum size             | The maximum size, in bytes, for the smart large object. The database server does not allow the smart large object to grow beyond this size.                                                                                                                                                                                                                                                                                                                                                                       | ifx_lo_specget_maxbytes()<br>ifx_lo_specset_maxbytes() |
| Allocation extent size   | The allocation extent size is specified in kilobytes. Optimally, the allocation extent is the single extent in a chunk that holds all the data for the smart large object.<br><br>The database server performs storage allocations for smart large objects in increments of the allocation extent size. It tries to allocate an allocation extent as a single extent in a chunk. However, if no single extent is large enough, the database server must use multiple extents as necessary to satisfy the request. | ifx_lo_specget_extsz()<br>ifx_lo_specset_extsz()       |
| Name of the sbspace      | The name of the sbspace that contains the smart large object. On this database server, an sbspace name can be up to 128 characters long and must be null terminated.                                                                                                                                                                                                                                                                                                                                              | ifx_lo_specget_sbspace()<br>ifx_lo_specset_sbspace()   |

## Create-time flags

Create-time flags tell HCL OneDB™ what options to assign to the smart large object.

The following table describes the create-time flags.

| Type of indicator | Create-time flag          | Description                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Logging           | LO_LOG                    | Tells the database server to log changes to the smart large object in the system log file.<br><br>Consider carefully whether to use the LO_LOG flag value. The database server incurs considerable overhead to log smart large objects. You must also make sure that the system log file is large enough to hold the value of the smart large object. For more information, see your <i>HCL OneDB™ Administrator's Guide</i> . |
|                   | LO_NOLOG                  | Tells the database server to turn off logging for all operations that involve the associated smart large object.                                                                                                                                                                                                                                                                                                               |
| Last access-time  | LO_KEEP_LASTACCESS_TIME   | Tells the database server to save the last access time for the smart large object. This access time is the time of the last read or write operation.<br><br>Consider carefully whether to use the LO_KEEP_LASTACCESS_TIME flag value. The database server incurs considerable overhead to maintain last access times for smart large objects.                                                                                  |
|                   | LO_NOKEEP_LASTACCESS_TIME | Tells the database server not to maintain the last access time for the smart large object.                                                                                                                                                                                                                                                                                                                                     |

The `ifx_lo_specset_flags()` function sets the create-time flags to a new value. The `ifx_lo_specget_flags()` function retrieves the current value of the create-time flag.

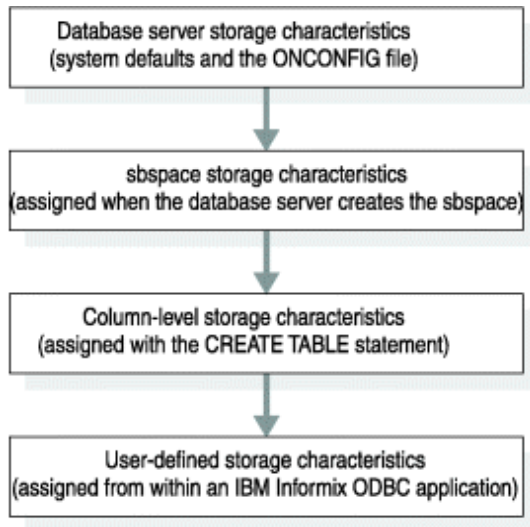
Logging indicators and the last access-time indicators are stored in the smart-large-object specification structure as a single flag value. To set a flag from each group, use the C-language OR operator to mask the two flag values together. However, masking mutually exclusive flags causes an error. If you do not specify a value for one of the flag groups, the database server uses the inheritance hierarchy to determine this information.

## Inheritance hierarchy

HCL OneDB™ uses an inheritance hierarchy to obtain storage characteristics.

The following figure shows the inheritance hierarchy for smart-large-object storage characteristics.

Figure 5. Inheritance hierarchy for storage characteristics



## Using system-specified storage characteristics

HCL OneDB™ uses one set of storage characteristics as the system-specified storage characteristics.

### About this task

HCL OneDB™ uses one of the following sets of storage characteristics:

- If the sbspace in which the smart large object is stored specifies a value for a particular storage characteristic, the database server uses the sbspace value as the system-specified storage characteristic.

The database administrator can use the `onspaces` utility to define storage characteristics for an sbspace.

- If the sbspace in which the smart large object is stored does not specify a value for a particular storage characteristic, the database server uses the system default as the system-specified storage characteristic.

The database server defines the system defaults for storage characteristics internally. However, you can specify a default sbspace name with the `SBSPACENAME` configuration parameter in the `onconfig` file. Also, an application call to `ifx_lo_col_info()` or `ifx_lo_specset_sbspace()` can supply the target sbspace in the smart-large-object specification structure.

**!** **Important:** An error occurs if the `SBSPACENAME` configuration parameter is not specified and the smart-large-object specification structure does not contain the name of the target sbspace.

It is recommended that you use the system-specified storage characteristics for the disk-storage information. For more information about sbspaces and the description of the `onspaces` utility, see your *HCL OneDB™ Administrator's Guide*.

To use system-specified storage characteristics for a new smart large object:

1. Call `ifx_lo_def_create_spec()` to allocate a smart-large-object specification structure and to initialize the structure to null values.
2. Call `ifx_lo_create()` to create an instance of the smart large object.

## Using column-level storage characteristics

The CREATE TABLE statement assigns storage characteristics to a database column.

### About this task

The PUT clause of the CREATE TABLE statement specifies storage characteristics for a smart-large-object column. The **syscolattns** system catalog table stores the column-level storage characteristics.

To use column-level storage characteristics for a new smart-large-object instance:

1. Call `ifx_lo_def_create_spec()` to allocate a smart-large-object specification structure and initialize this structure to null values.
2. Call `ifx_lo_col_info()` to retrieve the column-level storage characteristics and store them in the specified smart-large-object specification structure.
3. Call `ifx_lo_create()` to create an instance of the smart large object.

## User-defined storage characteristics

To specify user-defined storage characteristics, call an `ifx_lo_specset_*` function.

You can define a unique set of storage characteristics for a new smart large object, as follows:

- For a smart large object that will be stored in a column, you can override some storage characteristics for the column when you create an instance of a smart large object.

If you do not override some or all of these characteristics, the smart large object uses the column-level storage characteristics.

- You can specify a wider set of characteristics for a smart large object because a smart large object is not constrained by table column properties.

If you do not override some or all of these characteristics, the smart large object inherits the system-specified storage characteristics.

## Example of creating a smart large object

The code example, `locreate.c`, shows how to create a smart large object.

You can find the `locreate.c` file in the `%ONEDB_HOME%/demo/clidemo` directory on UNIX™ platforms and in the `%ONEDB_HOME%\demo\odbcdemo` directory in Windows™ environments. You can also find instructions on how to build the **odbc\_demo** database in the same location.

```
/*
** locreate.c
```

```

**
** To create a smart large object
**
** ODBC Functions:
** SQLAllocHandle
** SQLBindParameter
** SQLConnect
** SQLFreeStmt
** SQLGetInfo
** SQLDisconnect
** SQLExecDirect
**
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN 12
#define ERRMSG_LEN 200

UCHAR defDsn[] = "odbc_demo";

int checkError (SQLRETURN rc,
 SQLSMALLINT handleType,
 SQLHANDLE handle,
 char *errmsg)
{
 SQLRETURN retcode = SQL_SUCCESS;

 SQLSMALLINT errNum = 1;
 SQLCHAR sqlState[6];
 SQLINTEGER nativeError;
 SQLCHAR errMsg[ERRMSG_LEN];
 SQLSMALLINT textLengthPtr;

 if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
 {
 while (retcode != SQL_NO_DATA)
 {
 retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
 &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

 if (retcode == SQL_INVALID_HANDLE)
 {
 fprintf (stderr, "checkError function was called with an
 invalid handle!!\n");
 return 1;
 }
 }
 }
}

```

```

 if ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO))
 fprintf (stderr, "ERROR: %d: %s : %s \n", nativeError,
 sqlState, errMsg);

 errNum++;
 }

 fprintf (stderr, "%s\n", errMsg);
 return 1; /* all errors on this handle have been reported */
}
else
 return 0; /* no errors to report */
}

int main (long argc,
 char *argv[])
{
 /* Declare variables
 */

 /* Handles */
 SQLHDBC hdbc;
 SQLHENV henv;
 SQLHSTMT hstmt;

 /* Smart large object file descriptor */
 long lofd;
 long lofd_valsize = 0;

 /* Smart large object pointer structure */
 char* loptr_buffer;
 short loptr_size;
 long loptr_valsize = 0;

 /* Smart large object specification structure */
 char* lospec_buffer;
 short lospec_size;
 long lospec_valsize = 0;

 /* Write buffer */
 char* write_buffer;
 short write_size;
 long write_valsize = 0;

 /* Miscellaneous variables */
 UCHAR dsn[20]; /*name of the DSN used for connecting to the
 database*/

 SQLRETURN rc = 0;
 int in;

 FILE* hfile;
 char* lo_file_name = "advert.txt";

 char colname[BUFFER_LEN] = "item.advert";
 long colname_size = SQL_NTS;

```

```

long mode = LO_RDWR;
long cbMode = 0;

char* insertStmt = "INSERT INTO item VALUES (1005, 'Helmet', 235,
 'Each', ?, '39.95')";

/* STEP 1. Get data source name from command line (or use default).
** Allocate environment handle and set ODBC version.
** Allocate connection handle.
** Establish the database connection.
** Allocate the statement handle.
*/

/* If (dsn is not explicitly passed in as arg) */
if (argc != 2)
{
 /* Use default dsn - odbc_demo */
 fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
 strcpy ((char *)dsn, (char *)defDsn);
}
else
{
 /* Use specified dsn */
 strcpy ((char *)dsn, (char *)argv[1]);
 fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
 fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
 return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
 (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
 SQLSetEnvAttr failed\nExiting!!"))
 return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
 Handle Allocation failed\nExiting!!"))
 return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
 failed\n"))
 return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);

```



```

if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
 Handle Allocation failed\nExiting!!"))
 return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2. Get the size of the smart large object specification
** structure.
** Allocate a buffer to hold the structure.
** Create a default smart large object specification structure.
** Reset the statement parameters.
*/

/* Get the size of a smart large object specification structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
 sizeof(lospec_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 2 -- SQLGetInfo
 failed\n"))
 goto Exit;

/* Allocate a buffer to hold the smart large object specification
 structure*/
lospec_buffer = malloc (lospec_size);

/* Create a default smart large object specification structure */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
 lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter failed\n"))
 goto Exit;
rc = SQLExecDirect (hstmt, "{call ifx_lo_def_create_spec(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 2 done...default smart large object specification
 structure created\n");

/* STEP 3. Initialise the smart large object specification structure
** with values for the database column where the smart large
** object is being inserted.
** Reset the statement parameters.
*/

/* Initialise the smart large object specification structure */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
 BUFFER_LEN, 0, colname, BUFFER_LEN, &colname_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --

```

```

 SQLBindParameter failed (param 1)\n"))
 goto Exit;

 lospec_valsize = lospec_size;

 rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
 lospec_size, &lospec_valsize);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

 rc = SQLExecDirect (hstmt, "{call ifx_lo_col_info(?, ?)}", SQL_NTS);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLExecDirect failed\n"))
 goto Exit;

 /* Reset the statement parameters */
 rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLFreeStm failed\n"))
 goto Exit;

 fprintf(stdout, "STEP 3 done...smart large object specification
 structure initialised\n");

/* STEP 4. Get the size of the smart large object pointer structure.
** Allocate a buffer to hold the structure.
*/

 /* Get the size of the smart large object pointer structure */
 rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
 sizeof(loptr_size), NULL);
 if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 4 --
 SQLGetInfo failed\n"))
 goto Exit;

 /* Allocate a buffer to hold the smart large object pointer structure */
 loptr_buffer = malloc (loptr_size);

 fprintf (stdout, "STEP 4 done...smart large object pointer structure
 allocated\n");

/* STEP 5. Create a new smart large object.
** Reset the statement parameters.
*/

 /* Create a new smart large object */
 rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
 lospec_size, &lospec_valsize);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

```

```

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG,
 SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

loptr_valsize = loptr_size;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
 loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 3)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_SLONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 4)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_create(?, ?, ?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 5 done...smart large object created\n");

/* STEP 6. Open the file containing data for the new smart large object.
** Allocate a buffer to hold the smart large object data.
** Read data from the input file into the smart large object.
** data buffer
** Write data from the data buffer into the new smart large.
** object.
** Reset the statement parameters.
**
*/

/* Open the file containing data for the new smart large object */
hfile = open (lo_file_name, "rt");
/* sneaky way to get the size of the file */
write_size = lseek (open (lo_file_name, "rt"), 0L, SEEK_END);

/* Allocate a buffer to hold the smart large object data */
write_buffer = malloc (write_size + 1);

/* Read smart large object data from file */
read (hfile, write_buffer, write_size);

write_buffer[write_size] = '\0';
write_valsize = write_size;

```

```

/* Write data from the data buffer into the new smart large object */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
 (UDWORD)write_size, 0, write_buffer, write_size, &write_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_write(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 6 done...data written to new smart large
 object\n");

/* STEP 7. Insert the new smart large object into the database.
** Reset the statement parameters.
*/

/* Insert the new smart large object into the database */
loptr_valsize = loptr_size;

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
 loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLBindParameter failed\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, insertStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 7 done...smart large object inserted into the
 database\n");

```

```

/* STEP 8. Close the smart large object.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLBindParameter failed\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLExecDirect failed\n"))
 goto Exit;

fprintf (stdout, "STEP 8 done...smart large object closed\n");

/* STEP 9. Free the allocated buffers.
*/

free (lospec_buffer);
free (loptr_buffer);
free (write_buffer);

fprintf (stdout, "STEP 9 done...smart large object buffers freed\n");

Exit:

/* CLEANUP: Close the statement handle
** Free the statement handle
** Disconnect from the datasource
** Free the connection and environment handles
** Exit
*/

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
}

```

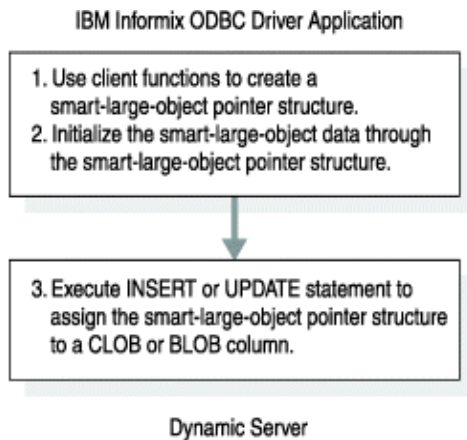
## Transfer smart-large-object data

An INSERT or UPDATE statement does not perform the actual input of the smart-large-object data. It does, however, provide a means for the application to identify which smart-large-object data to associate with the column.

A BLOB or CLOB column in a database table stores the smart-large-object pointer structure for a smart large object. Therefore, when you store a BLOB or CLOB column, you provide a smart-large-object pointer structure for the column in a **loptr** variable to the INSERT or UPDATE statement.

The following figure shows how an application transfers the data of a smart large object to the database server.

Figure 6. Transfer smart-large-object data from client application to database server



The smart large object that a smart-large-object pointer structure identifies exists if the smart-large-object pointer structure exists. When you store a smart-large-object pointer structure in a database, the database server deallocates the smart large object when appropriate.

If your application does not store the smart-large-object pointer structure for a new smart large object in the database, the smart-large-object pointer structure is only valid to access the version of the smart large object that was current when the pointer was passed to the application. If the smart large object is updated later, the pointer is invalid. The smart-large-object pointer structures that you store in a row do not expire when the object version changes.

When you retrieve a row and then update a smart large object that is contained in that row, the database server exclusively locks the row for the time that it updates the smart large object. Moreover, long updates for smart large objects (whether logging is enabled and whether they are associated with a table row) create the potential for a long transaction condition if the smart large object takes a long time to update or create.

The smart-large-object pointer structure, not the CLOB or BLOB data itself, is stored in a CLOB or BLOB column in the database. Therefore, SQL statements such as INSERT and SELECT accept and return a smart-large-object pointer structure as the column value for a smart-large-object column.

## Access a smart large object

This section describes how to select, open, delete, modify, and close a smart large object by using either the standard ODBC API or by using `ifx_lo` functions.

## Smart-large-object automation

Instead of accessing smart large objects with the `ifx_lo` functions, you can access smart large objects by using the standard ODBC API.

Operations supported when accessing smart large objects with the standard ODBC API include select, insert, update, and delete for CLOB and BLOB data types. You cannot access BYTE and TEXT simple large objects in this way.

## Set the access method using `SQL_INFX_ATTR_LO_AUTOMATIC`

You can use the `SQL_INFX_ATTR_LO_AUTOMATIC` attribute to tell the database server whether you will access smart large objects by using the ODBC API or the `ifx_lo` functions.

If the application enables the `SQL_INFX_ATTR_LO_AUTOMATIC` attribute as a connection attribute, all statements for that connection inherit the attribute value. To change this attribute value per statement, you have to set and reset it as a statement attribute. If you enable this attribute for the statement, the application can access the smart large object by using the standard ODBC way, as previously described. If you do not enable this attribute for the statement, the application accesses smart large objects by using `ifx_lo` functions. The application cannot use the `ifx_lo` functions if this attribute is enabled for the statement.

You can also enable the `SQL_INFX_ATTR_LO_AUTOMATIC` attribute by turning on the **Report Standard ODBC Types** option under the **Advanced** tab of the ODBC Administration for HCL OneDB™ Driver DSN.

`SQLDescribeCol` for a CLOB data type column returns `SQL_LONGVARCHAR` for the `DataPtrType`. `SQLDescribeCol` for a BLOB data type column returns `SQL_LONGVARBINARY`, if the `SQL_INFX_ATTR_LO_AUTOMATIC` attribute is enabled for that statement.

`SQLColAttributes` for a CLOB data type column returns `SQL_LONGVARCHAR` for the Field Identifier of `SQL_DESC_TYPE`, whereas for the BLOB data type column it returns `SQL_LONGVARBINARY` only if the `SQL_INFX_ATTR_LO_AUTOMATIC` attribute is enabled for that statement.

## Insert, update, and delete smart large objects using the ODBC API

When you insert, update, and delete either a CLOB or BLOB data type, the application binds the data type by using `SQLBindParameter` with a C type.

When you insert, update, or delete a CLOB data type, the application binds the CLOB data type by using `SQLBindParameter` with C type as **SQL\_C\_CHAR** and SQL type as `SQL_LONGVARCHAR`.

When you insert, update, or delete a BLOB data type, the application binds BLOB data type by using `SQLBindParameter` with C type as **SQL\_C\_BINARY** and SQL type as `SQL_LONGVARBINARY`.

HCL OneDB™ ODBC Driver performs insertion of smart large objects in the following way:

- The driver sends a request to the database server to create a smart large object on the server side in the form of a new file.
- The driver gets back the file descriptor (for example, `lofd`) of this file from the database server.

- The driver sends the preceding lofd file and the smart-large-object data that was bound by the application with SQLBindParameter to the database server.
- The database server writes the data onto the file.

## Select smart large objects using the ODBC API

When you select a CLOB data type, the application binds the C type of the column as **SQL\_C\_CHAR**. When you select a BLOB data type, the C type is bound as **SQL\_C\_BINARY**.

HCL OneDB™ ODBC Driver selects smart large objects in the following way:

- The driver sends a request to the database server to open the smart large object as a file on the server side.
- The driver gets back the file descriptor (for example, lofd) of this file from the database server.
- The driver sends the preceding lofd and a read request to the database server to read the smart-large-object data from the file.
- The database server reads the data from the corresponding file by using the preceding lofd and sends it to the driver.
- The driver writes the data to the buffer that was bound by the application with SQLBindParameter.

## The ifx\_lo functions

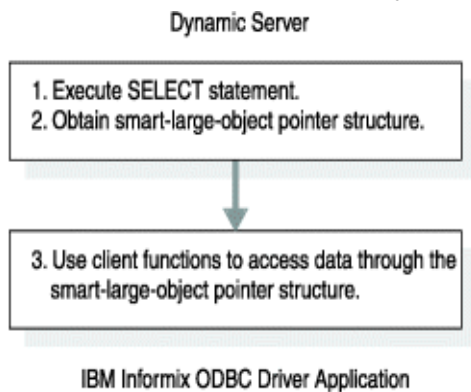
This section describes how to select, open, delete, modify, and close a smart large object by using ifx\_lo functions.

## Select a smart large object using ifx\_lo functions

A SELECT statement does not perform the actual output for the smart-large-object data. It does, however, establish a means for the application to identify a smart large object so that the application can then perform operations on the smart large object.

The following figure shows how the database server transfers the data of a smart large object to the application.

Figure 7. Transferring smart-large-object data from database server to client application



## Open a smart large object using ifx\_lo functions

When you open a smart large object, you obtain a smart-large-object file descriptor for the smart large object.



Through the smart-large-object file descriptor, you can access the data of a smart large object as if it were in an operating-system file.

## Access modes

When you open a smart large object, you specify the access mode for the data. The access mode determines which read and write operations are valid on the open smart large object.

The following table describes the access modes that `ifx_lo_open()` and `ifx_lo_create()` support.

| Access mode     | Purpose                                                                                                                                                                                                                                                                                                                                                                 | Constant      |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| Read only       | Only read operations are valid on the data.                                                                                                                                                                                                                                                                                                                             | LO_RDONLY     |
| Dirty read      | Lets you read uncommitted data pages for the smart large object. You cannot write to a smart large object after you set the mode to LO_DIRTY_READ. When you set this flag, you reset the current transaction isolation mode to dirty read for this smart large object.<br><br>Do not base updates on data that you obtain from a smart large object in dirty-read mode. | LO_DIRTY_READ |
| Write only      | Only write operations are valid on the data.                                                                                                                                                                                                                                                                                                                            | LO_WRONLY     |
| Append          | Intended for use with LO_WRONLY or LO_RDWR. Sets the location pointer to the end of the object immediately before each write. Appends any data you write to the end of the smart large object. If LO_APPEND is used alone, the object is opened for reading only.                                                                                                       | LO_APPEND     |
| Read/write      | Both read and write operations are valid on the data.                                                                                                                                                                                                                                                                                                                   | LO_RDWR       |
| Buffered access | Uses standard database server buffer pool.                                                                                                                                                                                                                                                                                                                              | LO_BUFFER     |
| Lightweight I/O | Uses private buffers from the session pool of the database server.                                                                                                                                                                                                                                                                                                      | LO_NOBUFFER   |

When you open a smart large object with LO\_APPEND only, the database server opens the smart large object as read-only. Seek operations and read operations move the file pointer. Write operations fail and do not move the file pointer.

You can mask the LO\_APPEND flag with another access mode. In any of these OR combinations, the seek operation remains unaffected. The following table shows the effect on the read and write operations that each of the OR combinations has.

| OR operation          | Read operations                                                                                       | Write operations                                                                                                                              |
|-----------------------|-------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| LO_RDONLY   LO_APPEND | Occur at the file position and then move the file position to the end of the data that has been read. | Fail and do not move the file position.                                                                                                       |
| LO_WRONLY   LO_APPEND | Fail and do not move the file position.                                                               | Move the file position to the end of the smart large object and then write the data; file position is at the end of the data after the write. |
| LO_RDWR   LO_APPEND   | Occur at the file position and then move the file position to the end of the data that has been read. | Move the file position to the end of the smart large object and then write the data; file position is at the end of the data after the write. |

## Lightweight I/O

When the database server accesses smart large objects, it uses buffers from the buffer pool for buffered access. Unbuffered access is called *lightweight I/O*.

Lightweight I/O uses private buffers instead of the buffer pool to hold smart large objects. These private buffers are allocated out of the database server session pool.

Lightweight I/O allows you to bypass the overhead of the least recently used (LRU) queues that the database server uses to manage the buffer pool. For more information about LRU queues, see your *HCL OneDB™ Performance Guide*.

You can specify lightweight I/O by setting the flags parameter to LO\_NOBUFFER when you create or open a smart large object. To specify buffered access, which is the default, use the LO\_BUFFER flag.



**Important:** Keep in mind the following issues when you use lightweight I/O:

- Close smart large objects with `ifx_lo_close()` when you finish with them to free memory allocated to the private buffers.
- All open operations that use lightweight I/O for a particular smart large object share the same private buffers. Consequently, one operation can cause the pages in the buffer to be flushed while other operations expect the object to be present in the buffer.

The database server imposes the following restrictions on switching from lightweight I/O to buffered I/O:

- You can use the `ifx_lo_alter()` function to switch a smart large object from lightweight I/O (`LO_NOBUFFER`) to buffered I/O (`LO_BUFFER`) if the smart large object is not open. However, `ifx_lo_alter()` generates an error if you try to change a smart large object that uses buffered I/O to one that uses lightweight I/O.
- Unless you first use `ifx_lo_alter()` to change the access mode to buffered access (`LO_BUFFER`), you can only open a smart large object that was created with lightweight I/O with the `LO_NOBUFFER` access-mode flag. If an open operation specifies `LO_BUFFER`, the database server ignores the flag.
- You can open a smart large object that has been created with buffered access (`LO_BUFFER`) with the `LO_NOBUFFER` flag only if you open the object in read-only mode. If you attempt to write to the object, the database server returns an error. To write to the smart large object, you must close it and then reopen it with the `LO_BUFFER` flag and an access flag that allows write operations.

You can use the database server utility `onspaces` to specify lightweight I/O for all smart large objects in an sbspace. For more information about the `onspaces` utility, see your *HCL OneDB™ Administrator's Guide*.

## Smart-large-object locks

To prevent simultaneous access to smart-large-object data, the database server locks a smart large object when you open it.

Locks on smart large objects are different from row locks. If you retrieve a smart large object from a row, the database server might hold a row lock as well as a smart-large-object lock. The database server locks smart large objects because many columns can contain the same smart-large-object data.

To specify the lock mode of a smart large object, pass the access-mode flags, `LO_RDONLY`, `LO_DIRTY_READ`, `LO_APPEND`, `LO_WRONLY`, `LO_RDWR`, and `LO_TRUNC`, to the `ifx_lo_open()` and `ifx_lo_create()` functions. When you specify `LO_RDONLY`, the database server places a lock on the smart-large-object data. When you specify `LO_DIRTY_READ`, the database server does not place a lock on the smart-large-object data. If you specify any other access-mode flag, the database server obtains an update lock, which it promotes to an exclusive lock on first write or other update operation.

Share and update locks (read-only mode or write mode before an update operation occurs) are held until your application takes one of the following actions:

- Closes the smart large object
- Commits the transaction or rolls it back

Exclusive locks are held until the end of a transaction even if you close the smart large object.



**Important:** You lose the lock at the end of a transaction even if the smart large object remains open. When the database server detects that a smart large object does not have an active lock, it places a new lock the next time



that you access the smart large object. The lock that it places is based on the original open mode of the smart large object.

## Duration of an open operation on a smart large object

After you open a smart large object with the `ifx_lo_create()` function or the `ifx_lo_open()` function, it remains open until certain events occurs.

A smart large object remains open until one of these events occur:

- The `ifx_lo_close()` function closes the smart large object.
- The session ends.



**Important:** The end of the current transaction does not close a smart large object. It does, however, release any lock on a smart large object.

Close smart large objects as soon as you finish with them. Leaving smart large objects open unnecessarily uses system memory. Leaving many smart large objects open can eventually produce an out-of-memory condition.

## Delete a smart large object

A smart large object cannot be deleted until certain conditions are met.

A smart large object is not deleted until both of the following conditions are met:

- The current transaction commits.
- The smart large object is closed, if the application opened the smart large object.

## Modifying a smart large object

You can modify a smart large object by using either an `UPDATE` or `INSERT` statement.

### About this task

To modify the data of a smart large object:

1. Read and write the data in the open smart large object.
2. Use an `UPDATE` or `INSERT` statement to store the smart-large-object pointer in the database.

## Close a smart large object

After you finish modifying a smart large object, call `ifx_lo_close()` to deallocate the resources that are assigned to it.

When the resources are freed, you can reallocate them to other structures that your application needs. You can also reallocate the smart-large-object file descriptor to other smart large objects.

## Example of retrieving a smart large object from the database using ifx\_lo functions

The code example, `loselect.c`, shows how to retrieve a smart large object from the database.

You can find the `loselect.c` file in the `%ONEDB_HOME%/demo/clidemo` directory on UNIX™ platforms and in the `%ONEDB_HOME%\demo\odbcdemo` directory on Windows™ platforms. You can also find instructions on how to build the **odbc\_demo** database in the same location.

```

/*
** loselect.c
**
** To access a smart large object
** SQLBindCol
** SQLBindParameter
** SQLConnect
** SQLFetch
** SQLFreeStmt
** SQLGetInfo
** SQLDisconnect
** SQLExecDirect
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define ERRMSG_LEN 200

UCHAR defDsn[] = "odbc_demo";

int checkError (SQLRETURN rc,
 SQLSMALLINT handleType,
 SQLHANDLE handle,
 char *errmsg)
{
 SQLRETURN retcode = SQL_SUCCESS;

 SQLSMALLINT errNum = 1;
 SQLCHAR sqlState[6];
 SQLINTEGER nativeError;
 SQLCHAR errMsg[ERRMSG_LEN];
 SQLSMALLINT textLengthPtr;

 if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
 {
 while (retcode != SQL_NO_DATA)
 {

```

```

 retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
 &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

 if (retcode == SQL_INVALID_HANDLE)
 {
 fprintf (stderr, "checkError function was called with an
 invalid handle!!\n");
 return 1;
 }

 if ((retcode == SQL_SUCCESS) || (retcode ==
 SQL_SUCCESS_WITH_INFO))
 fprintf (stderr, "ERROR: %d: %s : %s \n", nativeError,
 sqlState, errMsg);

 errNum++;
 }

 fprintf (stderr, "%s\n", errmsg);
 return 1; /* all errors on this handle have been reported */
}
else
 return 0; /* no errors to report */
}

int main (long argc,
 char *argv[])
{
 /* Declare variables
 */

 /* Handles */
 SQLHDBC hdbc;
 SQLHENV henv;
 SQLHSTMT hstmt;

 /* Smart large object file descriptor */
 long lofd;
 long lofd_valsize = 0;

 /* Smart large object pointer structure */
 char* loptr_buffer;
 short loptr_size;
 long loptr_valsize = 0;

 /* Smart large object status structure */
 char* lostat_buffer;
 short lostat_size;
 long lostat_valsize = 0;

 /* Smart large object data */
 char* lo_data;
 long lo_data_valsize = 0;

 /* Miscellaneous variables */
 UCHAR dsn[20]; /*name of the DSN used for connecting to the
 database*/

```

```

SQLRETURN rc = 0;
int in;

char* selectStmt = "SELECT advert FROM item WHERE item_num =
 1004";
long mode = LO_RDONLY;
long lo_size;
long cbMode = 0, cbLoSize = 0;

/* STEP 1. Get data source name from command line (or use default)
** Allocate the environment handle and set ODBC version
** Allocate the connection handle
** Establish the database connection
** Allocate the statement handle
*/

/* If(dsn is not explicitly passed in as arg) */
if (argc != 2)
{
 /* Use default dsn - odbc_demo */
 fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
 strcpy ((char *)dsn, (char *)defDsn);
}
else
{
 /* Use specified dsn */
 strcpy ((char *)dsn, (char *)argv[1]);
 fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
 fprintf (stdout, "Environment Handle Allocation
 failed\nExiting!!\n");
 return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
 (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
 SQLSetEnvAttr failed\nExiting!!\n"))
 return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
 Handle Allocation failed\nExiting!!\n"))
 return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
 failed\nExiting!!"))

```

```

 return (1);
/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
 Handle Allocation failed\nExiting!!"))
 return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2. Select a smart-large object from the database
** -- the select statement executed is -
** "SELECT advert FROM item WHERE item_num = 1004"
*/

/* Execute the select statement */
rc = SQLExecDirect (hstmt, selectStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLExecDirect failed\n"))
 goto Exit;

fprintf (stdout, "STEP 2 done...select statement executed...smart large
 object retrieved from the databse\n");

/* STEP 3. Get the size of the smart large object pointer structure.
** Allocate a buffer to hold the structure.
** Get the smart large object pointer structure from the
** database.
** Close the result set cursor.
*/

/* Get the size of the smart large object pointer structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
 sizeof(loptr_size),
 NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 3 -- SQLGetInfo
 failed\n"))
 goto Exit;

/* Allocate a buffer to hold the smart large object pointer structure */
loptr_buffer = malloc (loptr_size);

/* Bind the smart large object pointer structure buffer allocated to the
 column in the result set & fetch it from the database */
rc = SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size,
 &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindCol failed\n"))
 goto Exit;

rc = SQLFetch (hstmt);
if (rc == SQL_NO_DATA_FOUND)
{
 fprintf (stdout, "No Data Found\nExiting!!\n");
 goto Exit;
}

```



```

if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 -- SQLFetch
 failed\n"))
 goto Exit;

/* Close the result set cursor */
rc = SQLCloseCursor (hstmt);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLCloseCursor failed\n"))
 goto Exit;

fprintf (stdout, "STEP 3 done...smart large object pointer structure
 fetched from the database\n");

/* STEP 4. Use the smart large object's pointer structure to open it
** and obtain the smart large object file descriptor.
** Reset the statement parameters.
** */

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
 loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 3)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{? = call ifx_lo_open(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 4 done...smart large object opened... file
 descriptor obtained\n");

/* STEP 5. Get the size of the smart large object status structure.
** Allocate a buffer to hold the structure.
** Get the smart large object status structure from the
** database.

```

```

** Reset the statement parameters.
*/

/* Get the size of the smart large object status structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size,
 sizeof(lostat_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 5 -- SQLGetInfo
 failed\n"))
 goto Exit;

/* Allocate a buffer to hold the smart large object status structure. */
lostat_buffer = malloc(lostat_size);

/* Get the smart large object status structure from the database. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
 lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLExecDiret failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 5 done...smart large object status structure
 fetched from the database\n");

/* STEP 6. Use the smart large object's status structure to get the
** size of the smart large object.
** Reset the statement parameters.
*/

/* Use the smart large object status structure to get the size of the
 smart large object */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
 lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG,

```

```

 SQL_BIGINT, (UDWORD)0, 0, &lo_size, sizeof(lo_size), &cbLoSize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 6 done...smart large object size = %ld bytes\n",
 lo_size);

/* STEP 7. Allocate a buffer to hold the smart large object's data.
** Read the smart large object's data using its file descriptor.
** Null-terminate the last byte of the smart large-object's data.
** Print out the contents of the smart large object.
** Reset the statement parameters.
*/

/* Allocate a buffer to hold the smart large object's data chunks */
lo_data = malloc (lo_size + 1);

/* Read the smart large object's data */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR,
 lo_size, 0, lo_data, lo_size, &lo_data_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_read(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Null-terminate the last byte of the smart large objects data */
lo_data[lo_size] = '\0';

/* Print the contents of the smart large object */
fprintf (stdout, "Smart large object contents are.....\n\n\n%s\n\n\n",
 lo_data);

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

```

```

if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 7 done...smart large object read completely\n");

/* STEP 8. Close the smart large object.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLBindParameter failed\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLExecDirect failed\n"))
 goto Exit;

fprintf (stdout, "STEP 8 done...smart large object closed\n");

/* STEP 9. Free the allocated buffers.
*/

free (loptr_buffer);
free (lostat_buffer);
free (lo_data);

fprintf (stdout, "STEP 9 done...smart large object buffers freed\n");

Exit:

/* CLEANUP: Close the statement handle
** Free the statement handle
** Disconnect from the datasource
** Free the connection and environment handles
** Exit
*/

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

```

```

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
}

```

## Retrieve the status of a smart large object

The status information of a smart large object has corresponding client functions.

The following table describes the status information and the corresponding client functions.

| Disk-storage information              | Description                                                                                                                                                                          | Client functions         |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| Last access time                      | The time, in seconds, that a smart large object was last accessed.<br><br>This value is available only if the LO_KEEP_LASTACCESS_TIME flag is set for the smart large object.        | ifx_lo_stat_atime()      |
| Last time of status change            | The time, in seconds, of the last status change for a smart large object.<br><br>A change in status includes updates, changes in ownership, and changes to the number of references. | ifx_lo_stat_ctime()      |
| Last modification time (seconds)      | The time, in seconds, that a smart large object was last modified.                                                                                                                   | ifx_lo_stat_mtime_sec()  |
| Last modification time (microseconds) | The microsecond component of the time of last modification.<br><br>This value is only supported on platforms that provide system time to microsecond granularity.                    | ifx_lo_stat_mtime_usec() |
| Reference count                       | A count of the number of references to a smart large object.                                                                                                                         | ifx_lo_stat_refcnt()     |
| Size                                  | The size, in bytes, of a smart large object.                                                                                                                                         | ifx_lo_stat_size()       |

The time values (such as last access time and last change time) might differ slightly from the system time. This difference is due to the algorithm that the database server uses to obtain the time from the operating system.

## Example of retrieving information about a smart large object

The code example, `loinfo.c`, shows how to retrieve information about a smart large object.

You can find the `loinfo.c` file in the `%ONEDB_HOME%/demo/clidemo` directory on UNIX™ platforms and in the `%ONEDB_HOME%\demo\odbcdemo` directory in Windows™ environments. You can also find instructions on how to build the `odbc_demo` database in the same location.

```

/*
** loinfo.c
**
** To check the status of a smart large object
**
** ODBC Functions:
** SQLBindCol
** SQLBindParameter
** SQLConnect
** SQLFetch
** SQLFreeStmt
** SQLDisconnect
** SQLExecDirect
**
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN 20
#define ERRMSG_LEN 200

UCHAR defDsn[] = "odbc_demo";

int checkError (SQLRETURN rc,
 SQLSMALLINT handleType,
 SQLHANDLE handle,
 char *errmsg)
{
 SQLRETURN retcode = SQL_SUCCESS;

 SQLSMALLINT errNum = 1;
 SQLCHAR sqlState[6];
 SQLINTEGER nativeError;
 SQLCHAR errMsg[ERRMSG_LEN];
 SQLSMALLINT textLengthPtr;

 if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
 {
 while (retcode != SQL_NO_DATA)
 {
 retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
 &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);
 }
 }
}

```

```

 if (retcode == SQL_INVALID_HANDLE)
 {
 fprintf (stderr, "checkError function was called with an
 invalid handle!!\n");
 return 1;
 }

 if ((retcode == SQL_SUCCESS) || (retcode ==
 SQL_SUCCESS_WITH_INFO))
 fprintf (stderr, "ERROR: %d: %s : %s \n", nativeError,
 sqlState, errMsg);

 errNum++;
}

fprintf (stderr, "%s\n", errMsg);
return 1; /* all errors on this handle have been reported */
}
else
 return 0; /* no errors to report */
}

int main (long argc,
 char *argv[])
{
 /* Declare variables
 */

 /* Handles */
 SQLHDBC hdbc;
 SQLHENV henv;
 SQLHSTMT hstmt;

 /* Smart large object file descriptor */
 long lofd;
 long lofd_valsize = 0;

 /* Smart large object specification structure */
 char* lospec_buffer;
 short lospec_size;
 long lospec_valsize = 0;

 /* Smart large object status structure */
 char* lostat_buffer;
 short lostat_size;
 long lostat_valsize = 0;

 /* Smart large object pointer structure */
 char* loptr_buffer;
 short loptr_size;
 long loptr_valsize = 0;

 /* Miscellaneous variables */
 UCHAR dsn[20]; /*name of the DSN used for connecting to the
 database*/
 SQLRETURN rc = 0;
 int in;

```

```

char* selectStmt = "SELECT advert FROM item WHERE item_num =
 1004";
long lo_size;
long mode = LO_RDONLY;

char sbspace_name[BUFFER_LEN];
long sbspace_name_size = SQL_NTS;

long cbMode = 0, cbLoSize = 0;

/* STEP 1. Get data source name from command line (or use default).
** Allocate the environment handle and set ODBC version.
** Allocate the connection handle.
** Establish the database connection.
** Allocate the statement handle.
*/

/* If (dsn is not explicitly passed in as arg) */
if (argc != 2)
{
 /* Use default dsn - odbc_demo */
 fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
 strcpy ((char *)dsn, (char *)defDsn);
}
else
{
 /* Use specified dsn */
 strcpy ((char *)dsn, (char *)argv[1]);
 fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}

/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
 fprintf (stdout, "Environment Handle Allocation
 failed\nExiting!!\n");
 return (1);
}

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
 (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
 SQLSetEnvAttr failed\nExiting!!\n"))
 return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
 Handle Allocation failed\nExiting!!\n"))
 return (1);

/* Establish the database connection */

```



```

rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
 failed\nExiting!!"))
 return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
 Handle Allocation failed\nExiting!!"))
 return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2. Select a smart-large object from the database.
** -- the select statement executed is -
** "SELECT advert FROM item WHERE item_num = 1004"
*/

/* Execute the select statement */
rc = SQLExecDirect (hstmt, selectStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLExecDirect failed\n"))
 goto Exit;

fprintf (stdout, "STEP 2 done...select statement executed...smart large
 object retrieved from the databse\n");

/* STEP 3. Get the size of the smart large object pointer structure.
** Allocate a buffer to hold the structure.
** Get the smart large object pointer structure from the database.
** Close the result set cursor.
*/

/* Get the size of the smart large object pointer structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_PTR_LENGTH, &loptr_size,
 sizeof(loptr_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 3 -- SQLGetInfo
 failed\n"))
 goto Exit;

/* Allocate a buffer to hold the smart large object pointer structure */
loptr_buffer = malloc (loptr_size);

/* Bind the smart large object pointer structure buffer allocated to the
 column in the result set & fetch it from the database */
rc = SQLBindCol (hstmt, 1, SQL_C_BINARY, loptr_buffer, loptr_size,
 &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindCol failed\n"))
 goto Exit;

rc = SQLFetch (hstmt);
if (rc == SQL_NO_DATA_FOUND)
{
 fprintf (stdout, "No Data Found\nExiting!!\n");
}

```

```

 goto Exit;
}
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 -- SQLFetch
 failed\n"))
 goto Exit;

/* Close the result set cursor */
rc = SQLCloseCursor (hstmt);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLCloseCursor failed\n"))
 goto Exit;

fprintf (stdout, "STEP 3 done...smart large object pointer structure
 fetched from the database\n");

/* STEP 4. Use the smart large object's pointer structure to open it
** and obtain the smart large object file descriptor.
** Reset the statement parameters.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)loptr_size, 0, loptr_buffer,
 loptr_size, &loptr_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &mode, sizeof(mode), &cbMode);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 3)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{? = call ifx_lo_open(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 4 done...smart large object opened... file
 descriptor obtained\n");

/* STEP 5. Get the size of the smart large object status structure.
** Allocate a buffer to hold the structure.

```

```

** Get the smart large object status structure from the database.
** Reset the statement parameters.
*/

/* Get the size of the smart large object status structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_STAT_LENGTH, &lostat_size,
 sizeof(lostat_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 5 -- SQLGetInfo
 failed\n"))
 goto Exit;

/* Allocate a buffer to hold the smart large object status structure. */
lostat_buffer = malloc(lostat_size);

/* Get the smart large object status structure from the database. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
 lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 5 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 5 done...smart large object status structure
 fetched from the database\n");

/* STEP 6. Use the smart large object's status structure to get the size
** of the smart large object.
** Reset the statement parameters.
** You can use additional ifx_lo_stat_*() functions to get more
** status information about the smart large object.
** You can also use it to retrieve the smart large object
** specification structure and get further information about the
** smart large object using its specification structure.
*/

/* Use the smart large object status structure to get the size of the
 smart large object. */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,

```

```

 lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_LONG,
 SQL_BIGINT, (UDWORD)0, 0, &lo_size, sizeof(lo_size), &cbLoSize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_size(?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 6 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "LARGE OBJECT SIZE = %ld\n", lo_size);
fprintf (stdout, "STEP 6 done...smart large object size retrieved\n");

/* STEP 7. Get the size of the smart large object specification structure.
** Allocate a buffer to hold the structure.
** Get the smart large object specification structure from the
** database.
** Reset the statement parameters.
**
*/

/* Get the size of the smart large object specification structure */
rc = SQLGetInfo (hdbc, SQL_INFX_LO_SPEC_LENGTH, &lospec_size,
 sizeof(lospec_size), NULL);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 7 -- SQLGetInfo
 failed\n"))
 goto Exit;

/* Allocate a buffer to hold the smart large object specification
 structure */
lospec_buffer = malloc (lospec_size);

/* Get the smart large object specification structure from the
 database */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lostat_size, 0, lostat_buffer,
 lostat_size, &lostat_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lospec_size, 0, lospec_buffer,
 lospec_size, &lospec_valsize);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --

```

```

 SQLBindParameter failed (param 2)\n"))
 goto Exit;

 rc = SQLExecDirect (hstmt, "{call ifx_lo_stat_cspect(?, ?)}", SQL_NTS);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 7 --
 SQLExecDirect failed\n"))
 goto Exit;

 fprintf (stdout, "STEP 7 done...smart large object status structure
 fetched from the database\n");

/* STEP 8. Use the smart large object's specification structure to get
** the sbspace name where the smart large object is stored.
** Reset the statement parameters.
*/

/* Use the smart large object's specification structure to get the
 sbspace name of the smart large object. */
 rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_UDT_FIXED, (UDWORD)lo_spec_size, 0, lo_spec_buffer,
 lo_spec_size, &lo_spec_valsize);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

 rc = SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR,
 BUFFER_LEN, 0, sbspace_name, BUFFER_LEN, &sbspace_name_size);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

 rc = SQLExecDirect (hstmt, "{call ifx_lo_specget_sbspace(?, ?)}",
 SQL_NTS);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 8 --
 SQLExecDirect failed\n"))
 goto Exit;

 fprintf (stdout, "LARGE OBJECT SBSPACE NAME = %s\n", sbspace_name);
 fprintf (stdout, "STEP 8 done...large object sbspace name retrieved\n");

/* STEP 9. Close the smart large object.
*/

 rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_INTEGER, (UDWORD)0, 0, &lofd, sizeof(lofd), &lofd_valsize);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 9 --
 SQLBindParameter failed\n"))
 goto Exit;

 rc = SQLExecDirect (hstmt, "{call ifx_lo_close(?)}", SQL_NTS);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 9 --
 SQLExecDirect failed\n"))
 goto Exit;

 fprintf (stdout, "STEP 9 done...smart large object closed\n");

```

```

/* STEP 10.Free the allocated buffers.
*/

free (loptr_buffer);
free (lostat_buffer);
free (lospec_buffer);

fprintf (stdout, "STEP 10 done...smart large object buffers freed\n");

Exit:

/* CLEANUP: Close the statement handle.
** Free the statement handle.
** Disconnect from the datasource.
** Free the connection and environment handles.
** Exit.
*/

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);
}

```

## Read or write a smart large object to or from a file

You can use the SQL functions to read or write a smart large object to or from a file.

You can use the SQL functions FILETOBLOB() and FILETOCLOB() to transfer data from a file to a smart large object. The file can be on a client computer or on a server computer.

You can use the SQL function LOTOFILE() to transfer data from a smart large object to a file. The file might be on a client computer or on a server computer. LOTOFILE() accepts a smart-large-object pointer as a parameter. You can use the smart-large-object pointer structure for this parameter.

For more information about these SQL functions, see the *HCL OneDB™ Guide to SQL: Syntax*.

## Rows and collections

Rows and collections are composite values that consist of one or more elements.

The information in these topics apply only if your database server is .

You can use the SELECT, UPDATE, INSERT, and DELETE statements to access an entire row or collection. However, these SQL statements do not let you access an element that is in a row or collection. To access an element, you need to retrieve the row or collection and then access the element from the local copy of the row or collection.

For more information about rows and collections, see the *HCL OneDB™ Guide to SQL: Reference*, and *HCL OneDB™ User-Defined Routines and Data Types Developer's Guide*.

## Allocating and binding a row or collection buffer

When you retrieve a row or collection, the database server puts the row or collection into a buffer that is local to your HCL OneDB™ ODBC Driver application.

### About this task

To allocate and bind a row or collection buffer:

1. Call `ifx_rc_create()` to allocate the buffer.
2. Call `SQLBindCol()` to bind the buffer handle to the database column.
3. Execute a SELECT statement to transfer the row or collection data to the local buffer.
4. Use the row or collection buffer.
5. Call `ifx_rc_free()` to deallocate the buffer.

## Fixed-type buffers and unfixed-type buffers

There are several differences between fixed-type buffers and unfixed-type buffers.

The following table describes the differences between fixed-type buffers and unfixed-type buffers.

| Buffer       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fixed type   | <p>When you call <code>ifx_rc_create()</code> to create a row or collection buffer, you specify the following data types for the buffer:</p> <ul style="list-style-type: none"> <li>• The buffer data type (a row or one of the collection types)</li> <li>• The data types of the elements that are in the row or collection</li> </ul> <p>When you retrieve the row or collection, the database server compares the source and target data types and converts data from one HCL OneDB™ SQL data type to another as necessary.</p> <p>You can modify the row or collection buffer before you retrieve data into the buffer.</p> |
| Unfixed type | <p>When you call <code>ifx_rc_create()</code> to create a row or collection buffer, you specify only the buffer data type (a row or a collection) and not the element types.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

| Buffer | Description                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | <p>When you retrieve the row or collection, the database server does not compare data types because you did not specify the target data types. Instead, the row or collection buffer adopts the data types of the source data.</p> <p>You must initialize the row or collection buffer before you modify it. To initialize the buffer, retrieve a row or collection into it.</p> <p>The buffer type remains unfixed even when it contains data.</p> |

## Buffers and memory allocation

When you retrieve data into a buffer that already contains a row or collection, HCL OneDB™ ODBC Driver does not reuse the same buffer.

Instead, HCL OneDB™ ODBC Driver performs the following steps:

1. Creates a row or collection buffer.
2. Associates the new buffer with the given buffer handle.
3. Deallocates the original buffer.

## SQL data

The database server calls cast functions to convert the data from the source HCL OneDB™ SQL data types to the target HCL OneDB™ SQL data types.

If the data types for a row or collection that are on a database server differ from the data types for a row or collection buffer into which the data is retrieved, the database server calls cast functions to convert the data from the source HCL OneDB™ SQL data types to the target HCL OneDB™ SQL data types. The following table lists the provider of the cast functions for each combination of source data type and target data type. Cast functions that a data type provides are located on the database server.

| Source data type | Target data type | Provider of cast functions |
|------------------|------------------|----------------------------|
| Built-in         | Built-in         | Database server            |
| Built-in         | Extended         | Data type                  |
| Extended         | Built-in         | Data type                  |
| Extended         | Extended         | Data type                  |

## Performing a local fetch

HCL OneDB™ ODBC Driver performs a local fetch when you retrieve a row or collection from one location on the client computer to another location on the client computer.

### About this task



A local fetch has the following limits on SQL data conversion:

- HCL OneDB™ ODBC Driver cannot convert extended data types for which the cast functions are on a database server.
- HCL OneDB™ ODBC Driver cannot convert data from one named row type to another. Only the database server can perform this type of conversion.
- HCL OneDB™ ODBC Driver cannot convert SQL data types when retrieving an entire row or collection. Thus, HCL OneDB™ ODBC Driver can perform a local fetch of an entire row or collection only if the internal structures for the source and destination are the same or if the destination is an unfixed-type buffer.

For example, if you define a local collection as **list (char(1) not null)**, the database server can put a **list (int not null)** value from the database server into the local collection. During this operation, the database server converts each integer into a string and constructs a new list to return to the client computer. You cannot perform this operation on the client computer where you retrieve a local list of integers into a list of characters.

To perform a local fetch:

1. Call `ifx_rc_create()` to allocate a row or collection buffer.
2. Call `SQLBindCol()` to bind the buffer handle to the local row or collection.
3. Execute a `SELECT` statement to transfer the row or collection data to the local buffer.
4. For each element in the row or collection, call `ifx_rc_fetch()` to copy the value to the buffer.
5. Use the row or collection buffer.
6. Call `ifx_rc_free()` to deallocate the buffer.

## Example of retrieving row and collection data from the database

The sample program, `rcselect.c`, retrieves row and collection data from the database and displays it.

This example also illustrates the fact that the same client functions can use row and collection handles interchangeably.

You can find the `rcselect.c` file in the `%ONEDB_HOME%/demo/clidemo` directory on UNIX™ and in the `%ONEDB_HOME%\demo\odbcdemo` directory in Windows™. You can also find instructions on how to build the **odbc\_demo** database in the same location.

```

/*
** rcselect.c
**
** To access rows and collections
** ODBC Functions:
** SQLBindParameter
** SQLConnect
** SQLDisconnect
** SQLExecDirect
** SQLFetch
** SQLFreeStmt
**
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#ifndef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxccli.h"

#define BUFFER_LEN 25
#define ERRMSG_LEN 200

UCHAR defDsn[] = "odbc_demo";

int checkError (SQLRETURN rc,
 SQLSMALLINT handleType,
 SQLHANDLE handle,
 char *errmsg)
{
 SQLRETURN retcode = SQL_SUCCESS;

 SQLSMALLINT errNum = 1;
 SQLCHAR sqlState[6];
 SQLINTEGER nativeError;
 SQLCHAR errMsg[ERRMSG_LEN];
 SQLSMALLINT textLengthPtr;

 if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
 {
 while (retcode != SQL_NO_DATA)
 {
 retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
 &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

 if (retcode == SQL_INVALID_HANDLE)
 {
 fprintf (stderr, "checkError function was called with an
 invalid handle!!\n");
 return 1;
 }

 if ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO))
 fprintf (stderr, "ERROR: %d: %s : %s \n", nativeError,
 sqlState, errMsg);

 errNum++;
 }

 fprintf (stderr, "%s\n", errMsg);
 return 1; /* all errors on this handle have been reported */
 }
 else
 return 0; /* no errors to report */
}

/*
** Executes the given select statement and assumes the results will be
** either rows or collections. The 'hrc' parameter may reference either

```

```

** a row or a collection. Rows and collection handles may often be used
** interchangeably.
**
** Each row of the select statement will be fetched into the given row or
** collection handle. Then each field of the row or collection will be
** individually converted into a character buffer and displayed.
**
** This function returns 0 if an error occurs, else returns 1
**
*/

int do_select (SQLHDBC hdbc,
 char* select_str,
 HINFX_RC hrc)
{
 SQLHSTMT hRCStmt;
 SQLHSTMT hSelectStmt;
 SQLRETURN rc = 0;

 short index, rownum;
 short position = SQL_INFX_RC_ABSOLUTE;
 short jump;

 char fname[BUFFER_LEN];
 char lname[BUFFER_LEN];
 char rc_data[BUFFER_LEN];

 SQLINTEGER cbFname = 0, cbLname = 0, cbHrc = 0;
 SQLINTEGER cbPosition = 0, cbJump = 0, cbRCData = 0;

 /* STEP A. Allocate the statement handles for the select statement and
 ** the statement used to retrieve the row/collection data.
 */

 /* Allocate the statement handle */
 rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hRCStmt);
 if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step A -- Statement
 Handle Allocation failed for row/collection
 statement\nExiting!!"))
 return 0;

 /* Allocate the statement handle */
 rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hSelectStmt);
 if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step A -- Statement
 Handle Allocation failed for select statement\nExiting!!"))
 return 0;

 fprintf (stdout, "STEP A done...statement handles allocated\n");

 /* STEP B. Execute the select statement.
 ** Bind the result set columns -
 ** -- col1 = fname
 ** col2 = lname
 ** col3 = row/collection data
 */

```

```

/* Execute the select statement */
rc = SQLExecDirect (hSelectStmt, select_str, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
 SQLExecDirect failed\n"))
 return 0;

/* Bind the result set columns */
rc = SQLBindCol (hSelectStmt, 1, SQL_C_CHAR, (SQLPOINTER)fname,
 BUFFER_LEN, &cbFname);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
 SQLBindCol failed for column 'fname'\n"))
 return 0;

rc = SQLBindCol (hSelectStmt, 2, SQL_C_CHAR, (SQLPOINTER)lname,
 BUFFER_LEN, &cbLname);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
 SQLBindCol failed for column 'lname'\n"))
 return 0;

rc = SQLBindCol (hSelectStmt, 3, SQL_C_BINARY, (SQLPOINTER)hrc,
 sizeof(HINFX_RC), &cbHrc);
if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in Step B --
 SQLBindCol failed for row/collection column\n"))
 return 0;

fprintf (stdout, "STEP B done...select statement executed and result set
 columns bound\n");

/* STEP C. Retrieve the results.
*/

for (rownum = 1;; rownum++)
{
 rc = SQLFetch (hSelectStmt);
 if (rc == SQL_NO_DATA_FOUND)
 {
 fprintf (stdout, "No data found...\n");
 break;
 }
 else if (checkError (rc, SQL_HANDLE_STMT, hSelectStmt, "Error in
 Step C -- SQLFetch failed\n"))
 return 0;

 fprintf(stdout, "Retrieving row number %d:\n\tfname -- %s\n\tlname --
 %s\n\tRow/Collection Data --\n", rownum, fname, lname);

 /* For each row in the result set, display each field of the
 retrieved row/collection */
 for (index = 1;; index++)
 {
 strcpy(rc_data, "<null>");

 /* Each value in the local row/collection will be fetched into a
 * character buffer and displayed using fprintf().
 */

```

```

rc = SQLBindParameter (hRCStmt, 1, SQL_PARAM_OUTPUT, SQL_C_CHAR,
 SQL_CHAR, 0, 0, rc_data, BUFFER_LEN, &cbRCData);
if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
 SQLBindParameter failed (param 1)\n"))
 return 0;

rc = SQLBindParameter (hRCStmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrc,
 sizeof(HINFX_RC), &cbHrc);
if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
 SQLBindParameter failed (param 2)\n"))
 return 0;

rc = SQLBindParameter (hRCStmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
 SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
 SQLBindParameter failed (param 3)\n"))
 return 0;

jump = index;
rc = SQLBindParameter (hRCStmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
 SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in Step C --
 SQLBindParameter failed (param 4)\n"))
 return 0;

rc = SQLExecDirect(hRCStmt, "{ ? = call ifx_rc_fetch(?, ?, ?)
 SQL_NTS);
if (rc == SQL_NO_DATA_FOUND)
{
 break;
}
else if (checkError (rc, SQL_HANDLE_STMT, hRCStmt, "Error in
 Step C -- SQLExecDirect failed\n"))
 return 0;

/* Display retrieved row */
fprintf(stdout, "\t\t%d: %s\n", index, rc_data);
}
}

fprintf (stdout, "STEP C done...results retrieved\n");

/* Free the statement handles */
SQLFreeHandle (SQL_HANDLE_STMT, hSelectStmt);
SQLFreeHandle (SQL_HANDLE_STMT, hRCStmt);

return 1; /* no error */
}

/*
 * This function allocates the row and collection buffers, passes
 * them to the do_select() function, along with an appropriate select
 * statement and then frees all allocated handles.
 */
int main (long argc,

```

```

 char *argv[])
{
 /* Declare variables
 */

 /* Handles */
 SQLHDBC hdbc;
 SQLHENV henv;
 SQLHSTMT hstmt;
 HINFX_RC hrow, hlist;

 /* Miscellaneous variables */

 UCHAR dsn[20]; /*name of the DSN used for connecting to the
 database*/
 SQLRETURN rc = 0;
 int in;

 int data_size = SQL_NTS;
 char* listSelectStmt = "SELECT fname, lname, contact_dates FROM
 customer";
 char* rowSelectStmt = "SELECT fname, lname, address FROM
 customer";

 SQLINTEGER cbHlist = 0, cbHrow = 0;

 /* STEP 1. Get data source name from command line (or use default).
 ** Allocate environment handle and set ODBC version.
 ** Allocate connection handle.
 ** Establish the database connection.
 ** Allocate the statement handle.
 */

 /* If(dsn is not explicitly passed in as arg) */
 if (argc != 2)
 {
 /* Use default dsn - odbc_demo */
 fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
 strcpy ((char *)dsn, (char *)defDsn);
 }
 else
 {
 /* Use specified dsn */
 strcpy ((char *)dsn, (char *)argv[1]);
 fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
 }

 /* Allocate the Environment handle */
 rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
 if (rc != SQL_SUCCESS)
 {
 fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
 return (1);
 }

 /* Set the ODBC version to 3.5 */

```

```

rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
 (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
 SQLSetEnvAttr failed\nExiting!!"))
 return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
 Handle Allocation failed\nExiting!!"))
 return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
 failed\n"))
 return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
 Handle Allocation failed\nExiting!!"))
 return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2. Allocate an unfixed collection handle.
** Retrieve database rows containing a list.
** Reset the statement parameters.
*/

/* Allocate an unfixed list handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
 SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC),
 &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter (param 1) failed\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
 0, 0, (UCHAR *) "list", 0, &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter (param 2) failed\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{? = call ifx_rc_create(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Retrieve database rows containing a list */
if (!do_select (hdbc, listSelectStmt, hlist))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);

```

```

if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 2 done...list data retrieved\n");
fprintf (stdout, "\nHit <Enter> to continue...");
in = getchar ();

/* STEP 3. Allocate an unfixed row handle.
** Retrieve database rows containing a row.
** Reset the statement parameters.
*/

/* Allocate an unfixed row handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
 SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC),
 &cbHrow);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 1) failed\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
 0, 0, (UCHAR *) "row", 0, &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 2) failed\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, "{? = call ifx_rc_create(?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLExecDirect failed\n"))
 goto Exit;

/* Retrieve databse rows containing a row */
if (!do_select (hdbc, rowSelectStmt, hrow))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 3 done...row data retrieved\n");

/* STEP 4. Free the row and list handles.
*/

/* Free the row handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
 &cbHrow);

rc = SQLExecDirect (hstmt, (UCHAR *) "{call ifx_rc_free(?)}", SQL_NTS);

/* Free the list handle */
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,

```



```

 SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
 &cbHlist);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free?}", SQL_NTS);

fprintf (stdout, "STEP 4 done...row and list handles freed\n");

Exit:

/* CLEANUP: Close the statement handle.
** Free the statement handle.
** Disconnect from the datasource.
** Free the connection and environment handles.
** Exit.
**
*/

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);
fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);

```

## Example of creating a row and a list on the client

The code example, `rccreate.c`, creates a row and a list on the client, adds items to them, and inserts them into the database.

You can find the `rccreate.c` file in the `%ONEDB_HOME%/demo/clidemo` directory on UNIX™ and in the `%ONEDB_HOME%\demo\odbcdemo` directory in Windows™. You can also find instructions on how to build the **odbc\_demo** database in the same location.

```

/*
** rccreate.c
**
** To create a collection & insert it into the database table
**
**
** ODBC Functions:
** SQLBindParameter
** SQLConnect
** SQLDisconnect
** SQLExecDirect
**
*/

#include <stdio.h>
#include <stdlib.h>

```

```

#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#include "infxcli.h"

#define BUFFER_LEN 25
#define ERRMSG_LEN 200

UCHAR defDsn[] = "odbc_demo";

int checkError (SQLRETURNrc,
 SQLSMALLINT handleType,
 SQLHANDLE handle,
 char *errmsg)
{
 SQLRETURN retcode = SQL_SUCCESS;

 SQLSMALLINT errNum = 1;
 SQLCHAR sqlState[6];
 SQLINTEGER nativeError;
 SQLCHAR errMsg[ERRMSG_LEN];
 SQLSMALLINT textLengthPtr;
 if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
 {
 while (retcode != SQL_NO_DATA)
 {
 retcode = SQLGetDiagRec (handleType, handle, errNum, sqlState,
 &nativeError, errMsg, ERRMSG_LEN, &textLengthPtr);

 if (retcode == SQL_INVALID_HANDLE)
 {
 fprintf (stderr, "checkError function was called with an
 invalid handle!!\n");
 return 1;
 }

 if ((retcode == SQL_SUCCESS) || (retcode ==
 SQL_SUCCESS_WITH_INFO)) fprintf (stderr, "ERROR: %d: %s
 : %s \n", nativeError, sqlState, errMsg);

 errNum++;
 }

 fprintf (stderr, "%s\n", errMsg);
 return 1; /* all errors on this handle have been reported */
 }
 else
 return 0; /* no errors to report */
}

int main (long argc,
 char *argv[])

```

```

{
/* Declare variables
*/

/* Handles */
SQLHDB hdbc;
SQLHENV henv;
SQLHSTMT hstmt;

HINFX_RC hrow;
HINFX_RC hlist;

/* Miscellaneous variables */
UCHAR dsn[20];/*name of the DSN used for connecting to the
 database*/
SQLRETURN rc = 0;
int i, in;
int data_size = SQL_NTS;
short position = SQL_INFX_RC_ABSOLUTE;
short jump;

UCHAR row_data[4][BUFFER_LEN] = {"520 Topaz Way", "Redwood City",
 "CA", "94062"};
int row_data_size = SQL_NTS;

UCHAR list_data[2][BUFFER_LEN] = {"1991-06-20", "1993-07-17"};
int list_data_size = SQL_NTS;

char* insertStmt = "INSERT INTO customer VALUES (110, 'Roy',
 'Jaeger', ?, ?)";
SQLINTEGER cbHrow = 0, cbHlist = 0, cbPosition = 0, cbJump = 0;
/* STEP 1. Get data source name from command line (or use default).
** Allocate environment handle and set ODBC version.
** Allocate connection handle.
** Establish the database connection.
** Allocate the statement handle.
*/

/* If(dsn is not explicitly passed in as arg) */
if (argc != 2)
{
/* Use default dsn - odbc_demo */
fprintf (stdout, "\nUsing default DSN : %s\n", defDsn);
strcpy ((char *)dsn, (char *)defDsn);
}
else
{
/* Use specified dsn */
strcpy ((char *)dsn, (char *)argv[1]);
fprintf (stdout, "\nUsing specified DSN : %s\n", dsn);
}
/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
return (1);
}
}

```

```

/* Set the ODBC version to 3.5 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION,
 (SQLPOINTER)SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 --
 SQLSetEnvAttr failed\nExiting!!"))
 return (1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, "Error in Step 1 -- Connection
 Handle Allocation failed\nExiting!!"))
 return (1);

/* Establish the database connection */
rc = SQLConnect (hdbc, dsn, SQL_NTS, "", SQL_NTS, "", SQL_NTS);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- SQLConnect
 failed\n"))
 return (1);

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, "Error in Step 1 -- Statement
 Handle Allocation failed\nExiting!!"))
 return (1);

fprintf (stdout, "STEP 1 done...connected to database\n");

/* STEP 2. Allocate fixed-type row handle -- this creates a non-null row
** buffer, each of whose values is null, and can be updated.
** Allocate a fixed-type list handle -- this creates a non-null
** but empty list buffer into which values can be inserted.
** Reset the statement parameters.
** */

/* Allocate a fixed-type row handle -- this creates a row with each
value empty */

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
 SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, &hrow, sizeof(HINFX_RC),
 &cbHrow);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter (param 1) failed for row handle\n")) goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
 0, 0, (UCHAR *) "ROW(address1 VARCHAR(25), city VARCHAR(15), state
 VARCHAR(15), zip VARCHAR(5))", 0, &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter (param 2) failed for row handle\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, (UCHAR *) "{? = call ifx_rc_create(?)",
 SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLExecDirect failed for row handle\n"))
 goto Exit;

```

```

/* Allocate a fixed-type list handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_BINARY,
 SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, &hlist, sizeof(HINFX_RC),
 &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter (param 1) failed for list handle\n"))
 goto Exit;

data_size = SQL_NTS;
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
 0, 0, (UCHAR *) "LIST (DATETIME YEAR TO DAY NOT NULL)",0,
 &data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLBindParameter (param 2) failed for list handle\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, (UCHAR *) "{? = call ifx_rc_create(?)}",
 SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLExecDirect failed for list handle\n"))
 goto Exit;

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 2 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 2 done...fixed-type row and collection handles
 allocated\n");

/* STEP 3. Update the elements of the fixed-type row buffer allocated.
** Insert elements into the fixed-type list buffer allocated.
** Reset the statement parameters.
*/

/* Update elements of the row buffer */
for (i=0; i<4; i++)
{
 rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
 &cbHrow);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 1) failed for row handle\n"))
 goto Exit;

 rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
 SQL_CHAR, BUFFER_LEN, 0, row_data[i], 0, &row_data_size);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 2) failed for row handle\n"))
 goto Exit;

 rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
 SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
 if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 3) failed for row handle\n"))

```

```

 goto Exit; jump = i + 1;
rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
 SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 4) failed for row handle\n"))
 goto Exit;

rc = SQLExecDirect (hstmt,
 (UCHAR *)"{call ifx_rc_update(?, ?, ?, ?)", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLExecDirect failed for row handle\n"))
 goto Exit;
}

/* Insert elements into the list buffer */
for (i=0; i<2; i++)
{
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
 &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 1) failed for list handle\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
 SQL_DATE, 25, 0, list_data[i], 0, &list_data_size);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 2) failed for list handle\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 3, SQL_PARAM_INPUT, SQL_C_SHORT,
 SQL_SMALLINT, 0, 0, &position, 0, &cbPosition);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 3) failed for list handle\n"))
 goto Exit;

jump = i + 1;
rc = SQLBindParameter (hstmt, 4, SQL_PARAM_INPUT, SQL_C_SHORT,
 SQL_SMALLINT, 0, 0, &jump, 0, &cbJump);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLBindParameter (param 4) failed for list handle\n"))
 goto Exit;

rc = SQLExecDirect (hstmt,
 (UCHAR *)"{call ifx_rc_insert(?, ?, ?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLExecDirect failed for list handle\n"))
 goto Exit;
}

/* Reset the statement parameters */
rc = SQLFreeStmt (hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 3 --
 SQLFreeStmt failed\n"))
 goto Exit;

fprintf (stdout, "STEP 3 done...row and list buffers populated\n");

```

```

/* STEP 4. Bind parameters for the row and list handles.
** Execute the insert statement to insert the new row into table
** 'customer'.
*/

rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hrow,
 sizeof(HINFX_RC), &cbHrow);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 1)\n"))
 goto Exit;

rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_COLLECTION, sizeof(HINFX_RC), 0, hlist,
 sizeof(HINFX_RC), &cbHlist);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLBindParameter failed (param 2)\n"))
 goto Exit;

rc = SQLExecDirect (hstmt, (UCHAR *)insertStmt, SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, "Error in Step 4 --
 SQLExecDirect failed\n"))
 goto Exit;

fprintf (stdout, "STEP 4 done...new row inserted into table
 'customer'\n");

/* STEP 5. Free the row and list handles.
*/

/* Free the row handle */
rc = SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_ROW, sizeof(HINFX_RC), 0, hrow, sizeof(HINFX_RC),
 &cbHrow);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

/* Free the list handle */
rc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY,
 SQL_INFX_RC_LIST, sizeof(HINFX_RC), 0, hlist, sizeof(HINFX_RC),
 &cbHlist);

rc = SQLExecDirect(hstmt, (UCHAR *)"{call ifx_rc_free(?)}", SQL_NTS);

fprintf (stdout, "STEP 5 done...row and list handles freed\n");

Exit:

/* CLEANUP: Close the statement handle.
** Free the statement handle.
** Disconnect from the datasource.
** Free the connection and environment handles.
** Exit.
*/

```

```

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

fprintf (stdout, "\n\nHit <Enter> to terminate the program...\n\n");
in = getchar ();
return (rc);

```

## Modify a row or collection

HCL OneDB™ ODBC Driver provides functions that can be used to modify rows and collections.

The following table provides an overview of the functions that HCL OneDB™ ODBC Driver provides for modifying rows and collections.

| Function         | Modification                      | Row | Collection                     |
|------------------|-----------------------------------|-----|--------------------------------|
| ifx_rc_delete()  | Delete an element                 | No  | Yes                            |
| ifx_rc_insert()  | Insert an element                 | No  | Yes (See the following table.) |
| ifx_rc_setnull() | Set the row or collection to null | Yes | Yes                            |
| ifx_rc_update()  | Update the value of an element    | Yes | Yes                            |

The following table describes the collection locations into which you can insert an element. You can insert an element only at the end of a SET or MULTISSET collection because elements in these types of collections do not have ordered positions.

|                 | Beginning | Middle | End |
|-----------------|-----------|--------|-----|
| <b>List</b>     | Yes       | Yes    | Yes |
| <b>Multiset</b> | No        | No     | Yes |
| <b>Set</b>      | No        | No     | Yes |



**Tip:** If you only need to insert or update a row or collection column with literal values, you do not need to use a row or collection buffer. Instead, you can explicitly list the literal value in either the INTO clause of the INSERT statement or the SET clause of the UPDATE statement.

Each row and collection maintains a seek position that points to the current element in the row or collection. When the row or collection is created, the seek position points to the first element that is in the row or collection. All calls to client functions



share the same seek position for a row or collection buffer. Therefore, one client function can affect the seek position for another client function that uses the same buffer handle. The following table describes how client functions use and modify the seek position.

| Client function | Acts on                       | Changes                                                               |
|-----------------|-------------------------------|-----------------------------------------------------------------------|
| ifx_rc_delete() | At the specified position     | Sets the seek position to the position after the one that was deleted |
| ifx_rc_fetch()  | At the specified position     | Sets the seek position to the specified position                      |
| ifx_rc_insert() | Before the specified position | Sets the seek position to the specified position                      |
| ifx_rc_update() | At the specified position     | Sets the seek position to the specified position                      |

## Retrieve information about a row or collection

HCL OneDB™ ODBC Driver provides functions that can be used to retrieve information about rows and collections.

The following table provides an overview of the functions that HCL OneDB™ ODBC Driver provides for retrieving information about rows and collections. The `ifx_rc_describe()` function returns the data types of elements in a row or collection.

| Function          | Information                             | Reference                                                  |
|-------------------|-----------------------------------------|------------------------------------------------------------|
| ifx_rc_count()    | Number of columns                       | <a href="#">The ifx_rc_count() function on page 183</a>    |
| ifx_rc_describe() | Data type information                   | <a href="#">The ifx_rc_describe() function on page 185</a> |
| ifx_rc_isnull()   | Value that indicates whether it is null | <a href="#">The ifx_rc_isnull() function on page 189</a>   |
| ifx_rc_typespec() | Type specification                      | <a href="#">The ifx_rc_typespec() function on page 190</a> |

## Client functions

These topics describe the HCL OneDB™ ODBC Driver client functions. Use these functions to access and manipulate smart large objects and rows and collections.

The information in these topics apply only if your database server is .

## Call a client function

This section describes the syntax of client functions, their input/output arguments, return values, and `SQL_BIGINT`.

## SQL syntax

This SQL syntax is for a client function.

```
{? = call client_function(?, ?,...)}
```

Use the first parameter marker ("?") only when the first parameter is an output parameter.

The following code example invokes a client function when the first parameter is an output parameter:

```
{? = call ifx_lo_open(?, ?, ?)}
```

The following code example invokes a client function when the first parameter is not an output parameter:

```
{call ifx_lo_create(?, ?, ?, ?)}
```

## Function syntax

The database server and the application both partially implement each client function.

You can execute a client function with either `SQLPrepare()` and `SQLExecute()` or with `SQLExecDirect()`. You need to call `SQLBindParameter()` or `SQLBindCol()` to bind each parameter before you call `SQLExecute()` or `SQLExecDirect()`.

## Executing a client function with `SQLPrepare()` and `SQLExecute()`

You can execute a client function with the `SQLPrepare()` and `SQLExecute()` functions.

### About this task

To execute a client function with `SQLPrepare()` and `SQLExecute()`:

1. Prepare the SQL statement for the client function.
2. Bind the parameters.
3. Execute the SQL statement.

### Results

The following code example illustrates these steps for `ifx_lo_open()`:

```
rc = SQLPrepare(hstmt, "{? = call ifx_lo_open(?, ?, ?)}", SQL_NTS);
rc = SQLBindParameter(...);
rc = SQLExecute(hstmt);
```

## Executing a client function with `SQLExecDirect()`

You can execute a client function with the `SQLExecDirect()` function.

### About this task

To execute a client function with `SQLExecDirect()`:

1. Bind the parameters.
2. Execute the SQL statement.

### Results

The following code example illustrates these steps for `ifx_lo_open()`:

```
rc = SQLBindParameter(...);
rc = SQLExecDirect(hstmt, "{? = call ifx_lo_open(?, ?, ?)}", SQL_NTS);
```

## Input and output parameters

Most of the input and output parameters for client functions are output parameters from the perspective of the client application.

However, a client function that accepts an input/output parameter initializes the parameter internally and sends it to the database server with the request to execute the client function. Therefore, you need to pass these parameters as input/output parameters to the driver.

## The SQL\_BIGINT data type

HCL OneDB™ supports the INT8 HCL OneDB™ SQL data type.

By default, the driver maps INT8 to the SQL\_BIGINT HCL OneDB™ ODBC Driver SQL data type and to the **SQL\_C\_CHAR** default HCL OneDB™ ODBC Driver C data type. However, client functions cannot access all the data type conversion functions.

Therefore, you must use a data type other than **SQL\_C\_CHAR** when you use a value of type SQL\_BIGINT.

For example, before you call `ifx_lo_specset_estbytes()`, you need to bind a variable for the `estbytes` input argument. Because `estbytes` is an SQL\_BIGINT, you would normally bind `estbytes` to an **SQL\_C\_CHAR**. However, **SQL\_C\_CHAR** does not work for SQL\_BIGINT for a client function. The following code example illustrates how to bind `estbytes` to an **SQL\_C\_LONG** instead of an **SQL\_C\_CHAR** for `ifx_lo_specset_estbytes()`:

```
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
 SQL_BIGINT, (UDWORD)0, 0, &estbytes, sizeof(estbytes), NULL);
rc = SQLExecDirect(hstmt, "{call ifx_lo_specset_estbytes(?, ?)}", SQL_NTS);
```

## Return codes

The client functions do not provide return codes.

For success or failure information, see the return codes for the HCL OneDB™ ODBC Driver function with which you call the client function (`SQLExecDirect()` or `SQLExecute()`).

## Functions for smart large objects

This section describes each client function that the driver provides for smart large objects.

The functions are listed alphabetically. For more information, see [Smart large objects on page 104](#).

### The `ifx_lo_alter()` function

The `ifx_lo_alter()` function alters the storage characteristics of a smart large object.

#### Syntax

```
ifx_lo_alter(loptr, lospec)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                                |
|---------------|--------------------|-------|--------------------------------------------|
| <i>loptr</i>  | SQL_INFX_UDT_FIXED | Input | Smart-large-object pointer structure       |
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure |

## Usage

The `ifx_lo_alter()` function performs the following steps to update the storage characteristics of a smart large object:

1. Gets an exclusive lock for the smart large object.
2. Uses the characteristics that are in the *lospec* smart-large-object specification structure to update the storage characteristics of the smart large object. The `ifx_lo_alter()` function lets you change the following storage characteristics:
  - Logging characteristics
  - Last-access time characteristics
  - Extent size
3. Unlocks the smart large object.

As an alternative to calling this function, you can call one of the following functions if you want to change only one of these characteristics:

- `ifx_lo_specset_flags()`
- `ifx_lo_specset_extsz()`

## The `ifx_lo_close()` function

The `ifx_lo_close()` function closes a smart large object.

### Syntax

```
ifx_lo_close(lofd)
```


## Arguments

The function accepts the following argument.

| Argument    | Type        | Use   | Description                        |
|-------------|-------------|-------|------------------------------------|
| <i>lofd</i> | SQL_INTEGER | Input | Smart-large-object file descriptor |

## Usage

The `ifx_lo_close()` function closes a smart large object. During this function, the database server tries to unlock the smart large object. If the isolation mode is repeatable read or if the lock is an exclusive lock, the database server does not release the lock until the end of the transaction.

 **Tip:** If you do not update a smart large object inside a `BEGIN WORK` transaction block, each update is a separate transaction.

## The `ifx_lo_col_info()` function

The `ifx_lo_col_info()` function updates a smart-large-object specification structure with column-level storage characteristics.

### Syntax

```
ifx_lo_col_info(colname, lospec)
```


### Arguments

The function accepts the following arguments.

| Argument       | Type               | Use   | Description                                                                                                                                                                                                                                                                                                                                                     |
|----------------|--------------------|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>colname</i> | SQL_CHAR           | Input | <p>Pointer to a buffer that contains the name of a database column</p> <p>This value must be in the following format:</p> <pre>database@server_name:table.column</pre> <p>If the column is in a database that is ANSI-compliant, you can include the owner name. In this case, use the following format:</p> <pre>database@server_name:owner.table.column</pre> |
| <i>lospec</i>  | SQL_INFX_UDT_FIXED | I/O   | Smart-large-object specification structure                                                                                                                                                                                                                                                                                                                      |

### Usage

The `ifx_lo_col_info()` function sets the fields for a smart-large-object specification structure to the storage characteristics for the *colname* database column. If the specified column does not have column-level storage characteristics defined, the database server uses the storage characteristics that are inherited.

 **Important:** You must call `ifx_lo_def_create_spec()` before you call this function.

## The `ifx_lo_create()` function

The `ifx_lo_create()` function creates and opens a new smart large object.

## Syntax

```
ifx_lo_create(lospec, flags, loptr, lofd)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use    | Description                                                                                                     |
|---------------|--------------------|--------|-----------------------------------------------------------------------------------------------------------------|
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Input  | Smart-large-object specification structure that contains storage characteristics for the new smart large object |
| <i>flags</i>  | SQL_INTEGER        | Input  | Mode in which to open the new smart large object.                                                               |
| <i>loptr</i>  | SQL_INFX_UDT_FIXED | I/O    | Smart-large-object pointer structure                                                                            |
| <i>lofd</i>   | SQL_INTEGER        | Output | Smart-large-object file descriptor. This file descriptor is only valid within the current database connection.  |

## Usage

The `ifx_lo_create()` function performs the following steps to create and open a new smart large object:

1. Creates a smart-large-object pointer structure.
2. Assigns a pointer to this structure and returns this pointer in *loptr*.
3. Assigns storage characteristics for the smart large object from the smart-large-object specification structure that *lospec* indicates.

If *lospec* is null, `ifx_lo_create()` uses the system-specified storage characteristics. If the smart-large-object specification structure exists but does not contain storage characteristics, `ifx_lo_create()` uses the storage characteristics from the inheritance hierarchy.

4. Opens the smart large object in the access mode that *flags* specifies.
5. Associates the smart large object with the current connection.

When you close this connection, the database server deallocates any associated smart large objects that have a reference count of zero. The reference count indicates the number of database columns that refer to the smart large object.

6. Returns a file descriptor that identifies the smart large object.

The database server uses the default parameters that the call to `ifx_lo_create()` establishes to determine whether to lock or log subsequent operations on the smart large object.

## The `ifx_lo_def_create_spec()` function

The `ifx_lo_def_create_spec()` function creates a smart-large-object specification structure.

## Syntax

```
ifx_lo_def_create_spec(lospec)
```

## Arguments

The function accepts the following argument.

| Argument      | Type               | Use | Description                                |
|---------------|--------------------|-----|--------------------------------------------|
| <i>lospec</i> | SQL_INFX_UDT_FIXED | I/O | Smart-large-object specification structure |

## Usage

The `ifx_lo_def_create_spec()` function creates a smart-large-object specification structure and initializes the fields to null values. If you do not change these values, the null values tell the database server to use the system-specified defaults for the storage characteristics of the smart large object.

## The `ifx_lo_open()` function

The `ifx_lo_open()` function opens a smart large object.

## Syntax

```
ifx_lo_open(lofd, loptr, flags)
```

## Arguments

The function accepts the following arguments.

| Argument     | Type               | Use    | Description                                                                                                    |
|--------------|--------------------|--------|----------------------------------------------------------------------------------------------------------------|
| <i>lofd</i>  | SQL_INTEGER        | Output | Smart-large-object file descriptor. This file descriptor is only valid within the current database connection. |
| <i>loptr</i> | SQL_INFX_UDT_FIXED | Input  | Smart-large-object pointer structure                                                                           |
| <i>flags</i> | SQL_INTEGER        | Input  | Mode in which to open the smart large object.                                                                  |

## Usage

The `ifx_lo_open()` function performs the following steps to open a smart large object:

1. Opens the *loptr* smart large object in the access mode that *flags* specifies.
2. Sets the seek position to byte zero.
3. Locks the smart large object.



**Important:** The database server does not check access permissions on the smart large object. Your application must make sure that the user or application is trusted.

As the following table describes, the access mode determines the type of lock.

| Access mode                             | Type of lock                                                                                                                                                                          |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Dirty read                              | No lock                                                                                                                                                                               |
| Read only                               | Shared lock                                                                                                                                                                           |
| Write only, write/append, or read/write | Update lock. When you call <code>ifx_lo_write()</code> or <code>ifx_lo_writewithseek()</code> for the smart large object, the database server promotes the lock to an exclusive lock. |

The database server loses this lock when the current transaction terminates. The database server obtains the lock again the next time you call a function that needs a lock.

As an alternative, you can use a `BEGIN WORK` transaction block and place a `COMMIT WORK` or `ROLLBACK WORK` statement after the last statement that needs to use the lock.

1. Associates the smart large object with the current connection.

When you close this connection, the database server deallocates any associated smart large objects that have a reference count of zero. The reference count indicates the number of database columns that refer to the smart large object.

2. Returns a file descriptor that identifies the smart large object.

The database server uses the default parameters that the call to `ifx_lo_open()` establishes to determine whether to lock or log subsequent operations on the smart large object.

## The `ifx_lo_read()` function

The `ifx_lo_read()` function reads data from an open smart large object.

### Syntax

```
ifx_lo_read(lofd, buf)
```

### Arguments

The function accepts the following arguments.

| Argument          | Type        | Use    | Description                                                              |
|-------------------|-------------|--------|--------------------------------------------------------------------------|
| <code>lofd</code> | SQL_INTEGER | Input  | Smart-large-object file descriptor                                       |
| <code>buf</code>  | SQL_CHAR    | Output | Pointer to a character buffer into which the function will read the data |



## Usage

The `ifx_lo_read()` function reads data from an open smart large object. The read begins at the current seek position for `lofd`. You can call `ifx_lo_tell()` to obtain the current seek position.

The `ifx_lo_read()` function reads `cbValueMax` bytes of data. `cbValueMax` is an input argument for `SQLBindParameter()` and `SQLBindCol()`. The size of `buf` or `cbValueMax` cannot exceed 2 gigabytes. To read a smart large object that is larger than 2 gigabytes, read it in 2-gigabyte chunks. The `ifx_lo_read()` function reads this data into the user-defined buffer to which `buf` points.

If `SQLBindParameter()` or `SQLBindCol()` returns `SQL_SUCCESS`, then `pcbValue`, which is an argument for each of these functions, contains the number of bytes that the function read from the smart large object. If `SQLBindParameter()` or `SQLBindCol()` returns `SQL_SUCCESS_WITH_INFO`, then `pcbValue` contains the number of bytes that are available to read from the smart large object.

## The `ifx_lo_readwithseek()` function

The `ifx_lo_readwithseek()` function performs a seek operation and then reads data from an open smart large object.

## Syntax

```
ifx_lo_readwithseek(lofd, buf, offset, whence)
```

## Arguments

The function accepts the following arguments.

| Argument            | Type                     | Use    | Description                                                                                                                                                                                                                                           |
|---------------------|--------------------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>lofd</code>   | <code>SQL_INTEGER</code> | Input  | Smart-large-object file descriptor                                                                                                                                                                                                                    |
| <code>buf</code>    | <code>SQL_CHAR</code>    | Output | Pointer to a character buffer into which the function will read the data                                                                                                                                                                              |
| <code>offset</code> | <code>SQL_BIGINT</code>  | Input  | Offset from the starting seek position, in bytes. Instead of using the default HCL OneDB™ ODBC Driver C data type of <code>SQL_C_CHAR</code> for <code>offset</code> , use <code>SQL_C_LONG</code> or <code>SQL_C_SHORT</code> .                      |
| <code>whence</code> | <code>SQL_INTEGER</code> | Input  | Starting seek position. The possible values are:<br><b>LO_SEEK_CUR</b><br>The current seek position in the smart large object<br><b>LO_SEEK_END</b><br>The end of the smart large object<br><b>LO_SEEK_SET</b><br>The start of the smart large object |

## Usage

The `ifx_lo_readwithseek()` function performs a seek operation and then reads data from an open smart large object. The read begins at the seek position of `lofd` that the `offset` and `whence` arguments indicate.

The `ifx_lo_readwithseek()` function reads `cbValueMax` bytes of data. `cbValueMax` is an input argument for `SQLBindParameter()` and `SQLBindCol()`. The size of `buf` or `cbValueMax` cannot exceed 2 GB. To read a smart large object that is larger than 2 gigabytes, read it in 2-GB chunks. The `ifx_lo_readwithseek()` function reads this data into the user-defined buffer to which `buf` points.

If `SQLBindParameter()` or `SQLBindCol()` returns `SQL_SUCCESS`, then `pcbValue`, which is an argument for each of these functions, contains the number of bytes that the function read from the smart large object. If `SQLBindParameter()` or `SQLBindCol()` returns `SQL_SUCCESS_WITH_INFO`, then `pcbValue` contains the number of bytes that are available to read from the smart large object.

## The `ifx_lo_seek()` function

The `ifx_lo_seek()` function sets the file position for the next read or write operation on an open smart large object.

## Syntax

```
ifx_lo_seek(lofd, offset, whence, seek_pos)
```

## Arguments

The function accepts the following arguments.

| Argument              | Type                     | Use   | Description                                                                                                                                                                                                                                                       |
|-----------------------|--------------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>lofd</code>     | <code>SQL_INTEGER</code> | Input | Smart-large-object file descriptor                                                                                                                                                                                                                                |
| <code>offset</code>   | <code>SQL_BIGINT</code>  | Input | Offset from the starting seek position, in bytes. Instead of using the default HCL OneDB™ ODBC Driver C data type of <code>SQL_C_CHAR</code> for <code>offset</code> , use <code>SQL_C_LONG</code> or <code>SQL_C_SHORT</code> .                                  |
| <code>whence</code>   | <code>SQL_INTEGER</code> | Input | Starting seek position. The possible values are:<br><br><b>LO_SEEK_CUR</b><br>The current seek position in the smart large object<br><br><b>LO_SEEK_END</b><br>The end of the smart large object<br><br><b>LO_SEEK_SET</b><br>The start of the smart large object |
| <code>seek_pos</code> | <code>SQL_BIGINT</code>  | I/O   | New seek position. Instead of using the default HCL OneDB™ ODBC Driver C data type of <code>SQL_C_CHAR</code> for <code>seek_pos</code> , use                                                                                                                     |

| Argument | Type | Use | Description                                                                                          |
|----------|------|-----|------------------------------------------------------------------------------------------------------|
|          |      |     | <b>SQL_C_LONG</b> . For more information, see <a href="#">The SQL_BIGINT data type on page 163</a> . |

## Usage

The `ifx_lo_seek()` function sets the seek position of *lofd* to the position that the *offset* and *whence* arguments indicate.

## The `ifx_lo_specget_estbytes()` function

The `ifx_lo_specget_estbytes()` function gets the estimated number of bytes from a smart-large-object specification structure.

## Syntax

```
ifx_lo_specget_estbytes(lospec, estbytes)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type               | Use    | Description                                                                                                                                                                                                                                                              |
|-----------------|--------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lospec</i>   | SQL_INFX_UDT_FIXED | Input  | Smart-large-object specification structure                                                                                                                                                                                                                               |
| <i>estbytes</i> | SQL_BIGINT         | Output | Estimated final size of the smart large object, in bytes. This estimate is an optimization hint for the smart-large-object optimizer. Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>estbytes</i> , use <b>SQL_C_LONG</b> . |

## Usage

The `ifx_lo_specget_estbytes()` function gets the estimated number of bytes from a smart-large-object specification structure.

## The `ifx_lo_specget_extsz()` function

The `ifx_lo_specget_extsz()` function gets the allocation extent from a smart-large-object specification structure.

## Syntax

```
ifx_lo_specget_extsz(lospec, extsz)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                                |
|---------------|--------------------|-------|--------------------------------------------|
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure |

| Argument     | Type        | Use    | Description                                                                                                                                                                                                                                                                                                                |
|--------------|-------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>extsz</i> | SQL_INTEGER | Output | Extent size of the smart large object, in bytes. This value is the size of the allocation extents to be allocated for the smart large object when the database server writes beyond the end of the current extent. This value overrides the estimate that the database server generates for how large an extent should be. |

### Usage

The `ifx_lo_specget_extsz()` function gets the allocation extent from a smart-large-object specification structure.

### The `ifx_lo_specget_flags()` function

The `ifx_lo_specget_flags()` function gets the create-time flags from a smart-large-object specification structure.

### Syntax

```
ifx_lo_specget_flags(lospec, flags)
```

### Arguments

The function accepts the following arguments.

| Argument      | Type               | Use    | Description                                                                             |
|---------------|--------------------|--------|-----------------------------------------------------------------------------------------|
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Input  | Smart-large-object specification structure                                              |
| <i>flags</i>  | SQL_INTEGER        | Output | Create-time flags. For more information, see <a href="#">Access modes on page 121</a> . |

### Usage

The `ifx_lo_specget_flags()` function gets the create-time flags from a smart-large-object specification structure.

### The `ifx_lo_specget_maxbytes()` function

The `ifx_lo_specget_maxbytes()` function gets the maximum number of bytes from a smart-large-object specification structure.

### Syntax

```
ifx_lo_specget_maxbytes(lospec, maxbytes)
```

### Arguments

The function accepts the following arguments.

| Argument        | Type               | Use   | Description                                                                                                                                                                           |
|-----------------|--------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lospec</i>   | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure                                                                                                                                            |
| <i>maxbytes</i> | SQL_BIGINT         | Input | Maximum size, in bytes, of the smart large object. Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>maxbytes</i> , use <b>SQL_C_LONG</b> . |

## Usage

The `ifx_lo_specget_maxbytes()` function gets the maximum number of bytes from a smart-large-object specification structure.

## The `ifx_lo_specget_sbspace()` function

The `ifx_lo_specget_sbspace()` function gets the sbspace name from a smart-large-object specification structure.

## Syntax

```
ifx_lo_specget_sbspace(lospec, sbspace)
```

## Arguments

The function accepts the following arguments.

| Argument       | Type               | Use    | Description                                                                                                                  |
|----------------|--------------------|--------|------------------------------------------------------------------------------------------------------------------------------|
| <i>lospec</i>  | SQL_INFX_UDT_FIXED | Input  | Smart-large-object specification structure                                                                                   |
| <i>sbspace</i> | SQL_CHAR           | Output | Name of the sbspace for the smart large object. An sbspace name can be up to 18 characters long and must be null terminated. |

## Usage

The `ifx_lo_specget_sbspace()` function returns the name of the sbspace in which to store the smart large object. The function copies up to  $(pcbValue-1)$  bytes into the *sbspace* buffer and makes sure that it is null terminated. *pcbValue* is an argument for `SQLBindParameter()` and `SQLBindCol()`.

## The `ifx_lo_specset_estbytes()` function

The `ifx_lo_specset_estbytes()` function sets the estimated number of bytes in a smart-large-object specification structure.

## Syntax

```
ifx_lo_specset_estbytes(lospec, estbytes)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type               | Use   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|--------------------|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lospec</i>   | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>estbytes</i> | SQL_BIGINT         | Input | <p>Estimated final size of the smart large object, in bytes. This estimate is an optimization hint for the smart large object optimizer. This value cannot exceed 2 gigabytes.</p> <p>If you do not specify an <i>estbytes</i> value when you create a new smart large object, the database server gets the value from the inheritance hierarchy of storage characteristics.</p> <p>Do not change this system value unless you know the estimated size for the smart large object. If you do set the estimated size for a smart large object, do not specify a value much higher than the final size of the smart large object. Otherwise, the database server might allocate unused storage.</p> <p>Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>estbytes</i>, use <b>SQL_C_LONG</b> or <b>SQL_C_SHORT</b>.</p> |

## Usage

The `ifx_lo_specset_estbytes()` function sets the estimated number of bytes in a smart-large-object specification structure.

## The `ifx_lo_specset_extsz()` function

The `ifx_lo_specset_extsz()` function sets the allocation extent size in a smart-large-object specification structure.

## Syntax

```
ifx_lo_specset_extsz(lospec, extsz)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                                |
|---------------|--------------------|-------|--------------------------------------------|
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure |

| Argument     | Type        | Use   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------|-------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>extsz</i> | SQL_INTEGER | Input | <p>Extent size of the smart large object, in bytes. This value specifies the size of the allocation extents to be allocated for the smart large object when the database server writes beyond the end of the current extent. This value overrides the estimate that the database server generates for how large an extent should be.</p> <p>If you do not specify an <i>extsz</i> value when you create a new smart large object, the database server attempts to optimize the extent size based on past operations on the smart large object and other storage characteristics (such as maximum bytes) that it obtains from the inheritance hierarchy of storage characteristics.</p> <p>Do not change this system value unless you know the allocation extent size for the smart large object. Only applications that encounter severe storage fragmentation should ever set the allocation extent size. For such applications, make sure that you know exactly the number of bytes by which to extend the smart large object.</p> |

## Usage

The `ifx_lo_specset_extsz()` function sets the allocation extent size in a smart-large-object specification structure.

## The `ifx_lo_specset_flags()` function

The `ifx_lo_specset_flags()` function sets the create-time flags in a smart-large-object specification structure.

## Syntax

```
ifx_lo_specset_flags(lospec, flags)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                                |
|---------------|--------------------|-------|--------------------------------------------|
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure |
| <i>flags</i>  | SQL_INTEGER        | Input | Create-time flags.                         |

## Usage

The `ifx_lo_specset_flags()` function sets the create-time flags in a smart-large-object specification structure.

## The `ifx_lo_specset_maxbytes()` function

The `ifx_lo_specset_maxbytes()` function sets the maximum number of bytes in a smart-large-object specification structure.

## Syntax

```
ifx_lo_specset_maxbytes(lospec, maxbytes)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type               | Use   | Description                                                                                                                                                                                                                                      |
|-----------------|--------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lospec</i>   | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure                                                                                                                                                                                                       |
| <i>maxbytes</i> | SQL_BIGINT         | Input | Maximum size of the smart large object, in bytes. This value cannot exceed 2 gigabytes. Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>maxbytes</i> , use <b>SQL_C_LONG</b> or <b>SQL_C_SHORT</b> . |

## Usage

The `ifx_lo_specset_maxbytes()` function sets the maximum number of bytes in a smart-large-object specification structure.

## The `ifx_lo_specset_sbspace()` function

The `ifx_lo_specset_sbspace()` function sets the sbspace name in a smart-large-object specification structure.

## Syntax

```
ifx_lo_specset_sbspace(lospec, sbspace)
```

## Arguments

The function accepts the following arguments.

| Argument       | Type               | Use   | Description                                                                                                                                                                                                                                            |
|----------------|--------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lospec</i>  | SQL_INFX_UDT_FIXED | Input | Smart-large-object specification structure                                                                                                                                                                                                             |
| <i>sbspace</i> | SQL_CHAR           | Input | Name of the sbspace for the smart large object. An sbspace name can be up to 18 characters long and must be null terminated. If you do not specify an <i>sbspace</i> when you create a new smart large object, the database server obtains the sbspace |



| Argument | Type | Use | Description                                                                                                  |
|----------|------|-----|--------------------------------------------------------------------------------------------------------------|
|          |      |     | name from either the column information or from the SBSPACENAME parameter of the <code>onconfig</code> file. |

## Usage

The `ifx_lo_specset_sbospace()` function uses *pcbValue* to determine the length of the sbospace name. *pcbValue* is an argument for `SQLBindParameter()` and `SQLBindCol()`.

## The `ifx_lo_stat()` function

The `ifx_lo_stat()` function initializes a smart-large-object status structure.

## Syntax

```
ifx_lo_stat(lofd, lostat)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                         |
|---------------|--------------------|-------|-------------------------------------|
| <i>lofd</i>   | SQL_INTEGER        | Input | Smart-large-object file descriptor  |
| <i>lostat</i> | SQL_INFX_UDT_FIXED | I/O   | Smart-large-object status structure |

## Usage

Before you call `ifx_lo_stat()`, call `SQLGetInfo()` to get the size of the smart-large-object status structure. Use this size to allocate memory for the structure.

The `ifx_lo_stat()` function allocates a smart-large-object status structure and initializes it with the status information for the smart large object.

## The `ifx_lo_stat_atime()` function

The `ifx_lo_stat_atime()` function retrieves the last access time for a smart large object.

## Syntax

```
ifx_lo_stat_atime(lostat, atime)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                         |
|---------------|--------------------|-------|-------------------------------------|
| <i>lostat</i> | SQL_INFX_UDT_FIXED | Input | Smart-large-object status structure |

| Argument     | Type        | Use    | Description                                                                                                                                                                                     |
|--------------|-------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>atime</i> | SQL_INTEGER | Output | Time of the last access for a smart large object, in seconds. The database server maintains the time of last access only if the LO_KEEP_LASTACCESS_TIME flag is set for the smart large object. |

## Usage

The `ifx_lo_stat_atime()` function retrieves the last access time for a smart large object.

## The `ifx_lo_stat_cspec()` function

The `ifx_lo_stat_cspec()` function retrieves a smart-large-object specification structure.

## Syntax

```
ifx_lo_stat_cspec(lostat, lospec)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use    | Description                                |
|---------------|--------------------|--------|--------------------------------------------|
| <i>lostat</i> | SQL_INFX_UDT_FIXED | Input  | Smart-large-object status structure        |
| <i>lospec</i> | SQL_INFX_UDT_FIXED | Output | Smart-large-object specification structure |

## Usage

The `ifx_lo_stat_cspec()` function retrieves a smart-large-object specification structure and returns a pointer to the structure.

## The `ifx_lo_stat_ctime()` function

The `ifx_lo_stat_ctime()` function retrieves the time of the last change of a smart large object.

## Syntax

```
ifx_lo_stat_ctime(lostat, ctime)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use   | Description                                                                                       |
|---------------|--------------------|-------|---------------------------------------------------------------------------------------------------|
| <i>lostat</i> | SQL_INFX_UDT_FIXED | Input | Smart-large-object status structure                                                               |
| <i>ctime</i>  | SQL_INTEGER        | Out   | Time of the last change of the smart large object, in seconds. The time of the last status change |

| Argument | Type | Use | Description                                                                                                                            |
|----------|------|-----|----------------------------------------------------------------------------------------------------------------------------------------|
|          |      |     | includes modification of storage characteristics, including a change in the number of references and writes to the smart large object. |

## Usage

The `ifx_lo_stat_ctime()` function retrieves the time of the last change of a smart large object.

## The `ifx_lo_stat_refcnt()` function

The `ifx_lo_stat_refcnt()` function retrieves the number of references to a smart large object.

## Syntax

```
ifx_lo_stat_refcnt(lostat, refcount)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type               | Use    | Description                                                                                                                      |
|-----------------|--------------------|--------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>lostat</i>   | SQL_INFX_UDT_FIXED | Input  | Smart-large-object status structure                                                                                              |
| <i>refcount</i> | SQL_INTEGER        | Output | Number of references to a smart large object. This value is the number of database columns that refer to the smart large object. |

## Usage

The `ifx_lo_stat_refcnt()` function retrieves the number of references to a smart large object.

A database server can remove a smart large object and reuse any resources that are allocated to it when the reference count for the smart large object is zero and one of the following events occurs:

- The transaction in which the reference count is decremented to zero commits.
- The connection during which the smart large object was created terminates, but the reference count is not incremented.

The database server increments a reference counter when it stores the smart-large-object pointer structure for a smart large object in a row.

## The `ifx_lo_stat_size()` function

The `ifx_lo_stat_size()` function retrieves the size of a smart large object.

## Syntax

```
ifx_lo_stat_size(lostat, size)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type               | Use    | Description                                                                                                                                                                                                  |
|---------------|--------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lostat</i> | SQL_INFX_UDT_FIXED | Input  | Smart-large-object status structure                                                                                                                                                                          |
| <i>size</i>   | SQL_BIGINT         | Output | Size of a smart large object, in bytes. This value cannot exceed 2 gigabytes. Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>size</i> , use <b>SQL_C_LONG</b> . |

## Usage

The `ifx_lo_stat_size()` function retrieves the size of a smart large object.

## The `ifx_lo_tell()` function

The `ifx_lo_tell()` function retrieves the current file or seek position for an open smart large object.

## Syntax

```
ifx_lo_tell(lofd, seek_pos)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type        | Use   | Description                                                                                                                                                                                                                               |
|-----------------|-------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lofd</i>     | SQL_INTEGER | Input | Smart-large-object file descriptor                                                                                                                                                                                                        |
| <i>seek_pos</i> | SQL_BIGINT  | I/O   | New seek position, which is the offset for the next read or write operation on the smart large object. Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>seek_pos</i> , use <b>SQL_C_LONG</b> . |

## Usage

The `ifx_lo_tell()` function retrieves the current file or seek position for an open smart large object.

This function works correctly for smart large objects up to 2 gigabytes in size.

## The `ifx_lo_truncate()` function

The `ifx_lo_truncate()` function truncates a smart large object at the specified position.

## Syntax

```
ifx_lo_truncate(lofd, offset)
```

## Arguments

The function accepts the following arguments.

| Argument      | Type        | Use   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|-------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lofd</i>   | SQL_INTEGER | Input | Smart-large-object file descriptor                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <i>offset</i> | SQL_BIGINT  | Input | End of the smart large object. If this value exceeds the end of the smart large object, the function extends the smart large object. If this value is less than the end of the smart large object, the database server reclaims all storage from the offset position to the end of the smart large object.<br><br>Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>offset</i> , use <b>SQL_C_LONG</b> or <b>SQL_C_SHORT</b> . |

## Usage

The `ifx_lo_truncate()` function sets the end of a smart large object to the location that the *offset* argument specifies.

## The `ifx_lo_write()` function

The `ifx_lo_write()` function writes data to an open smart large object.

## Syntax

```
ifx_lo_write(lofd, buf)
```

## Arguments

The function accepts the following arguments.

| Argument    | Type        | Use   | Description                                                                                                                         |
|-------------|-------------|-------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>lofd</i> | SQL_INTEGER | Input | Smart-large-object file descriptor                                                                                                  |
| <i>buf</i>  | SQL_CHAR    | Input | Buffer that contains the data that the function writes to the smart large object. The size of the buffer cannot exceed 2 gigabytes. |

## Usage

The `ifx_lo_write()` function writes data to an open smart large object. The write begins at the current seek position for *lofd*. You can call `ifx_lo_tell()` to obtain the current seek position.

The `ifx_lo_write()` function writes *cbValueMax* bytes of data. *cbValueMax* is an input argument for `SQLBindParameter()` and `SQLBindCol()`. The size of *buf* or *cbValueMax* cannot exceed 2 GB. To write to a smart large object that is larger than 2 gigabytes, write to it in 2-GB chunks. The `ifx_lo_write()` function gets the data from the user-defined buffer to which *buf* points.

If `SQLExecDirect()` or `SQLExecute()` returns `SQL_SUCCESS_WITH_INFO`, then the database server wrote less than *cbValueMax* bytes of data to the smart large object and *pcbValue*, which is an argument for each of these functions, contains the number of bytes that the function wrote. This condition can occur when the sbspace runs out of space.

## The `ifx_lo_writewithseek()` function

The `ifx_lo_writewithseek()` function performs a seek operation and then writes data to an open smart large object.

### Syntax

```
ifx_lo_writewithseek(lofd, buf, offset, whence)
```

### Arguments

The function accepts the following arguments.

| Argument      | Type        | Use   | Description                                                                                                                                                                                                                                                       |
|---------------|-------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>lofd</i>   | SQL_INTEGER | Input | Smart-large-object file descriptor.                                                                                                                                                                                                                               |
| <i>buf</i>    | SQL_CHAR    | Input | Buffer that contains the data that the function writes to the smart large object. The size of the buffer must not exceed 2 gigabytes.                                                                                                                             |
| <i>offset</i> | SQL_BIGINT  | Input | Offset from the starting seek position, in bytes. Instead of using the default HCL OneDB™ ODBC Driver C data type of <b>SQL_C_CHAR</b> for <i>offset</i> , use <b>SQL_C_LONG</b> or <b>SQL_C_SHORT</b> .                                                          |
| <i>whence</i> | SQL_INTEGER | Input | Starting seek position. The possible values are:<br><br><b>LO_SEEK_CUR</b><br>The current seek position in the smart large object<br><br><b>LO_SEEK_END</b><br>The end of the smart large object<br><br><b>LO_SEEK_SET</b><br>The start of the smart large object |

### Usage

The `ifx_lo_writewithseek()` function performs a seek operation and then writes data to an open smart large object. The write begins at the seek position of *lofd* that the *offset* and *whence* arguments indicate.

The `ifx_lo_writewithseek()` function writes *cbValueMax* bytes of data. *cbValueMax* is an input argument for `SQLBindParameter()` and `SQLBindCol()`. The size of *buf* or *cbValueMax* cannot exceed 2 GB. To write to a smart large object that is larger than 2 gigabytes, write to it in 2-GB chunks. The `ifx_lo_writewithseek()` function gets the data from the user-defined buffer to which *buf* points.

If `SQLExecDirect()` or `SQLExecute()` returns `SQL_SUCCESS_WITH_INFO`, then the database server wrote less than *cbValueMax* bytes of data to the smart large object and *pcbValue*, which is an argument for each of these functions, contains the number of bytes that the function wrote. This condition can occur when the sbspace runs out of space.

## Functions for rows and collections

This section describes each client function that HCL OneDB™ ODBC Driver provides for rows and collections.

The functions are listed alphabetically. For more information about rows and collections, see [Rows and collections on page 143](#).

### The `ifx_rc_count()` function

The `ifx_rc_count()` function returns the number of elements or fields that are in a row or collection.

#### Syntax

```
ifx_rc_count(rowcount, rchandle)
```

#### Arguments

The function accepts the following arguments.

| Argument        | Type         | Use    | Description                                                    |
|-----------------|--------------|--------|----------------------------------------------------------------|
| <i>rowcount</i> | SQL_SMALLINT | Output | Number of elements or fields that are in the row or collection |
| <i>rchandle</i> | HINFX_RC     | Input  | Handle for a row or collection buffer                          |

#### Usage

The `ifx_rc_count()` function returns the number of elements or fields that are in the row or collection.

### The `ifx_rc_create()` function

The `ifx_rc_create()` function creates a buffer for a row or collection.

#### Syntax

```
ifx_rc_create(rchandle, typespec)
```

#### Arguments

The function accepts the following arguments.

| Argument        | Type     | Use    | Description                                                 |
|-----------------|----------|--------|-------------------------------------------------------------|
| <i>rchandle</i> | HINFX_RC | Output | Handle for a row or collection buffer                       |
| <i>typespec</i> | SQL_CHAR | Input  | Type specification for the buffer. See the following table. |

The following table describes the syntax for the *typespec* argument.

| Type of buffer          | Syntax                                                                                                                                                                                                                                                                                                                                     | Example                                                                  |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| Unfixed-type collection | COLLECTION                                                                                                                                                                                                                                                                                                                                 | COLLECTION                                                               |
| Fixed-type collection   | <p>COLLECTION {SET   MULTISET   LIST} (<i>type</i> not null)</p> <p>or</p> <p>{SET   MULTISET   LIST (<i>type</i> not null)</p> <p>where <i>type</i> is the HCL OneDB™ SQL data type for the elements in the collection</p>                                                                                                                | <p>COLLECTION SET (int not null)</p> <p>or</p> <p>SET (int not null)</p> |
| Unfixed-type row        | ROW                                                                                                                                                                                                                                                                                                                                        | ROW                                                                      |
| Fixed-type row          | <p>ROW ["<i>name</i>"] (<i>field_id</i> <i>type</i> [, <i>field_id</i> <i>type</i>, ...])</p> <p>where:</p> <ul style="list-style-type: none"> <li>• <i>name</i> is an optional name for the entire row</li> <li>• <i>field_id</i> is the name for a field</li> <li>• <i>type</i> is the HCL OneDB™ SQL data type for the field</li> </ul> | <p>ROW "employee_t" (name char(255), id_num int, dept int)</p>           |

### Usage

The *ifx\_rc\_create()* function allocates memory for a row or collection buffer and returns a handle to the buffer. The following table describes how the function initializes the buffer.

| Type of buffer          | Initial value for the row or collection | Initial value for the contents of the row or collection |
|-------------------------|-----------------------------------------|---------------------------------------------------------|
| Fixed-type collection   | Non-null                                | Empty                                                   |
| Fixed-type row          | Non-null                                | Each value is null                                      |
| Unfixed-type collection | Null                                    | Empty                                                   |
| Unfixed-type row        | Null                                    | Empty                                                   |



For a row, the function sets the seek position to element number one. An empty collection buffer does not have a seek position.

## The ifx\_rc\_delete() function

The ifx\_rc\_delete() function deletes an element from a collection.

### Syntax

```
ifx_rc_delete(rchandle, action, jump)
```

### Arguments

The function accepts the following arguments.

| Argument        | Type         | Use   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------|--------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rchandle</i> | HINFX_RC     | Input | Handle for a collection buffer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>action</i>   | SQL_SMALLINT | Input | Location of the element relative to the seek position. The possible values are <ul style="list-style-type: none"> <li>• SQL_INFX_RC_ABSOLUTE: Element number <i>jump</i> where the first element in the buffer is element number one</li> <li>• SQL_INFX_RC_CURRENT: Current® element</li> <li>• SQL_INFX_RC_FIRST: First element</li> <li>• SQL_INFX_RC_LAST: Last element</li> <li>• SQL_INFX_RC_NEXT: Next element</li> <li>• SQL_INFX_RC_PRIOR: Previous element</li> <li>• SQL_INFX_RC_RELATIVE: Element that is <i>jump</i> elements past the current element</li> </ul> |
| <i>jump</i>     | SQL_SMALLINT | Input | Offset when <i>action</i> is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

### Usage

The ifx\_rc\_delete() function deletes an element from a collection from the location that is specified by *action* and *jump*. The function sets the seek position to the position of the value that was deleted. It is not possible to delete an element from a row.

## The ifx\_rc\_describe() function

The ifx\_rc\_describe() function returns descriptive information about the data type for a row or collection or for an element that is in a row or collection.

## Syntax

```
ifx_rc_describe(rchandle, fieldnum, fieldname, typecode,
columnsize, decdigits, nullable, typename, typeowner)
```

## Arguments

The function accepts the following arguments.

| Argument          | Type         | Use    | Description                                                                                                                                                                                                                                                                                                                  |
|-------------------|--------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rchandle</i>   | HINFX_RC     | Input  | Handle for a row or collection buffer                                                                                                                                                                                                                                                                                        |
| <i>fieldnum</i>   | SQL_SMALLINT | Input  | Field number. If this value is 0, the function returns information for the entire row or collection. For a collection, any value other than 0 causes the function to return information for the elements that are in the collection. For a row, this value specifies the element for which the function returns information. |
| <i>fieldname</i>  | SQL_CHAR     | Output | Field name. The function returns this value only for an element that is in a row.                                                                                                                                                                                                                                            |
| <i>typecode</i>   | SQL_SMALLINT | Output | HCL OneDB™ ODBC Driver SQL data type of the element                                                                                                                                                                                                                                                                          |
| <i>columnsize</i> | SQL_INTEGER  | Output | Column size. For a character element, this value is the size of the column, in bytes. For a numeric element, this value is the precision. For other data types, the function does not return this value.                                                                                                                     |
| <i>decdigits</i>  | SQL_SMALLINT | Output | Decimal digits. For a numeric element, this value is the number of digits after the decimal point. For other data types, the function does not return this value.                                                                                                                                                            |
| <i>nullable</i>   | SQL_SMALLINT | Output | Null indicator. The possible values are: <ul style="list-style-type: none"> <li>• SQL_NO_NULLS</li> <li>• SQL_NULLABLE</li> </ul>                                                                                                                                                                                            |
| <i>typename</i>   | SQL_CHAR     | Output | Type name. For a named row, this value is the name of the row. For collections and unnamed rows, the function does not return this value.                                                                                                                                                                                    |
| <i>typeowner</i>  | SQL_CHAR     | Output | Type owner. This value is the name of the owner of the data type. This name can be up to 18 characters long.                                                                                                                                                                                                                 |

## Usage

The `ifx_rc_describe()` function returns information about the data type for a row or collection or for an element that is in a row or collection. For elements that are in a collection, this information is the same for all elements that are in the collection. This function does not change the seek position.

## The `ifx_rc_fetch()` function

The `ifx_rc_fetch()` function retrieves the value of an element that is in a row or collection.

## Syntax

```
ifx_rc_fetch(result, rhandle, action, jump)
```

## Arguments

The function accepts the following arguments.

| Argument       | Type                     | Use    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------|--------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>result</i>  | Data type of the element | Output | Retrieved value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>rhandle</i> | HINFX_RC                 | Input  | Handle for a row or collection buffer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>action</i>  | SQL_SMALLINT             | Input  | Location of the element relative to the seek position. The possible values are: <ul style="list-style-type: none"> <li>• SQL_INFX_RC_ABSOLUTE: Element number <i>jump</i> where the first element in the buffer is element number one</li> <li>• SQL_INFX_RC_CURRENT: Current® element</li> <li>• SQL_INFX_RC_FIRST: First element</li> <li>• SQL_INFX_RC_LAST: Last element</li> <li>• SQL_INFX_RC_NEXT: Next element</li> <li>• SQL_INFX_RC_PRIOR: Previous element</li> <li>• SQL_INFX_RC_RELATIVE: Element that is <i>jump</i> elements past the current element</li> </ul> |
| <i>jump</i>    | SQL_SMALLINT             | Input  | Offset when <i>action</i> is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

## Usage

The `ifx_rc_fetch()` function retrieves the value of the element that is specified by *action* and *jump* and returns the value in *result*. The function sets the seek position to the position of the value that was just fetched.

## The ifx\_rc\_free() function

The ifx\_rc\_free() function frees a row or collection handle.

### Syntax

```
ifx_rc_free(rchandle)
```

### Arguments

The function accepts the following argument.

| Argument        | Type     | Use   | Description                           |
|-----------------|----------|-------|---------------------------------------|
| <i>rchandle</i> | HINFX_RC | Input | Handle for a row or collection buffer |

### Usage

The ifx\_rc\_free() function frees all the resources that are associated with a row or collection handle and frees the handle.

## The ifx\_rc\_insert() function

The ifx\_rc\_insert() function inserts a new element into a collection.

### Syntax

```
ifx_rc_insert(rchandle, value, action, jump)
```

### Arguments

The function accepts the following arguments.

| Argument        | Type                     | Use   | Description                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|--------------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rchandle</i> | HINFX_RC                 | Input | Handle for a collection buffer                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>value</i>    | Data type of the element | Input | Value to insert                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>action</i>   | SQL_SMALLINT             | Input | Location of the element relative to the seek position. The possible values are: <ul style="list-style-type: none"> <li>SQL_INFX_RC_ABSOLUTE: Element number <i>jump</i> where the first element in the buffer is element number one</li> <li>SQL_INFX_RC_CURRENT: Current@ element</li> <li>SQL_INFX_RC_FIRST: First element</li> <li>SQL_INFX_RC_LAST: Last element</li> <li>SQL_INFX_RC_NEXT: Next element</li> </ul> |

| Argument    | Type         | Use   | Description                                                                                                                                                                            |
|-------------|--------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             |              |       | <ul style="list-style-type: none"> <li>• SQL_INFX_RC_PRIOR: Previous element</li> <li>• SQL_INFX_RC_RELATIVE: Element that is <i>jump</i> elements past the current element</li> </ul> |
| <i>jump</i> | SQL_SMALLINT | Input | Offset when <i>action</i> is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE                                                                                                              |

## Usage

The `ifx_rc_insert()` function inserts a new element into a collection before the location that is specified by *action* and *jump*. The function sets the seek position to the position of the value that was inserted. It is not possible to insert a new element into a row.

The following table describes the allowable insertion locations for each type of collection.

| Type of collection | Allowable insertion locations |
|--------------------|-------------------------------|
| List               | Anywhere in the buffer        |
| Set or multiset    | At the end of the buffer      |

If the seek position specified by *action* and *jump* exceeds the end of the buffer, `ifx_rc_insert()` appends the new element at the end of the buffer. Likewise, if the seek position specified by *action* and *jump* precedes the beginning of the buffer, `ifx_rc_insert()` inserts the new element at the beginning of the buffer. If *action* specifies an insertion point other than the end for a set or multiset, `ifx_rc_insert()` fails.

For example, if *action* is SQL\_INFX\_RC\_LAST, the function inserts the new element before the last element. To append a new element, take one of the following actions:

- Set the seek position to the end of the buffer and set *action* to SQL\_INFX\_RC\_NEXT.
- Set *action* to SQL\_INFX\_RC\_ABSOLUTE or SQL\_INFX\_RC\_RELATIVE and set *jump* to a value that exceeds the end of the buffer.

To insert a new element at the beginning of a buffer, set *action* to SQL\_INFX\_RC\_FIRST.

## The `ifx_rc_isnull()` function

The `ifx_rc_isnull()` function returns a value that indicates whether a row or collection is null.

## Syntax

```
ifx_rc_isnull(nullflag, rchandle)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type         | Use    | Description                                                                                                                                                 |
|-----------------|--------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>nullflag</i> | SQL_SMALLINT | Output | Flag that indicates whether a row or collection is null. The possible values are: <ul style="list-style-type: none"> <li>• TRUE</li> <li>• FALSE</li> </ul> |
| <i>rchandle</i> | HINFX_RC     | Input  | Handle for a row or collection buffer                                                                                                                       |

## Usage

The `ifx_rc_isnull()` function returns a value that indicates whether a row or collection is null.

## The `ifx_rc_setnull()` function

The `ifx_rc_setnull()` function sets a row or collection to null.

## Syntax

```
ifx_rc_setnull(rchandle)
```

## Arguments

The function accepts the following argument.

| Argument        | Type     | Use   | Description                           |
|-----------------|----------|-------|---------------------------------------|
| <i>rchandle</i> | HINFX_RC | Input | Handle for a row or collection buffer |

## Usage

The `ifx_rc_setnull()` function sets a row or collection to null. The `ifx_rc_setnull()` function does not set each element within the row or collection to null.

## The `ifx_rc_typespec()` function

The `ifx_rc_typespec()` function returns the type specification for a row or collection.

## Syntax

```
ifx_rc_typespec(typespec, rchandle, flag)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type         | Use    | Description                                                                                                                                                                                                                    |
|-----------------|--------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>typespec</i> | SQL_CHAR     | Output | Type specification. The format for this value is the same as the type specification syntax for <code>ifx_rc_create()</code> .                                                                                                  |
| <i>rchandle</i> | HINFX_RC     | Input  | Handle for a row or collection buffer                                                                                                                                                                                          |
| <i>flag</i>     | SQL_SMALLINT | Input  | Flag that specifies whether to return the current or original type specification. If this value is TRUE, the function returns the original type specification. Otherwise, the function returns the current type specification. |

## Usage

The `ifx_rc_typespec()` function returns the type specification for a row or collection.

## The `ifx_rc_update()` function

The `ifx_rc_update()` function updates the value for an element that is in a row or collection.

## Syntax

```
ifx_rc_update(rchandle, value, action, jump)
```

## Arguments

The function accepts the following arguments.

| Argument        | Type                     | Use   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|--------------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rchandle</i> | HINFX_RC                 | Input | Handle for a row or collection buffer                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>value</i>    | Data type of the element | Input | Value with which to update the element                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>action</i>   | SQL_SMALLINT             | Input | Location of the element relative to the seek position. The possible values are: <ul style="list-style-type: none"> <li>• SQL_INFX_RC_ABSOLUTE: Element number <i>jump</i> where the first element in the buffer is element number one</li> <li>• SQL_INFX_RC_CURRENT: Current@ element</li> <li>• SQL_INFX_RC_FIRST: First element</li> <li>• SQL_INFX_RC_LAST: Last element</li> <li>• SQL_INFX_RC_NEXT: Next element</li> <li>• SQL_INFX_RC_PRIOR: Previous element</li> <li>• SQL_INFX_RC_RELATIVE: Element that is <i>jump</i> elements past the current element</li> </ul> |

| Argument    | Type         | Use   | Description                                                               |
|-------------|--------------|-------|---------------------------------------------------------------------------|
| <i>jump</i> | SQL_SMALLINT | Input | Offset when <i>action</i> is SQL_INFX_RC_ABSOLUTE or SQL_INFX_RC_RELATIVE |

## Usage

The `ifx_rc_update()` function updates the value for an element that immediately precedes the location that is specified by *action* and *jump*. The function sets the seek position to the position of the value that was updated.

## Improve application performance

These topics suggest ways to improve performance of HCL OneDB™ ODBC Driver applications.

### Error checking during data transfer

The **IFX\_LOB\_XFERSIZE** environment variable is used to specify the number of kilobytes in a CLOB or BLOB to transfer from a client application to the database server before checking whether an error has occurred.

The error check occurs each time the specified number of kilobytes is transferred. If an error occurs, the remaining data is not sent and an error is reported. If no error occurs, the file transfer continues until it finishes.

The valid range for **IFX\_LOB\_XFERSIZE** is from 1 to 9223372036854775808 kilobytes. The **IFX\_LOB\_XFERSIZE** environment variable is set on the client.

For more information about **IFX\_LOB\_XFERSIZE**, see the *HCL OneDB™ Guide to SQL: Reference*.

### Enable delimited identifiers in ODBC

By default delimited identifiers are disabled when connecting through ODBC.

There are three ways to enable them, listed here in order of decreasing precedence:

#### The **DELIMIDENT** connection string keyword

If you are using a connection string to connect you can set the **DELIMIDENT** keyword to enable or disable delimited identifiers. If the keyword is set to `y` then delimited identifiers are enabled for the connection. If the keywords are set to `n` delimited identifiers are disabled for the connection. If the keyword is present but is set to no value it has no effect on whether delimited identifiers are enabled.

For example, this connection string connects by using a data source name (DSN) of `mydsn` and enables delimited identifiers for the connection.

```
"DSN=mydsn;DELIMIDENT=y;"
```

This connection string also connects by using the DSN `mydsn` but has no effect on whether delimited identifiers are used.

```
"DSN=mydsn;DELIMIDENT=";
```



Setting the **DELIMIDENT** keyword in the connection string overrides any connection attributes or environment variables that enable or disable delimited identifiers.

### The **SQL\_INFX\_ATTR\_DELIMIDENT** connection attribute

You can enable or disable delimited identifiers for a given connection by setting the **SQL\_INFX\_ATTR\_DELIMIDENT** connection attribute before connecting. The **SQL\_INFX\_ATTR\_DELIMIDENT** connection attribute accepts the values listed in the following table.

**Table 9. Allowed values for the **SQL\_INFX\_ATTR\_DELIMIDENT** connection attribute**

| Value         | Effect                                                                                                                  |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| SQL_TRUE      | Delimited identifiers are enabled for the connection.                                                                   |
| SQL_FALSE     | Delimited identifiers are disabled for the connection.                                                                  |
| SQL_IFX_CLEAR | Clears any previous settings so that this connection attribute has no effect on whether delimited identifiers are used. |

For example, this call causes delimited identifiers to be enabled when the connection is made:

```
SQLSetConnectAttr(hdbc, SQL_INFX_ATTR_DELIMIDENT, SQL_TRUE, SQL_IS_INTEGER);
```

If this connection attribute is set to **SQL\_TRUE** or **SQL\_FALSE** the setting overrides the **DELIMIDENT** environment variable but not the **DELIMIDENT** connection string keyword.

### The **DELIMIDENT** environment variable

In some HCL OneDB™ APIs, such as ESQL/C, delimited identifiers are enabled by setting the **DELIMIDENT** environment variable to any value. In ODBC, however, delimited identifiers are enabled by setting the **DELIMIDENT** environment variable to `y` and are disabled by setting it to `n`.

## Connection level optimizations

Establishing a connection to a database is an expensive process. Optimally, an application performs as many tasks as possible while a connection is open.

This process can be achieved by:

- Pooling connections when using Windows™ Driver Manager
- Using multiple statement handles on the same connection handle

Also, you can fine tune application performance by setting the following connection level attributes:

- AutoCommit optimization
- Message transfer optimization (OPTMSG)
- Open-Fetch-Close optimization (OPTOFC)

## Optimizing query execution

There are several items you must consider when using prepared SQL queries.

Consider the following when using prepared SQL queries:

- `SQLExecDirect` is optimized for a single execution of an SQL statement. Thus, it is used for SQL queries that are not executed repeatedly.
- In cases where SQL queries are executed multiple times, using `SQLPrepare` and `SQLExecute` improves performance. Typically, you can do this with input and output parameters.
- SPL routines can be called from an ODBC application to perform certain SQL tasks and to expand what you can accomplish with SQL alone. Because SPL is native to the database and SPL routines are parsed and optimized at creation, rather than at runtime, SPL routines can improve performance for some tasks. SPL routines can also reduce traffic between a client application and the database server and reduce program complexity.
- When a stored procedure with a return value is executed using the HCL OneDB™ ODBC Driver, errors returned by the procedure are not returned to the application until a fetch is called on the result set. Error information from stored procedures with no returned values is available immediately following the execution of the procedure.

## Insert multiple rows

Use an insert cursor to efficiently insert rows into a table in bulk.

To create an insert cursor, set the `SQL_ENABLE_INSERT_CURSOR` attribute by using `SQLSetStmtOption`, then call `SQLParamOptions` with the number of rows as a parameter. You can create an insert cursor for data types `VARCHAR`, `LVARCHAR`, and `opaque`.

When you open an insert cursor, a buffer is created in memory to hold a block of rows. The buffer receives rows of data as the program produces them; then they are passed to the database server in a block when the buffer is full. The buffer reduces the amount of communication between the program and the database server. As a result, the insertions go faster.

## Automatically freeing a cursor

When an application uses a cursor, it usually sends a `FREE` statement to the database server to deallocate memory assigned to a cursor after it no longer needs that cursor.

Execution of this statement involves of message requests between the application and the database server. When the `AUTOFREE` is enabled, HCL OneDB™ ODBC Driver saves message requests because it does not need to execute the `FREE` statement. When the database server closes an insert cursor, it automatically frees the memory that it has allocated for it.

## Enabling the AUTOFREE feature

You can enable the `AUTOFREE` feature for an ODBC application in two ways.

### About this task

The `SQL_INFX_ATTR_AUTO_FREE` attribute can be set in any connection state between C2 and C5 (both included) when setting it using `SQLSetConnectAttr`, whereas it can be set by using `SQLSetStmtAttr` only when the statement is in S1

(allocated) state. The value of the `SQL_INFX_ATTR_AUTO_FREE` attribute can be retrieved by using `SQLGetConnectAttr` or `SQLSetStmtAttr`.

You can enable the `AUTOFREE` feature for an ODBC application in either of the following ways:

- Set the `SQL_INFX_ATTR_AUTO_FREE` attribute with `SQLSetConnectAttr`.

When you use `SQLSetConnectAttr` to enable this attribute, all new allocated statements for that connection inherit the attribute value. The only way to change this attribute value per statement is to set and reset it again as a statement attribute. The default is `DISABLED` for the connection attribute.

- Set the `SQL_INFX_ATTR_AUTO_FREE` attribute with `SQLSetStmtAttr`.

## The `AUTOFREE` feature

The `AUTOFREE` feature only works with result generating statements executed by using `SQLExecDirect`, as it opens the cursor which is then closed and released by the corresponding `SQLCloseCursor` or `SQLFreeStmt`.

The `AUTOFREE` feature does not work when the application has to prepare a statement once and then execute it several times (for example, using `SQLPrepare` to prepare and then executing it by calling `SQLExecute` several times). When you close the cursor with `SQLCloseCursor` after `SQLExecute`, it only closes the cursor but does not release the cursor memory on the database server side. But if you close the cursor by using `SQLFreeStmt` with `SQL_CLOSE` or `SQL_DROP`, it not only closes and releases the cursor, but it also unprepares the statement. In the latter case there is savings of a network roundtrip, but the application is unable to execute the statement again until it reprepares it.

When `AUTOFREE` is enabled, the application sees an improvement in the network performance when the application closes the cursor with `SQLCloseCursor` or `SQLFreeStmt` with `SQL_DROP`.

## Delay execution of the SQL `PREPARE` statement

You can defer execution of the `SQLPrepare` statement by enabling the `deferred-PREPARE` feature.

This feature works primarily with dynamic SQL statements where the application does a series of `SQLPrepare` and `SQLExecute` statements. It optimizes the number of round-trip messages to the database server by not sending `SQLPrepare` statements to the database server until the application calls `SQLExecute` on that statement.

When `deferred-PREPARE` is enabled, the following behavior is expected of the application:

- Execution of `SQLPrepare` does not put the statement in a prepared state.
- Syntax errors in an `SQLPrepare` statement are not known until the statement is executed because the SQL statement is never sent to the database server until it is executed. If open-fetch-close optimization is turned on, errors are not returned to the client until the first fetch, because open-fetch-close optimizes the `OPEN/FETCH` so that `OPEN` is sent on the first fetch.
- `SQLColAttributes`, `SQLDescribeCol`, `SQLNumResultCols`, and `SQLNumParams` always return `HY010` (function sequence error) if called after `SQLPrepare` but before `SQLExecute` by the application.

- SQLCopyDesc returns HY010 if the source descriptor handle is an IRD if called after SQLPrepare but before SQLExecute by the application.
- SQLGetDescField and SQLGetDescRec return HY010 if the descriptor handle is an IRD if called after SQLPrepare but before SQLExecute by the application.

You can enable the deferred-PREPARE feature for an ODBC application in either of the following ways:

- Set the SQL\_INFX\_ATTR\_DEFERRED\_PREPARE attribute with SQLSetConnectAttr.

When you use SQLSetConnectAttr to enable this attribute, all new allocated statements for that connection inherit the attribute value. The only way to change this attribute value per statement, is to set/reset it again as a statement attribute. The default is DISABLED for the connection attribute.

- Set the SQL\_INFX\_ATTR\_DEFERRED\_PREPARE attribute with SQLSetStmtAttr.

The SQL\_INFX\_ATTR\_DEFERRED\_PREPARE attribute can be set in any connection state between C2 and C5 (both included) when setting it using SQLSetConnectAttr, whereas it can be set by with SQLSetStmtAttr only when the statement is in S1 (allocated) state. The value of the SQL\_INFX\_ATTR\_DEFERRED\_PREPARE attribute can be retrieved with SQLGetConnectAttr or SQLSetStmtAttr.

## Set the fetch array size for simple-large-object data

To reduce the network overhead for fetches involving multiple rows of simple-large-object data, you can set the array size.

Set the array size so when the driver receives a multiple-row fetch request, it optimizes the fetch buffer size and the internal fetch array size, and eliminates a round trip to the database server for every simple large object.

Setting the array size greater than 1 can result in a performance improvement even for other types of data because it has the side effect of automatically increasing the fetch buffer size if necessary. (If the number of rows specified can fit in the current fetch buffer, setting it has little effect.)

An application can request that multiple rows be returned to it by setting the statement attribute SQL\_ATTR\_ROW\_ARRAY\_SIZE or setting the ARD header field SQL\_DESC\_ARRAY\_SIZE to a value greater than one, and then calling either SQLFetch or SQLFetchScroll. (The default value of SQL\_ATTR\_ROW\_ARRAY\_SIZE is one.) The driver then recognizes when it receives a multiple-row fetch request and optimizes the settings for the fetch buffer size and the internal fetch array size. Settings for these are based on the internal tuple size, the user setting of row array size, and the current setting of fetch array size.

You cannot use the internal fetch array feature under the following conditions:

- When OPTOFC and deferred-PREPARE are both enabled

To use the fetch array feature, the driver is dependent upon knowing how large a row is going to be, as received from the database server, before sending the fetch request to the database server. When both of these are enabled, this information is unavailable until after a fetch is performed.

- When using scroll cursors

There are separate internal client-to-server protocols used for scroll cursors that are distinct from those protocols used for fetching arrays. The database server does not support simple large object columns in a scroll cursor. An error is returned.

- When using SQLGetData

In order for the driver to use the fetch array feature, it has to be able to tell the database server how much data it is prepared to receive at the time of the fetch request. Calls to SQLGetData take place after SQLFetch.

According to the ODBC standard, when using block cursors, the application must call SQLSetPos to position the cursor on a particular row before calling SQLGetData. SQLSetPos is only usable with scroll cursors and simple-large-object columns are not allowed in scroll cursors. Also according to the standard, SQLGetData must not be used with a forward-only cursor with a rowset size greater than 1.

The alternative to using SQLGetData is to use SQLBindCol, which would come before the call to SQLFetch.

You might want to optimize use of SQL\_ATTR\_ROW\_ARRAY\_SIZE so the application sets the value of it according to the maximum number of rows that can be transported in a single buffer. After a statement is prepared, the application might call SQLGetStmtAttr to get the value of SQL\_INFX\_ATTR\_FET\_ARR\_SIZE. If the data fits in one fetch buffer, the internal setting of SQL\_INFX\_ATTR\_FET\_ARR\_SIZE equals the application setting of SQL\_ATTR\_ROW\_ARRAY\_SIZE. In practice, this is only useful on large result sets.

## The SPL output parameter feature

HCL OneDB™ ODBC Driver supports the ODBC defined method of getting the return value from a database procedure.

Specifically, ODBC supports the parameter to that precedes the equals sign in a procedure-call escape sequence. The host variable associated with that parameter is updated upon statement execution either with SQLExecute or SQLExecDirect.

In the HCL OneDB™ ODBC Driver definition of a procedure-call escape sequence, there is only one return value; therefore, the following restrictions are placed on this feature:

- Procedures used with this feature must return only one value, although they might return multiple rows.  
If this condition is not met, the parameter and its binding are ignored.
- Data from the first row only be placed in the host variable associated with the bound parameter, although procedures used with this feature can return multiple rows.

To return multiple-value, multiple-row result sets from the HCL OneDB™ database server, you have to fetch the data as though it were the result columns of a select statement. This output parameter feature works with existing applications that bind column or columnss and call SQLFetch or call SQLFetch and SQLGetData when accessing data through a procedure call. Therefore, no error or warning is generated when more than one row is available to be returned.

You can use either or both methods for retrieving the data from a stored procedure. A host variable can be bound as a parameter or as a column, or both. If separate buffers are used, only the host variable bound as a parameter is updated upon

statement execution, and only the host variable bound as a column is updated upon a fetch. Unbound columns accessed through SQLGetData remain unaffected.

## OUT and INOUT parameters

HCL OneDB™ Client Software Development Kit supports the use of OUT and INOUT parameters during execution of SPL.

The following data types are supported:

- BIGINT
- BLOB
- BOOLEAN
- DATETIME
- CHAR
- CLOB
- DECIMAL
- FLOAT
- INT8
- INTEGER
- INTERVAL
- LVARCHAR
- MONEY
- NCHAR
- NVARCHAR
- SMALLFLOAT
- SMALLINT
- VARCHAR

These restrictions exist when using OUT or INOUT parameters in SPL execution:

- Collection data types such as LIST, MULTISSET, ROW, and SET are not supported.
- Returning result sets is not supported. After executing SPL with OUT or INOUT parameters, you cannot call SQLFetch or SQLGetData.
- Only one value can be returned; that is, only one set of OUT or INOUT parameters can be returned per individual SPL execution.

The following SPL execution example creates one OUT, one INOUT, and one IN (default) parameter and one return value.

```
create procedure myproc(OUT intparam INT, INOUT charparam char(20),
 inparam int) returns int
<body of SPL>
end procedure;
```

The following code example, `outinoutparamblob.c`, shows how to use OUT and INOUT parameters with BLOB, INTEGER, and VARCHAR data types.

```

/* Drop procedure */
SQLExecDirect(hstmt, (UCHAR *)"drop procedure spl_out_param_blob;", SQL_NTS);
SQLExecDirect(hstmt, (UCHAR *)"drop table tab_blob;", SQL_NTS);

/* Create table with BLOB column */
rc = SQLExecDirect(hstmt, (UCHAR *)"create table tab_blob(c_blob BLOB,
c_int INTEGER, c_char varchar(20));", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error in Step 2 --
SQLExecDirect failed\n"))
goto Exit;

/* Insert one row into the table */
rc = SQLExecDirect(hstmt, (UCHAR *)"insert into tab_blob
values(filetoblob('insert.data', 'c'), 10, 'blob_test');", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error in Step 2
-- SQLExecDirect failed\n"))
goto Exit;

/* Create procedure */
rc = SQLExecDirect(hstmt, "CREATE PROCEDURE spl_out_param_blob(inParam int,
OUT blobparam BLOB, OUT intparam int, OUT charparam varchar(20)) \n"
"returning integer; \n"
"select c_blob, c_int, c_char into blobparam,
intparam, charparam from tab_blob; \n"
"return inParam; \n"
"end procedure; ",
SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error in Step 2
-- SQLExecDirect failed\n"))
goto Exit;

/* Prepare stored procedure to be executed */
rc = SQLPrepare(hstmt, (UCHAR *)"{? = call spl_out_param_blob
(?, ?, ?, ?)}", SQL_NTS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLPrepare failed\n"))
goto Exit;

/* Bind the required parameters */
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_LONG,
SQL_INTEGER, 3, 0, &sParm1, 0, &cbParm1);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 1 failed\n"))
goto Exit;

rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 10, 0, &sParm2, 0, &cbParm2);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 2 failed\n"))
goto Exit;

rc = SQLBindParameter(hstmt, 3, SQL_PARAM_OUTPUT, SQL_C_BINARY,
SQL_LONGVARBINARY, sizeof(blob_buffer), 0, blob_buffer,
sizeof(blob_buffer), &cbParm3);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 3 failed\n"))
goto Exit;

```

```

rc = SQLBindParameter(hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_LONG,
SQL_INTEGER, 10, 0, &sParm3, 0, &cbParm4);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 4 failed\n"))
goto Exit;

rc = SQLBindParameter (hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR,
SQL_VARCHAR, sizeof(schar), 0, schar, sizeof(schar), &cbParm6);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLBindParameter 5 failed\n"))
goto Exit;

/* Exeute the prepared stored procedure */
rc = SQLExecute(hstmt);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 2 -- SQLExecute failed\n"))
goto Exit;

len =
strlen("1234567890123456789012345678901234567890 ");

if((sParm2 != sParm1) || (10 != sParm3) ||
(strcmp("blob_test", schar)) || (cbParm3 != len))
{
fprintf(stdout, "\n 1st Data compare failed!");
goto Exit;
}
else
{
fprintf(stdout, "\n 1st Data compare successful");
}

/* Reset the parameters */
rc = SQLFreeStmt(hstmt, SQL_RESET_PARAMS);
if (checkError (rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *)
"Error in Step 3 -- SQLFreeStmt failed\n"))
goto Exit;

/* Reset variables */
sParm1 = 0;
cbParm6 = cbParm1 = SQL_NTS;
cbParm3 = SQL_NULL_DATA;
schar[0]=0;
blob_buffer[0]=0;

```

## Asynchronous execution

Design your application to take advantage of data sources that support asynchronous execution. Asynchronous calls do not perform faster, but well-designed applications appear to run more efficiently.

Turning on asynchronous execution does not by itself improve performance. Well-designed applications, however, can take advantage of asynchronous query execution by allowing the user to work on other things while the query is being evaluated on the database server. Perhaps users start one or more subsequent queries or choose to work in another application, all



while the query is executing on the database server. Designing for asynchronous execution makes your application appear to run faster by allowing the user to work concurrently on multiple tasks.

By default, an application calls to an ODBC driver that then executes statements against the database server in a synchronous manner. In this mode of operation, the driver does not return control to the application until its own request to the database server is complete. For statements that take more than a few seconds to complete execution, this control return delay can result in the perception of poor performance.

Some data sources support asynchronous execution. When in asynchronous mode, an application calls to an ODBC driver and control is returned almost immediately. In this mode, the driver returns the status `SQL_STILL_EXECUTING` to the application and then sends the appropriate request to the database server for execution. The application polls the driver at various intervals at which point the driver itself polls the database server to see if the query has completed execution. If the query is still executing, then the status `SQL_STILL_EXECUTING` is returned to the application. If it has completed, then a status such as `SQL_SUCCESS` is returned, and the application can then begin to fetch records.

## Update data with positioned updates and deletes

Although positioned updates do not apply to all types of applications, try to use positioned updates and deletes whenever possible.

Positioned updates (with `UPDATE WHERE CURRENT OF CURSOR`) allow you to update data by positioning the database cursor to the row to be changed and signaling the driver to change the data. You are not forced to build a complex SQL statement; you supply the data to be changed.

Besides making the code more maintainable, positioned updates typically result in improved performance. Because the database server is already positioned on the row (for the `SELECT` statement currently in process), expensive operations to locate the row to be changed are unnecessary. If the row must be located, the database server typically has an internal pointer to the row available (for example, `ROWID`).

To support positioned `UPDATE` and `DELETE` statements with scrollable cursors, HCL OneDB™ ODBC Driver constructs a new searched `UPDATE` or `DELETE` statement from the original positioned statement. However, the database server cannot update scroll cursors directly. Instead, HCL OneDB™ ODBC Driver constructs a `WHERE` clause that references each column fetched in the `SELECT` statement referenced in the `WHERE CURRENT OF CURSOR` clause. Values from the rowset data cache of the `SELECT` statement are bound to each value in the constructed `WHERE` clause.

This method of positioning is both slower and more error prone than using a `WHERE CURRENT OF CURSOR` clause with `FORWARD ONLY` cursors. If the fetched rows do not contain a unique key value, the constructed `WHERE` clause might identify one or many rows, causing many rows to be deleted or updated. Deletion of rows in this manner affects both positioned `UPDATE` and `DELETE` statements, and `SQLSetPos` statements when you use scroll cursors.

Use `SQLSpecialColumns` to determine the optimal set of columns to use in the `WHERE` clause for updating data. Many times pseudocolumns provide the fastest access to the data; you can determine these columns only by using `SQLSpecialColumns`.

Many applications cannot be designed to take advantage of positioned updates and deletes. These applications typically update data by forming a `WHERE` clause that consists of some subset of the column values that are returned in the result

set. Some applications might formulate the WHERE clause by using all searchable result columns or by calling SQLStatistics to find columns that might be part of a unique index. These methods typically work but can result in fairly complex queries.

Consider the following example:

```
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
 address, city, state, zip FROM emp", SQL_NTS);
// fetchdata
;
rc = SQLExecDirect (hstmt, "UPDATE EMP SET ADDRESS = ?
 WHERE first_name = ? AND last_name = ? AND ssn = ? AND
 address = ? AND city = ? AND state = ? AND zip = ?", SQL_NTS);
// fairly complex query
```

Applications should call SQLSpecialColumns/SQL\_BEST\_ROWID to retrieve the optimal set of columns (possibly a pseudocolumn) that identifies any given record. Many databases support special columns that are not explicitly user-defined in the table definition but are hidden columns of every table (for example, ROWID, TID, and other columns). These pseudocolumns almost always provide the fastest access to the data because they typically are pointers to the exact location of the record. Because pseudocolumns are not part of the explicit table definition, they are not returned from SQLSpecialColumns. The only way to determine whether pseudocolumns exist is to call SQLSpecialColumns.

Consider the previous example, this time with SQLSpecialColumns:

```
;
rc = SQLSpecialColumns (hstmt, 'emp', ...);
;
rc = SQLExecDirect (hstmt, "SELECT first_name, last_name, ssn,
 address, city, state, zip, ROWID FROM emp", SQL_NTS);
// fetch data and probably "hide" ROWID from the user
;
rc = SQLExecDirect (hstmt, "UPDATE emp SET address = ? WHERE
 ROWID = ?", SQL_NTS);
// fastest access to the data!
```

If your data source does not contain special pseudocolumns, the result set of SQLSpecialColumns consists of the columns of the optimal unique index on the specified table (if a unique index exists). Therefore, your application does not additionally call SQLStatistics to find the smallest unique index.

## BIGINT and BIGSERIAL data types

BIGINT and BIGSERIAL data types have the same range of values as INT8 and SERIAL8 data types.

However, BIGINT and BIGSERIAL have advantages for storage and computation over INT8 and SERIAL8.

## Message transfer optimization

If you activate the message transfer optimization feature (OPTMSG), the driver minimizes message transfers with the database server for most HCL OneDB™ ODBC functions.

In addition, the driver chains messages from the database server together and eliminates some small message packets to accomplish optimized message transfers.

To activate message transfer optimization, set the `SQL_INFX_ATTR_OPTMSG` statement attribute to one (1). The optimization default is: OFF.

## Message chaining restrictions

HCL OneDB™ ODBC does not chain SQL functions even when you enable message transfer optimization.

The SQL functions that ODBC does not chain are:

- `SQLDisconnect`
- `SQLConnect`
- `SQLEndTran`
- `SQLExecute` (if the driver returns results by using the select or call procedure and when the driver uses insert cursors to perform a bulk insert)
- `SQLExtendedFetch`
- `SQLFetch`
- `SQLFetchScroll`
- `SQLPrepare`

When the driver reaches one of the functions listed previously, it performs the following actions:

1. Flushes the message queue to the database server only when it encounters SQL statements that require a response from the database server.

The driver does not flush the message queue when it encounters functions that do not require network traffic, such as `SQLAllocStmt`.

2. Continues message chaining for subsequent SQL statements.

## Disable message chaining

You can choose to disable message chaining.

Before you disable message chaining, consider the following situations:

- Some SQL statements require immediate replies. If you disable message chaining, re-enable the `OPTMSG` feature after the restricted SQL statement is completed.
- If you perform debugging, you can disable the `OPTMSG` feature when you are trying to determine how each SQL statement responds.
- If you enable `OPTMSG`, the message is queued up for the database server but it is not sent for processing. Consider disabling message chaining before the last SQL statement in the program to ensure that the database server processes all messages before the application exits.
- If you disable message chaining, you must reset the `SQL_INFX_ATTR_OPTMSG` attribute immediately after the SQL statement that requires it to avoid unintended chaining.

The following example shows how to disable message chaining by placing the `SQL_INFX_ATTR_OPTMSG` attribute after the `DELETE` statement. If you place the attribute after the delete statement, the driver can flush all the queued messages when the next SQL statement executes.

```
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 1);
SQLExecDirect(hstmt, (unsigned char *)
"delete from customer", SQL_NTS);
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 0);
SQLExecDirect(hstmt, (unsigned char *)
"create index ix1 on customer (zipcode)", SQL_NTS);
```

Unintended message chaining can make it difficult to determine which of the chained statements failed.

At the `CREATE INDEX` statement, the driver sends both the `DELETE` and the `CREATE INDEX` statements to the database server.

## Errors with optimized message transfers

When you enable the `OPTMSG` feature, HCL OneDB™ ODBC does not perform error handling on any chained statement.

If you are not sure whether a particular statement might generate an error, include error-handling statements in your code and do not enable message chaining for that statement.

The database server stops execution of subsequent statements when an error occurs in a chained statement. For example, in the following code fragment, the intent is to chain five `INSERT` statements:

```
SQLExecDirect(hstmt, "create table tab1 (col1 INTEGER)", SQL_NTS);
/* enable message chaining */
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 1);
/* these two INSERT statements execute successfully */
SQLExecDirect(hstmt, "insert into tab1 values (1)", SQL_NTS);
SQLExecDirect(hstmt, "insert into tab1 values (2)", SQL_NTS);
/* this INSERT statement generates an error because the data
 * in the VALUES clause is not compatible with the column type */
SQLExecDirect(hstmt, "insert into tab1 values ('a')", SQL_NTS);
/* these two INSERT statements never execute */
SQLExecDirect(hstmt, "insert into tab1 values (3)", SQL_NTS);
SQLExecDirect(hstmt, "insert into tab1 values (4)", SQL_NTS);
/* disable message chaining */
SQLSetStmtOption(hstmt, SQL_INFX_ATTR_OPTMSG, 0);
/* commit work */
rc = SQLEndTran (SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
if (rc != SQL_SUCCESS)
```

In this example, the following actions occur:

- The driver sends the five `INSERT` statements and the `COMMIT WORK` statements to the database server for execution.
- The database inserts `col1` values of 1 and 2 into the `tab1` table.
- The third `INSERT` statement generates an error, so the database server does not execute the subsequent `INSERT` statements or the `COMMIT WORK` statement.

- The driver flushes the message queue when the queue reaches the SQLEndTran function.
- The SQLEndTran function, which is the last statement in the chained statements, returns the error from the failed INSERT statement.

If you want to keep the values that the database server inserted into col1, you must commit them yourself.

## Error messages

These topics describe the HCL OneDB™ ODBC Driver error messages.

The topics provide information about:

- Diagnostic SQLSTATE values
- SQLSTATE values mapped to HCL OneDB™ error messages
- HCL OneDB™ ODBC Driver error messages mapped to specific SQLSTATE values

For a description of an error message, use the finderr utility.

## Diagnostic SQLSTATE values

Each HCL OneDB™ ODBC Driver function can return an SQLSTATE value that corresponds to the HCL OneDB™ error code.

A function can return additional SQLSTATE values that arise from implementation-specific situations. SQLERROR returns SQLSTATE values as defined by the GLS and SQL Access Group SQL CAE specification (1992).

SQLSTATE values are character strings that consist of a two-character class value followed by a three-character subclass value. A class value of 01 indicates a warning and is accompanied by a return code of SQL\_SUCCESS\_WITH\_INFO. Class values other than 01, except for the class 1M, indicate an error and are accompanied by a return code of SQL\_ERROR. The class 1M signifies warnings and errors that derive from the implementation of HCL OneDB™ ODBC Driver. The subclass value 000 in any class is for implementation-defined conditions within the given class. ANSI SQL-92 defines the assignment of class and subclass values.

## Map SQLSTATE values to HCL OneDB™ error messages

View the SQLSTATE values that HCL OneDB™ ODBC Driver can return.

The following table maps SQLSTATE values that HCL OneDB™ ODBC Driver can return.

A return value of SQL\_SUCCESS normally indicates a function has executed successfully, although the SQLSTATE 00000 also indicates success.

SQLSTATE	Error message	Can be returned from
01000	General warning	All HCL OneDB™ ODBC Driver functions except:

SQLSTATE	Error message	Can be returned from
		SQLAllocEnv SQLError
01002	Disconnect error	SQLDisconnect
01004	Data truncated	SQLBrowseConnect SQLColAttributes SQLDataSources SQLDescribeCol SQLDriverConnect SQLDrivers SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLGetCursorName SQLGetData SQLGetInfo SQLNativeSql SQLPutData SQLSetPos
01006	Privilege not revoked	SQLExecDirect SQLExecute
01S00	Invalid connection string attribute	SQLBrowseConnect SQLDriverConnect
01S01	Error in row	SQLExtendedFetch SQLSetPos
01S02	Option value changed	SQLSetConnectOption SQLSetStmtOption
01S03	No rows updated or deleted	SQLExecDirect SQLExecute SQLSetPos

SQLSTATE	Error message	Can be returned from
01S04	More than one row updated or deleted	SQLExecDirect SQLExecute SQLSetPos
07001	Wrong number of parameters	SQLExecDirect SQLExecute
07006	Restricted data type attribute violation	SQLBindParameter SQLExtendedFetch SQLFetch SQLGetData
08001	Unable to connect to data source	SQLBrowseConnect SQLConnect SQLDriverConnect
08002	Connection in use	SQLBrowseConnect SQLConnect SQLDriverConnect SQLSetConnectOption
08003	Connection not open	SQLAllocStmt SQLDisconnect SQLGetConnectOption SQLGetInfo SQLNativeSql SQLSetConnectOption SQLTransact
08004	Data source rejected establishment of connection	SQLBrowseConnect SQLConnect SQLDriverConnect
08007	Connection failure during transaction	SQLTransact
08S01	Communication link failure	SQLBrowseConnect SQLColumnPrivileges SQLColumns SQLConnect SQLDriverConnect

SQLSTATE	Error message	Can be returned from
		SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLForeignKeys SQLFreeConnect SQLGetData SQLGetTypeInfo SQLParamData SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLPutData SQLSetConnectOption SQLSetStmtOption SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables
21S01	Insert value list does not match column list	SQLExecDirect SQLPrepare
21S02	Degree of derived table does not match column list	SQLExecDirect SQLPrepare SQLSetPos
22001	String data right truncation	SQLPutData
22003	Numeric value out of range	SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLGetData SQLGetInfo SQLPutData SQLSetPos
22005	Error in assignment	SQLExecDirect SQLExecute



SQLSTATE	Error message	Can be returned from
		SQLExtendedFetch SQLFetch SQLGetData SQLPrepare SQLPutData SQLSetPos
22008	Datetime field overflow	SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLGetData SQLPutData SQLSetPos
22012	Division by zero	SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLGetData
22026	String data, length mismatch	SQLParamData
23000	Integrity constraint violation	SQLExecDirect SQLExecute SQLSetPos
24000	Invalid cursor state	SQLColAttributes SQLColumnPrivileges SQLColumns SQLDescribeCol SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLForeignKeys SQLGetData SQLGetStmtOption SQLGetTypeInfo SQLPrepare SQLPrimaryKeys

SQLSTATE	Error message	Can be returned from
		SQLProcedureColumns SQLProcedures SQLSetCursorName SQLSetPos SQLSetStmtOption SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables
25000	Invalid transaction state	SQLDisconnect
28000	Invalid authorization specification	SQLBrowseConnect SQLConnect SQLDriverConnect
34000	Invalid cursor name	SQLExecDirect SQLPrepare SQLSetCursorName
37000	Syntax error or access violation	SQLExecDirect SQLNativeSql SQLPrepare
3C000	Duplicate cursor name	SQLSetCursorName
40001	Serialization failure	SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch
42000	Syntax error or access violation	SQLExecDirect SQLExecute SQLPrepare SQLSetPos
70100	Operation aborted	SQLCancel
IM001	Driver does not support this function	All ODBC functions except: SQLAllocConnect SQLAllocEnv SQLDataSources

SQLSTATE	Error message	Can be returned from
		SQLDrivers SQLError SQLFreeConnect SQLFreeEnv SQLGetFunctions
IM002	Data source name not found and no default driver specified	SQLBrowseConnect SQLConnect SQLDriverConnect
IM003	Specified driver could not be loaded	SQLBrowseConnect SQLConnect SQLDriverConnect
IM004	Driver's SQLAllocEnv failed	SQLBrowseConnect SQLConnect SQLDriverConnect
IM005	Driver's SQLAllocConnect failed	SQLBrowseConnect SQLConnect SQLDriverConnect
IM006	Driver's SQLSetConnectOption failed	SQLBrowseConnect SQLConnect SQLDriverConnect
IM007	No data source or driver specified; dialog prohibited	SQLDriverConnect
IM008	Dialog failed	SQLDriverConnect
IM009	Unable to load translation shared library (DLL)	SQLBrowseConnect SQLConnect SQLDriverConnect SQLSetConnectOption
IM010	Data source name too long	SQLBrowseConnect SQLDriverConnect
IM011	Driver name too long	SQLBrowseConnect SQLDriverConnect

SQLSTATE	Error message	Can be returned from
IM012	DRIVER keyword syntax error	SQLBrowseConnect SQLDriverConnect
IM013	Trace file error	All ODBC functions.
S0001	Base table or view already exists	SQLExecDirect SQLPrepare
S0002	Base table not found	SQLExecDirect SQLPrepare
S0011	Index already exists	SQLExecDirect SQLPrepare
S0012	Index not found	SQLExecDirect SQLPrepare
S0021	Column already exists	SQLExecDirect SQLPrepare
S0022	Column not found	SQLExecDirect SQLPrepare
S0023	No default for column	SQLSetPos
S1000	General error	All ODBC functions except:
S1001	Memory allocation failure	All ODBC functions except: SQLAllocEnv SQLError SQLFreeConnect SQLFreeEnv
S1002	Invalid column number	SQLBindCol SQLColAttributes SQLDescribeCol SQLExtendedFetch SQLFetch SQLGetData

SQLSTATE	Error message	Can be returned from
S1003	Program type out of range	SQLBindCol SQLBindParameter SQLGetData
S1004	SQL data type out of range	SQLBindParameter SQLGetTypeInfo
S1008	Operation canceled	All ODBC functions that can be processed asynchronously:  SQLColAttributes SQLColumnPrivileges SQLColumns SQLDescribeCol SQLDescribeParam SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLForeignKeys SQLGetData SQLGetTypeInfo SQLMoreResults SQLNumParams SQLNumResultCols SQLParamData SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLPutData SQLSetPos SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables
S1009	Invalid argument value	SQLAllocConnect SQLAllocStmt SQLBindCol SQLBindParameter

SQLSTATE	Error message	Can be returned from
		SQLExecDirect SQLForeignKeys SQLGetData SQLGetInfo SQLNativeSql SQLPrepare SQLPutData SQLSetConnectOption SQLSetCursorName SQLSetPos SQLSetStmtOption
S1010	Function sequence error	SQLBindCol SQLBindParameter SQLColAttributes SQLColumnPrivileges SQLColumns SQLDescribeCol SQLDisconnect SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLForeignKeys SQLFreeConnect SQLFreeEnv SQLFreeStmt SQLGetConnectOption SQLGetCursorName SQLGetData SQLGetFunctions SQLGetStmtOption SQLGetTypeInfo SQLMoreResults SQLNumParams SQLNumResultCols SQLParamData SQLParamOptions SQLPrepare SQLPrimaryKeys

SQLSTATE	Error message	Can be returned from
		SQLProcedureColumns SQLProcedures SQLPutData SQLRowCount SQLSetConnectOption SQLSetCursorName SQLSetPos SQLSetScrollOptions SQLSetStmtOption SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables SQLTransact
S1011	Operation invalid at this time	SQLGetStmtOption SQLSetConnectOption SQLSetStmtOption
S1012	Invalid transaction operation code specified	SQLTransact
S1015	No cursor name available	SQLGetCursorName
S1090	Invalid string or buffer length	SQLBindCol SQLBindParameter SQLBrowseConnect SQLColAttributes SQLColumnPrivileges SQLColumns SQLConnect SQLDataSources SQLDescribeCol SQLDriverConnect SQLDrivers SQLExecDirect SQLExecute SQLForeignKeys SQLGetCursorName SQLGetData SQLGetInfo SQLNativeSql

SQLSTATE	Error message	Can be returned from
		SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLPutData SQLSetCursorName SQLSetPos SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables
S1091	Descriptor type out of range	SQLColAttributes
S1092	Option type out of range	SQLFreeStmt SQLGetConnectOption SQLGetStmtOption SQLSetConnectOption SQLSetStmtOption
S1093	Invalid parameter number	SQLBindParameter
S1094	Invalid scale value	SQLBindParameter
S1095	Function type out of range	SQLGetFunctions
S1096	Information type out of range	SQLGetInfo
S1097	Column type out of range	SQLSpecialColumns
S1098	Scope type out of range	SQLSpecialColumns
S1099	Nullable type out of range	SQLSpecialColumns
S1100	Uniqueness option type out of range	SQLStatistics
S1101	Accuracy option type out of range	SQLStatistics
S1103	Direction option out of range	SQLDataSources SQLDrivers
S1104	Invalid precision value	SQLBindParameter
S1105	Invalid parameter type	SQLBindParameter
S1106	Fetch type out of range	SQLExtendedFetch



SQLSTATE	Error message	Can be returned from
S1107	Row value out of range	SQLExtendedFetch SQLParamOptions SQLSetPos SQLSetScrollOptions
S1108	Concurrency option out of range	SQLSetScrollOptions
S1109	Invalid cursor position	SQLExecute SQLExecDirect SQLGetData SQLGetStmtOption SQLSetPos
S1110	Invalid driver completion	SQLDriverConnect
S1111	Invalid bookmark value	SQLExtendedFetch
S1C00	Driver not capable	SQLBindCol SQLBindParameter SQLColAttributes SQLColumnPrivileges SQLColumns SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLForeignKeys SQLGetConnectOption SQLGetData SQLGetInfo SQLGetStmtOption SQLGetTypeInfo SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLSetConnectOption SQLSetPos SQLSetScrollOptions SQLSetStmtOption SQLSpecialColumns

SQLSTATE	Error message	Can be returned from
		SQLStatistics SQLTablePrivileges SQLTables SQLTransact
S1T00	Time-out expired	SQLBrowseConnect SQLColAttributes SQLColumnPrivileges SQLColumns SQLConnect SQLDescribeCol SQLDriverConnect SQLExecDirect SQLExecute SQLExtendedFetch SQLFetch SQLForeignKeys SQLGetData SQLGetInfo SQLGetTypeInfo SQLMoreResults SQLNumParams SQLNumResultCols SQLParamData SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLPutData SQLSetPos SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables

### Map HCL OneDB™ error messages to SQLSTATE values

The rest of this section describes diagnostic SQLSTATE values for HCL OneDB™ ODBC Driver functions.

The return code for each SQLSTATE value is SQL\_ERROR unless a description indicates otherwise. When a function returns SQL\_SUCCESS\_WITH\_INFO or SQL\_ERROR, you can call SQLError to get the SQLSTATE value.

## SQLAllocConnect (core level only)

This table describes the SQLSTATE and error values for SQLAllocConnect.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value

## SQLAllocEnv (core level only)

SQLAllocEnv allocates memory for an environment handle and initializes the driver call level interface for application use.

An application must call SQLAllocEnv before it calls any other driver function.

A driver cannot return SQLSTATE values directly after the call to SQLAllocEnv because no valid handle exists with which to call SQLError.

Two levels of SQLAllocEnv functions exist, one within the driver manager (if you are using one) and one within the driver. The driver manager does not call the driver-level function until the application calls SQLConnect, SQLBrowseConnect, or SQLDriverConnect. If an error occurs in the driver-level SQLAllocEnv function, the driver manager-level SQLConnect, SQLBrowseConnect, or SQLDriverConnect function returns SQL\_ERROR. A subsequent call to SQLError with *henv*, SQL\_NULL\_HDBC, and SQL\_NULL\_HSTMT returns SQLSTATE IM004 (the driver SQLAllocEnv function failed), followed by one of the following errors from the driver:

- SQLSTATE S1000 (General error)
- The HCL OneDB™ ODBC Driver SQLSTATE value, which ranges from S1000 to S19ZZ.

For example, SQLSTATE S1001 (Memory-allocation failure) indicates that the call from the driver manager to the driver-level SQLAllocEnv returned SQL\_ERROR, and the *henv* from the driver manager was set to SQL\_NULL\_HENV.

## SQLAllocStmt (core level only)

SQLAllocStmt allocates memory for a statement handle and associates the statement handle with the connection that *hdbc* specifies.

An application must call SQLAllocStmt before it submits SQL statements.

The following table describes the SQLSTATE and error values for SQLAllocStmt.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08003	-11017	Connection not open

SQLSTATE	Error value	Error message
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
08S01	-11301	A protocol error has been detected. Current connection is closed.

## SQLBindCol (core level only)

SQLBindCol assigns the storage and HCL OneDB™ ODBC Driver C data type for a column in a result set.

The SQLBindCol assigns the storage as follows:

- A storage buffer that receives the contents of a column of data
- The length of the storage buffer
- A storage location that receives the actual length of the column of data returned by the fetch operation
- Data type conversion from the HCL OneDB™ SQL data type to the HCL OneDB™ ODBC driver C data type

The following table describes the SQLSTATE and error values for SQLBindCol.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1003	-11063	Program type out of range
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable



**Important:** An application can call SQLBindCol to bind a column to a new storage location, regardless of whether data has already been fetched. The new binding replaces the old binding for bookmark columns as well as other bound columns. The new binding does not apply to data already fetched; it takes effect the next time SQLFetch, SQLExtendedFetch, or SQLSetPos is called.

## SQLBindParameter (level one only)

SQLBindParameter binds a buffer to a parameter marker in an SQL statement.

The following table describes the SQLSTATE and error values for SQLBindParameter.

SQLSTATE	Error value	Error message
01000	-11001	General warning
07006	-11013	Restricted data type attribute violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1003	-11063	Program type out of range
S1004	-11064	SQL data type out of range
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1093	-11074	Invalid parameter number
S1094	-11075	Invalid scale value
S1104	-11084	Invalid precision value
S1105	-11085	Invalid parameter type
S1C00	-11092	Driver not capable

## SQLBrowseConnect (level two only)

SQLBrowseConnect supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source.

Each call to SQLBrowseConnect returns successive levels of attributes and attribute values. When all levels are enumerated, a connection to the data source is completed, and a SQLBrowseConnect string is returned with a return code of now connected to the data source.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
01S00	-11005	Invalid connection string attribute
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification

SQLSTATE	Error value	Error message
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver's SQLAllocEnv failed
IM005	-11044	Driver's SQLAllocConnect failed
IM006	-11045	Driver's SQLSetConnectOption failed
IM009	-11048	Unable to load translation shared library (DLL)
IM010	-11049	Data-source name too long
IM011	-11050	Driver name too long
IM012	-11051	DRIVER keyword syntax error
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11303	Input connection string too large
S1000	-11317	Invalid connectdatabase value specified
S1000	-11318	Invalid vmbcharlenexact value specified
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLCancel (core level only)

SQLCancel cancels the processing on an *hstmt* or a query.

The following table describes the SQLSTATE and error values for the function.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01S05	-11010	Cancel treated as FreeStmt/Close.
70100	-11039	Operation aborted
S1000	-11060	General error
S1001	-11061	Memory-allocation failure

SQLSTATE	Error value	Error message
08S01	-11301	A protocol error has been detected. Current connection is closed.

## SQLColAttributes (core level only)

SQLColAttributes returns descriptor information for a column in a result set.

It cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

The following table describes the SQLSTATE and error values for SQLColAttributes.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1091	-11072	Descriptor type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

SQLColAttributes can return any SQLSTATE that can be returned by SQLPrepare or SQLExecute when it is called after SQLPrepare and before SQLExecute, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

## SQLColumnPrivileges (level two only)

SQLColumnPrivileges returns a list of columns and associated privileges for the specified table. The driver returns the information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for SQLColumnPrivileges.

SQLSTATE	Error value	Error message
01000	-11001	General warning

SQLSTATE	Error value	Error message
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLColumns (level one only)

SQLColumns returns the list of column names in specified tables. The driver returns this information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for SQLColumns.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable



SQLSTATE	Error value	Error message
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLConnect (core level only)

SQLConnect loads a driver and establishes a connection to a data source.

The connection handle references where all information about the connection, including status, transaction state, and error information is stored.

The following table describes the SQLSTATE and error values for SQLConnect.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver's SQLAllocEnv failed
IM005	-11044	Driver's SQLAllocConnect failed
IM006	-11045	Driver's SQLSetConnectOption failed
IM009	-11048	Unable to load translation shared library (DLL)
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired
S1000	-11302	Insufficient connection information was supplied

SQLSTATE	Error value	Error message
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLDataSources (level two only)

SQLDataSources lists data-source names.

The following table describes the SQLSTATE and error values for SQLDataSources.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1103	-11083	Direction option out of range

## SQLDescribeCol (core level only)

SQLDescribeCol returns the result descriptor (column name, type, precision, scale, and whether it can have a NULL value) for one column in the result set.

It cannot be used to return information about the bookmark column (column 0).

The following table describes the SQLSTATE and error values for SQLDescribeCol.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length

SQLSTATE	Error value	Error message
S1T00	-11094	Time-out expired

SQLDescribeCol can return any SQLSTATE that SQLPrepare or SQLExecute returns when SQLDescribeCol is called after SQLPrepare and before SQLExecute, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

## SQLDisconnect

SQLDisconnect closes the connection associated with a specific connection handle.

The following table describes the SQLSTATE and error values for SQLDisconnect.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01002	-11002	Disconnect error
08003	-11017	Connection not open
25000	-11032	Invalid transaction state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
08S01	-11301	A protocol error has been detected. Current connection is closed.

## Usage

If an application calls SQLDisconnect after SQLBrowseConnect returns SQL\_NEED\_DATA and before it returns a different return code, the driver cancels the connection-browsing process and returns the *hdbc* to an unconnected state.

If an application calls SQLDisconnect while an incomplete transaction is associated with the connection handle, the driver returns SQLSTATE 25000 (Invalid transaction state), indicating that the transaction is unchanged and the connection is open. An incomplete transaction is one that was not committed or rolled back with SQLTransact.

If an application calls SQLDisconnect before it frees every *hstmt* associated with the connection, the driver frees each remaining *hstmt* after it successfully disconnects from the data source. However, if one or more of the *hstmts* associated with the connection are still executing asynchronously, SQLDisconnect returns SQL\_ERROR with an SQLSTATE value of S1010 (Function sequence error).

## SQLDriverConnect (level one only)

SQLDriverConnect is an alternative to SQLConnect.

It supports data sources that require more connection information than the three arguments in SQLConnect dialog boxes to prompt the user for all connection information and data sources that are not defined data source names.

SQLDriverConnect provides the following connection options:

- You can establish a connection by using a connection string that contains the data source name, one or more user IDs, one or more passwords, and other information that the data source requires.
- You can establish a connection by using a partial connection string or no additional information; in this case, HCL OneDB™ ODBC Driver can prompt the user for connection information.

After a connection is established, SQLDriverConnect connection string is completed. The application can use this string for subsequent connection requests.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
01S00	-11005	Invalid connection string attribute
08001	-11015	Unable to connect to data source
08002	-11016	Connection in use
08S01	-11020	Communication-link failure
28000	-11033	Invalid authorization specification
IM002	-11041	Data source not found and no default driver specified
IM003	-11042	Specified driver could not be loaded
IM004	-11043	Driver's SQLAllocEnv failed
IM005	-11044	Driver's SQLAllocConnect failed
IM006	-11045	Driver's SQLSetConnectOption failed
IM007	-11046	No data source or driver specified; dialog prohibited
IM008	-11047	Dialog failed
IM009	-11048	Unable to load translation shared library
IM010	-11049	Data-source name too long
IM011	-11050	Driver name too long
IM012	-11051	DRIVER keyword syntax error
S1000	-11060	General error
S1001	-11061	Memory-allocation failure

SQLSTATE	Error value	Error message
S1090	-11071	Invalid string or buffer length
S1110	-11090	Invalid driver completion
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11302	Insufficient connection information was supplied
S1000	-11303	Input connection string too large
S1000	-11317	Invalid connectdatabase value specified
S1000	-11318	Invalid vmbcharlenexact value specified
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLDrivers (level two only)

SQLDrivers lists driver descriptions and driver-attribute keywords.

The following table describes the SQLSTATE and error values for SQLDrivers.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1090	-11071	Invalid string or buffer length
S1103	-11083	Direction option out of range

## SQLError (core level only)

SQLError returns error or status information.

SQLError does not post error values for itself. SQLError returns SQL\_NO\_DATA\_FOUND when it cannot retrieve any error information (in which case *sqlstate* equals 00000). If SQLError cannot access error values for any reason that would normally return SQL\_ERROR, SQLError returns SQL\_ERROR but does not post any error values. If the buffer for the error message is too short, SQLError returns SQL\_SUCCESS\_WITH\_INFO but still does not return an SQLSTATE value for SQLError.

To determine that a truncation occurred in the error message, an application can compare *cbErrorMsgMax* to the actual length of the message text written to *pcbErrorMsg*.

## SQLExecDirect (core level only)

SQLExecDirect executes a preparable statement by using the current values of the parameter-marker variables if any parameters exist in the statement.

SQLExecDirect is the fastest way to submit an SQL statement for one-time execution.

The following table describes the SQLSTATE and error values for SQLExecDirect.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
01006	-11004	Privilege not revoked
01S03	-11008	No rows updated or deleted
01S04	-11009	More than one row updated or deleted
07001	-11012	Wrong number of parameters
07S01	-11014	Invalid use of default parameter
08S01	-11020	Communication-link failure
21S01	-11021	Insert value list does not match column list
21S02	-11022	Degree of derived table does not match column list
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
23000	-11030	Integrity-constraint violation
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
37000	-11035	Syntax error or access violation
40001	-11037	Serialization failure
42000	-11038	Syntax error or access violation
S0001	-11053	Base table or view already exists
S0002	-11054	Base table not found
S0011	-11055	Index already exists

SQLSTATE	Error value	Error message
S0012	-11056	Index not found
S0021	-11057	Column already exists
S0022	-11058	Column not found
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLExecute (core level only)

SQLExecute executes a prepared statement by using the current values of the parameter-marker variables if any parameter markers exist in the statement.

The following table describes the SQLSTATE and error values for SQLExecute.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
01006	-11004	Privilege not revoked
01S03	-11008	No rows updated or deleted
01S04	-11009	More than one row updated or deleted
07001	-11012	Wrong number of parameters

SQLSTATE	Error value	Error message
07S01	-11014	Invalid use of default parameter.
08S01	-11020	Communication-link failure
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
23000	-11030	Integrity constraint violation
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
42000	-11038	Syntax error or access violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

SQLExecute can return any SQLSTATE that SQLPrepare can return based on when the data source evaluates the SQL statement associated with the *hstmt*.

## SQLExtendedFetch (level two only)

SQLExtendedFetch extends the functionality of SQLFetch.

SQLExtendedFetch extends functionality in the following ways:



- It returns row-set data (one or more rows), in the form of an array, for each bound column.
- It scrolls through the result set according to the setting of a scroll-type argument.

SQLExtendedFetch works with SQLSetStmtOption.

To fetch one row of data at a time in a forward direction, an application calls SQLFetch.

The following table describes the SQLSTATE and error values for SQLExtendedFetch.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
01S01	-11006	Error in row
07006	-11013	Restricted data type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1106	-11086	Fetch type out of range
S1107	-11087	Row value out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

SQLSTATE	Error value	Error message
S1000	-11307	In <code>SQLExtendedFetch</code> , only <code>SQL_FETCH_NEXT</code> is supported for <code>SQL_SCROLL_Forward_only</code> cursors

If an error occurs that pertains to the entire row set, such as SQLSTATE S1T00 (Time-out expired), the driver returns `SQL_ERROR` and the appropriate SQLSTATE. The contents of the row set buffers are undefined, and the cursor position is unchanged.

If an error occurs that pertains to a single row, the driver performs the following actions:

- Sets the element in the `rgfRowStatus` array for the row to `SQL_ROW_ERROR`
- Posts SQLSTATE 01S01 (Error in row) in the error queue
- Posts zero or more additional SQLSTATE values for the error after SQLSTATE 01S01 (Error in row) in the error queue

After the driver processes the error or warning, it continues the operation for the remaining rows in the row set and returns `SQL_SUCCESS_WITH_INFO`. Thus, for each error that pertains to a single row, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.

After the driver processes the error, it fetches the remaining rows in the row set and returns `SQL_SUCCESS_WITH_INFO`. Thus, for each row that returns an error, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATE values.

If the row set contains rows that are already fetched, the driver is not required to return SQLSTATE values for errors that occurred when the rows were first fetched. However, it is required to return SQLSTATE 01S01 (Error in row) for each row in which an error originally occurred and to return `SQL_SUCCESS_WITH_INFO`. For example, a static cursor that maintains a cache might cache row-status information (so that it can determine which rows contain errors) but might not cache the SQLSTATE associated with those errors.

Error rows do not affect relative cursor movements. For example, suppose the result set size is 100, and the row-set size is 10. If the current row set is rows 11 through 20 and the element in the `rgfRowStatus` array for row 11 is `SQL_ROW_ERROR`, calling `SQLExtendedFetch` with the `SQL_FETCH_NEXT` fetch type still returns rows 21 through 30.

If the driver returns any warnings, such as SQLSTATE 01004 (Data truncated), it returns warnings that apply to the entire row set or to unknown rows in the row set before it returns error information that applies to specific rows. It returns warnings for specific rows with any other error information about those rows.

## SQLFetch (core level only)

`SQLFetch` fetches a row of data from a result set.

The driver returns data for all columns that were bound to storage locations with `SQLBindCol`.

The following table describes the SQLSTATE and error values for `SQLFetch`.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
07006	-11013	Restricted data-type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
40001	-11037	Serialization failure
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired

## SQLForeignKeys (level two only)

SQLForeignKeys can return a list of foreign keys.

SQLForeignKeys can return either of the following items:

- A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables)
- A list of foreign keys in other tables that refer to the primary key in the specified table

The driver returns each list as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for SQLForeignKeys.

SQLSTATE	Error value	Error message
01000	-11001	General warning

SQLSTATE	Error value	Error message
08S01	-11020	Communication link failure
24000	-11031	Invalid cursor state
IM001	-11040	Driver does not support this function
S1000	-11060	General error
S1001	-11061	Memory allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Timeout expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current@ connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

### SQLFreeConnect (core level only)

SQLFreeConnect releases a connection handle and frees all memory associated with the handle.

The following table describes the SQLSTATE and error values for SQLFreeConnect.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
S1000	-11060	General error
S1010	-11067	Function-sequence error

### SQLFreeEnv (core level only)

SQLFreeEnv frees the environment handle and releases all memory associated with the environment handle.

The following table describes the SQLSTATE and error values for SQLFreeEnv.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1010	-11067	Function-sequence error

### SQLFreeStmt (core level only)

SQLFreeStmt stops the processing that is associated with a specific *hstmt*, closes any open cursors that are associated with the *hstmt*, discards pending results, and, optionally, frees all resources associated with the statement handle.

The following table describes the SQLSTATE and error values for SQLFreeStmt.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1092	-11073	Option type out of range

### SQLGetConnectOption (level one only)

SQLGetConnectOption returns the current setting of a connection option.

The following table describes the SQLSTATE and error values for SQLGetConnectOption.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable

### SQLGetCursorName (core level only)

SQLGetCursorName returns the cursor name associated with a specified *hstmt*.

The following table describes the SQLSTATE and error values for SQLGetCursorName.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1015	-11070	No cursor name available
S1090	-11071	Invalid string or buffer length

## SQLGetData (level one only)

SQLGetData returns result data for a single unbound column in the current row.

The application must call SQLFetch or SQLExtendedFetch and (optionally) SQLSetPos to position the cursor on a row of data before it calls SQLGetData. It is possible to use SQLBindCol for some columns and use SQLGetData for others within the same row. This function can be used to retrieve character or binary data values in parts from a column with a character, binary, or data source-specific data type (for example, data from SQL\_LONGVARBINARY or SQL\_LONGVARCHAR columns).

The following table describes the SQLSTATE and error values for SQLGetData.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
07006	-11013	Restricted data- type attribute violation
08S01	-11020	Communication-link failure
22002	-11024	Indicator value required but not supplied
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime-field overflow
22012	-11028	Division by zero
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1002	-11062	Invalid column number

SQLSTATE	Error value	Error message
S1003	-11063	Program type out of range
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

## SQLGetFunctions (level one only)

SQLGetFunctions returns information about whether the driver supports a specific function.

The following table describes the SQLSTATE and error values for SQLGetFunctions.

SQLSTATE	Error value	Error message
01000	-1101	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1095	-11076	Function type out of range

## SQLGetInfo (level one only)

SQLGetInfo returns general information about the driver and data source associated with an *hdbc*.

The following table describes the SQLSTATE and error values for SQLGetInfo.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
08003	-11017	Connection not open
22003	-11025	Numeric value out of range
S1000	-11060	General error

SQLSTATE	Error value	Error message
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1090	-11071	Invalid string or buffer length
S1096	-11077	Information type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

### SQLGetStmtOption (level one only)

SQLGetStmtOption returns the current setting of a statement option.

The following table describes the SQLSTATE and error values for SQLGetStmtOption.

SQLSTATE	Error value	Error message
01000	-11001	General warning
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1109	-11089	Invalid cursor position
S1C00	-11092	Driver not capable

### SQLGetTypeInfo (level one only)

SQLGetTypeInfo returns information about data types that the data source supports.

The driver returns the information in the form of an SQL result set.

The following table describes the SQLSTATE and error values for SQLGetTypeInfo.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure



SQLSTATE	Error value	Error message
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1004	-11064	SQL data type out of range
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11305	SQLGetTypeInfo supported for FORWARD_ONLY cursors
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLMoreResults (level two only)

SQLMoreResults determines whether more results are available on an *hstmt* that contains SELECT, UPDATE, INSERT, or DELETE statements and, if so, initializes processing for those results.

The following table describes the SQLSTATE and error values for SQLMoreResults.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.

## SQLNativeSql (level two only)

SQLNativeSql returns the SQL string that the driver translates.

The following table describes the SQLSTATE and error values for SQLNativeSql.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
08003	-11017	Connection not open
37000	-11035	Syntax error or access violation
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1090	-11071	Invalid string or buffer length
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## Usage

The following example shows what SQLNativeSql might return for an input SQL string that contains the scalar function LENGTH:

```
SELECT {fn LENGTH(NAME)} FROM EMPLOYEE
```

HCL OneDB™ might return the following translated SQL string:

```
SELECT length(NAME) FROM EMPLOYEE
```

## SQLNumParams (level two only)

SQLNumParams returns the number of parameters in an SQL statement.

The following table describes the SQLSTATE and error values for SQLNumParams.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired

## SQLNumResultCols (core level only)

SQLNumResultCols returns the number of columns in a result set.

The following table describes the SQLSTATE and error values for SQLNumResultCols.

SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired

SQLNumResultCols can return any SQLSTATE that SQLPrepare or SQLExecute can return when SQLNumResultCols is called after SQLPrepare and before SQLExecute is called, depending on when the data source evaluates the SQL statement associated with the *hstmt*.

## SQLParamData (level one only)

SQLParamData is used with SQLPutData to supply parameter data when a statement executes.

The following table describes the SQLSTATE and error values for SQLParamData.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
22026	-11029	String data, length mismatch
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.

If `SQLParamData` is called while sending data for a parameter in an SQL statement, it can return any `SQLSTATE` that can be returned by the function that was called to execute the statement (`SQLExecute` or `SQLExecDirect`). If it is called while sending data for a column being updated or added with `SQLSetPos`, it can return any `SQLSTATE` that can be returned by `SQLSetPos`.

## SQLParamOptions (core and level two only)

`SQLParamOptions` allows an application to specify multiple values for the set of parameters assigned by `SQLBindParameter`.

The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. For example, an application can specify three sets of values for the set of parameters associated with an `INSERT` statement, and then execute the `INSERT` statement once to perform the three insert operations.

The following table lists the `SQLSTATE` values commonly returned by `SQLParamOptions` and explains each one in the context of this function; the notation `(DM)` precedes the description of each `SQLSTATE` returned by the driver manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR` unless noted otherwise.

<b>SQLSTATE</b>	<b>Error value</b>	<b>Error message</b>
01000		General warning
S1000		General error
S1001		Memory-allocation failure
S1010		Function-sequence error
S1107		Row value out of range

## SQLPrepare

`SQLPrepare` prepares an SQL string for execution.

The following table describes the `SQLSTATE` and error values for `SQLPrepare`.

<b>SQLSTATE</b>	<b>Error value</b>	<b>Error message</b>
01000	-11001	General warning
08S01	-11020	Communication-link failure
21S01	-11021	Insert value list does not match column list
21S02	-11022	Degree of derived table does not match column list
22005	-11026	Error in assignment
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
37000	-11035	Syntax error or access violation

SQLSTATE	Error value	Error message
42000	-11038	Syntax error or access violation
S0001	-11053	Base table or view already exists
S0002	-11054	Base table not found
S0011	-11055	Index already exists
S0012	-11056	Index not found
S0021	-11057	Column already exists
S0022	-11058	Column not found
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLPrimaryKeys (level two only)

SQLPrimaryKeys returns the column names that comprise the primary key for a table.

The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.

The following table describes the SQLSTATE and error values for SQLPrimaryKeys.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state

SQLSTATE	Error value	Error message
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLProcedureColumns (level two only)

SQLProcedureColumns returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.

The driver returns the information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for SQLProcedureColumns.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication link failure
24000	-11031	Invalid cursor state
IM001	-11040	Driver does not support this function
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable

SQLSTATE	Error value	Error message
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLProcedures (level two only)

SQLProcedures returns the list of procedure names stored in a specific data source.

*Procedure* is a generic term used to describe an *executable object*, or a named entity that can be started with input and output parameters, and which can return result sets similar to the results that SELECT statements return.

The following table describes the SQLSTATE and error values for SQLProcedures.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLPutData (level one only)

SQLPutData allows an application to send data for a parameter or column to the driver at statement execution time.

This function can send character or binary data values in parts to a column with a character, binary, or data-source-specific data type (for example, parameters of SQL\_LONGVARBINARY or SQL\_LONGVARCHAR).

The following table describes the SQLSTATE and error values for SQLPutData.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01004	-11003	Data truncated
07S01	-11014	Invalid use of default parameter
08S01	-11020	Communication-link failure
22001	-11023	String data right truncation
22003	-11025	Numeric value out of range
22005	-11026	Error in assignment
22008	-11027	Datetime-field overflow
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1T00	-11094	Time-out expired



**Important:** An application can use SQLPutData to send sections of character C data to a column with a character, binary, or data source-specific data type or to send binary C data to a column with a character, binary, or data source-specific data type. If SQLPutData is called more than once under any other conditions, it returns SQL\_ERROR and SQLSTATE 22003 (Numeric value out of range).

## SQLRowCount (core level only)

SQLRowCount returns the number of rows affected by an UPDATE, INSERT, or DELETE statement or by an SQL\_UPDATE, SQL\_ADD, or SQL\_DELETE operation in SQLSetPos.

The following table describes the SQLSTATE and error values for SQLRowCount.



SQLSTATE	Error value	Error message
01000	-11001	General warning
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error

## SQLSetConnectOption (level one only)

SQLSetConnectOption sets options that govern aspects of connections.

The following table describes the SQLSTATE and error values for SQLSetConnectOption.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01S02	-11007	Option value changed
08002	-11016	Connection in use
08003	-11017	Connection not open
08S01	-11020	Communication-link failure
IM009	-11048	Unable to load translation shared library (DLL)
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

When *fOption* is a statement option, SQLSetConnectOption can return any SQLSTATE that SQLSetStmtOption returns.

## SQLSetCursorName (core level only)

SQLSetCursorName associates a cursor name with an active *hstmt*.

If an application does not call `SQLSetCursorName`, the driver generates cursor names as needed for SQL statement processing.

The following table describes the SQLSTATE and error values for `SQLSetCursorName`.

SQLSTATE	Error value	Error message
01000	-11001	General warning
24000	-11031	Invalid cursor state
34000	-11034	Invalid cursor name
3C000	-11036	Duplicate cursor name
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length

## SQLSetStmtOption (level one only)

`SQLSetStmtOption` sets options that are related to an *hstmt*.

To set an option for all the statements associated with a specific *hdbc*, an application can call `SQLSetConnectOption`.

The following table describes the SQLSTATE and error values for `SQLSetStmtOption`.

SQLSTATE	Error value	Error message
01000	-11001	General warning
01S02	-11007	Option value changed
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1009	-11066	Invalid argument value
S1010	-11067	Function-sequence error
S1011	-11068	Operation invalid at this time
S1092	-11073	Option type out of range
S1C00	-11092	Driver not capable

## SQLSpecialColumns (level one only)

SQLSpecialColumns retrieves information about columns.

SQLSpecialColumns retrieves the following information about columns within a specified table:

- The optimal set of columns that uniquely identifies a row in the table
- Columns that are automatically updated when any value in the row is updated by a transaction

The following table describes the SQLSTATE and error values for SQLSpecialColumns.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1097	-11078	Column type out of range
S1098	-11079	Scope type out of range
S1099	-11080	Nullable type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLStatistics (level one only)

SQLStatistics retrieves a list of statistics about a single table and the indexes associated with the table.

The driver returns this information as a result set.

The following table describes the SQLSTATE and error values for SQLStatistics.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory- allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1100	-11081	Uniqueness option type out of range
S1101	-11082	Accuracy option type out of range
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

### SQLTablePrivileges (level two only)

SQLTablePrivileges returns a list of tables and the privileges associated with each table.

The driver returns the information as a result set on the specified *hstmt*.

The following table describes the SQLSTATE and error values for SQLTablePrivileges.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error

SQLSTATE	Error value	Error message
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLTables (level one only)

SQLTables returns the list of table names that are stored in a specific data source.

The driver returns this information as a result set.

The following table describes the SQLSTATE and error values for SQLTables.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08S01	-11020	Communication-link failure
24000	-11031	Invalid cursor state
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1008	-11065	Operation canceled
S1010	-11067	Function-sequence error
S1090	-11071	Invalid string or buffer length
S1C00	-11092	Driver not capable
S1T00	-11094	Time-out expired
S1C00	-11300	SQL_DEFAULT_PARAM not supported

SQLSTATE	Error value	Error message
08S01	-11301	A protocol error has been detected. Current connection is closed.
S1000	-11310	Create and Drop must be executed within a ServerOnly Connection
S1000	-11320	Syntax error
S1000	-11323	The statement contained an escape clause not supported by this database driver

## SQLTransact (core level only)

SQLTransact requests a commit or rollback operation for all active operations on all *hstmts* associated with a connection.

SQLTransact can also request that a commit or rollback operation is performed for all connections associated with the *henv*.

The following table describes the SQLSTATE and error values for SQLTransact.

SQLSTATE	Error value	Error message
01000	-11001	General warning
08003	-11017	Connection not open
S1000	-11060	General error
S1001	-11061	Memory-allocation failure
S1010	-11067	Function-sequence error
S1012	-11069	Invalid transaction operation code specified
S1C00	-11092	Driver not capable
08S01	-11301	A protocol error has been detected. Current connection is closed.

## Unicode

These topics provide a brief overview of the Unicode standard and shows how it is used within ODBC applications.

### Overview of Unicode

*Unicode* is a character encoding standard that provides a means of representing each character used in every major language.

In the Unicode standard, each character is assigned a unique numeric value and name. These values can be used consistently between applications across multiple platforms.

### Unicode versions

Although Unicode provides a consistent way of representing text across multiple languages, there are different versions which provide different data sizes for each character.

The following list describes the versions that are supported within HCL OneDB™ ODBC applications.

### UCS-2

ISO encoding standard that maps Unicode characters to 2 bytes each. UCS-2 is the common encoding standard on Windows™.

HCL OneDB™ ODBC Driver for IBM AIX® platforms supports UCS-2 encoding. HCL OneDB™ ODBC Driver for Windows™ supports only UCS-2.

### UCS-4

ISO encoding standard that maps Unicode characters into 4 bytes each.

The HCL OneDB™ ODBC Driver supports UCS-4 on UNIX™ platforms.

### UTF-8

Encoding standard that is based on a single (8 bit) byte. UTF-8 defines a mechanism to transform all Unicode characters into a variable length (1 - 4) encoding of bytes.

The HCL OneDB™ ODBC Driver uses UTF-8 encoding for all UNIX™ applications that connect to the Data Direct (formerly Merant) driver manager.

The 7-bit ASCII characters have the same encoding under both ASCII and UTF-8. This has the advantage that UTF-8 can be used with much existing software without extensive revision.

**!** **Important:** In applications that use Unicode, the driver does the work of code set conversion from Unicode to the database locale and vice versa. The UTF-8 is the only type of Unicode code set that can be set as the client locale.

## Unicode in an ODBC application

View the typical ODBC application architecture.

The following diagram shows the architecture of a typical ODBC application with a driver manager and the HCL OneDB™ ODBC Driver.

Figure 8. Typical ODBC application architecture



In this scenario, if an application calls to Unicode enabled APIs, then it must be connected to a Unicode enabled HCL OneDB™ ODBC Driver (Version 3.8 and later) to ensure that there is no loss of data. If the application calls to ANSI ODBC APIs, the application can be linked to either a Unicode enabled driver or an ANSI driver.

The HCL OneDB™ ODBC Driver continues to support HCL® OneDB® GLS. Hence all data fetched in character buffers are fetched in the client locale code set. Only data fetched with wide character buffers use Unicode.

On Windows™, if the ODBC driver is not Unicode enabled, the ODBC Driver Manager maps all Unicode API function calls to ANSI ODBC APIs.

If the ODBC driver is Unicode enabled, the Windows™ ODBC Driver Manager (Version 4.10 or later) maps all ANSI ODBC APIs to Unicode ODBC APIs. The Data Direct (formerly Merant) driver manager for UNIX™ also works this way.

**!** **Important:** In CSDK Version 2.70 there are two ODBC drivers. One with only ANSI APIs (called ANSI ODBC Driver, Version 3.34) and another with both ANSI and UNICODE APIs (called Unicode ODBC Driver, Version 3.80). For CSDK 2.80 and later, there is only one ODBC driver that supports both ANSI and UNICODE APIs.

**!** **Important:** The HCL® OneDB® Driver Manager Replacement (DMR) for UNIX™ platforms does not map between Unicode and ANSI APIs.

For details about how the Windows™ ODBC driver manager handles mapping, see the section "Function Mapping in the Driver Manager" in the *ODBC Programmer's Reference for Microsoft™*.

## Unicode in an ODBC application

This section provides details on compiling and configuring Unicode within HCL® OneDB® ODBC applications.

### Configuration

Since the HCL OneDB™ ODBC Driver supports different types of Unicode on UNIX™ platforms, the type of Unicode used by an application must be indicated in the ODBC section of the `odbc.ini` file.

Indicate the type of Unicode in the ODBC section as follows:

```
[ODBC]
.
.
.
UNICODE=UCS-4
```

**!** **Important:** A Unicode-enabled application must indicate the type of Unicode used in the `odbc.ini` file. If the Unicode parameter is not set in `odbc.ini`, the default type is UCS-4.

It is required that all UNIX™ ODBC applications must set the Unicode type in the `odbc.ini` file as follows:

- An ANSI ODBC application on UNIX™ (including AIX® 64-bit) must set `UNICODE=UCS-4`
- An ANSI ODBC application on IBM AIX® 32-bit must set `UNICODE-UCS-2`
- An ANSI ODBC application that uses the Data Direct (formerly Merant) ODBC driver manager never indicates a Unicode type other than UTF-8 in the `odbc.ini` file.

The following table provides an overview of the `odbc.ini` settings:



Platform	Driver manager	odbc.ini setting
AIX®	Data Direct	UTF-8
AIX® 32-bit	DMR or none	UCS-2
AIX® 64-bit	Data Direct	UTF-8
UNIX™	Data Direct	UTF-8
UNIX™	DMR or none	UCS-4
Windows™	Windows™ ODBC Driver Manager	N/A



### Important:

If all of the following conditions exist, the settings are automatically reset without any warning or error message:

- The application is an ANSI application.
- You are linking with DMR or none.
- The Unicode setting in the odbc.ini file does not match the values shown in the table.

## Supported Unicode functions

The HCL OneDB™ ODBC Driver supports both ANSI and Unicode version of all functions that accept pointer to character strings or SQLPOINTER in their arguments.

The following list describes the two types of functions supported:

### ODBC "A" functions

The normal ODBC functions that accept single byte (ASCII) data as input for all the character/string parameters.

### ODBC "W" functions

The Unicode functions that accept "wide characters" as input for all character/string parameters.

The ODBC specification defines these functions with the wchar\_t data type. This data type is the standard C library-wide character data type.

The following Unicode "wide" functions are supported by the HCL OneDB™ ODBC Driver:

- SQLColAttributeW
- SQLColAttributesW
- SQLConnectW
- SQLDescribeColW
- SQLErrorW
- SQLExecDirectW

- SQLGetConnectAttrW
- SQLGetCursorNameW
- SQLSetDescFieldW
- SQLGetDescFieldW
- SQLGetDescRecW
- SQLGetDiagFieldW
- SQLGetDiagRecW
- SQLPrepareW
- SQLSetConnectAttrW
- SQLSetCursorNameW
- SQLColumnsW
- SQLGetConnectOptionW
- SQLGetTypeInfoW
- SQLSetConnectOptionW
- SQLSpecialColumnsW
- SQLStatisticsW
- SQLTablesW
- SQLDataSourcesW
- SQLDriverConnectW
- SQLBrowseConnectW
- SQLColumnPrivilegesW
- SQLGetStmtAttrW
- SQLSetStmtAttrW
- SQLForeignKeysW
- SQLNativeSqlW
- SQLPrimaryKeysW
- SQLProcedureColumnsW
- SQLProceduresW
- SQLTablePrivilegesW
- SQLDriversW

As of Version 4.10, the SQLGetDiagRecW function *BufferLength* argument is defined as: Length of the *MessageText* buffer in characters.

## ODBC Smart trigger

Smart Triggers (also known as Pushdata) in ODBC are a set of classes/interfaces that provide an ease of use capability to the Push data feature.

A smart trigger is a set of commands issued to the database that sets up a push notification when certain changes happen to data in a table. These changes are detected by a SQL query that is run after INSERT, UPDATE, or DELETE commands are executed. It is available across all CSDK/ODBC supported platforms.

It uses ODBC's standard APIs `SQLSetStmtAttr()/SQLSetStmtAttrA()/SQLSetStmtAttrW()` and `SQLGetStmtAttr()/SQLGetStmtAttrA()/SQLGetStmtAttrW()` with following Informix extensions, defined in `infxcli.h` file.

```
SQL_INFX_ATTR_OPEN_SMART_TRIGGER
SQL_INFX_ATTR_JOIN_SMART_TRIGGER
SQL_INFX_ATTR_GET_LO_FILE_DESC_SMART_TRIGGER
SQL_INFX_ATTR_GET_SESSION_ID_SMART_TRIGGER
SQL_INFX_ATTR_REGISTER_SMART_TRIGGER
SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_LOOP
SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_NO_LOOP
SQL_INFX_ATTR_DELETE_SMART_TRIGGER
```

## ODBC API

Using the following ODBC API to use Smart Trigger:

1. Allocate Environment handle
2. Allocate Connection Handle
3. Connect to "sysadmin? database
4. Allocate statement handle
5. Call `SQLSetStmtAttr(SQL_INFX_ATTR_OPEN_SMART_TRIGGER)` and using structure `IFMX_OPEN_SMART_TRIGGER`
6. Call `SQLGetStmtAttr(SQL_INFX_ATTR_GET_LO_FILE_DESC_SMART_TRIGGER)` and get the File Descriptor ID (to be used for registering the event/queries, the same File Descriptor to be used for multiple event/queries)
7. Following steps could be in thread loop for each event/query to be registered.
  - a. `SQLAllocHandle(STMT)`
  - b. Fill/Populate the `SQL_INFX_ATTR_REGISTER_SMART_TRIGGER` structure
  - c. Call `SQLSetStmtAttr(SQL_INFX_ATTR_REGISTER_SMART_TRIGGER)`
  - d. Call `SQLGetStmtAttr(SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_LOOP / SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_NO_LOOP)`

Once you register the event or join the already registered session, the call becomes blocking (ODBC waits for IDS server to return the message/data), once the data/response or timeout message is received from IDS server, ODBC invokes the registered callback function with output buffer.



**Note:** It is read only buffer and application should not tamper the buffer.

Once the control comes back to application (callback function), application may decide to come out of the loop by setting `IFMX_JOIN_SMART_TRIGGER->ControlBackToApplication` OR `SQL_INFX_ATTR_REGISTER_SMART_TRIGGER->isDeregister` to `TRUE`. If you decide to continue, then there is no action needed by you in the callback function (other than consuming the received output).

## Register the smart trigger events

File Descriptor is required to "register the events". After successful call to `SQLSetStmtAttr(SQL_INFX_ATTR_OPEN_SMART_TRIGGER)`, application should call following `SQLGetStmtAttr()` API to get the "File Descriptor?". The same "File Descriptor?" should be used to "Register the smart trigger/pushdata events?".

```
SQLGetStmtAttr(hstmt, SQL_INFX_ATTR_GET_LO_FILE_DESC_SMART_TRIGGER, (int *)&FileDesc, SQL_NTS, NULL);
```

Application should use "File Descriptor" received from above SQLGetStmtAttr() call and other inputs like table, database, user, query etc to populate/fill the structure (mentioned above) IFMX\_REGISTER\_SMART\_TRIGGER. Application should make call to SQLSetStmtAttr() as follows. Application can register as many as events/queries they want in each thread (application example below). It is advised to use separate statement handle for each registration.

```
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &tmpHstmt);
SQLSetStmtAttr(tmpHstmt, SQL_INFX_ATTR_REGISTER_SMART_TRIGGER, (void *)&SmartTriggerInstance, SQL_NTS);
```

If the session was opened with detachable option then session ID will be created. Application can save the same session ID and could use the same later to attach/join to the session. Below API should be used to attach/join the session using structure IFMX\_JOIN\_SMART\_TRIGGER. All the registered events/queries with the attached/joined session will be in effect.

```
SQLSetStmtAttr(hstmt, SQL_INFX_ATTR_JOIN_SMART_TRIGGER, (void *)&gJoinSmartTrigger, SQL_NTS);
```

## Delete the registered session

If the session was registered as detachable session, there should be "session ID". The same session ID should be used to delete the session by making below call.

```
SQLSetStmtAttr(hstmt, SQL_INFX_ATTR_DELETE_SMART_TRIGGER, (int *)&sesID, SQL_NTS);
```

## API Structure

- **IFMX\_OPEN\_SMART\_TRIGGER structure:** This is input structure to be used in SQLSetStmtAttr() call while establishing the smart trigger/pushdata session. The allocation & deallocation of this structure and its members is application's responsibility. Each member of the structure is explained below:

**BOOL \*isDetachable ->** If the session is expected to be reused at later time, even after finishing the application, one should set this flag to TRUE. Otherwise set to FALSE.

**int \*timeOut->** Time in seconds, as per registered event/query, if no event happens, IDS sever will send "timeout" message to ODBC based on the value set for this member. Valid range is 0 to 12000 seconds. If it's beyond these value, internally it will be set to 300 seconds.

**short \*maxRecsPerRead->**This is number of records, returned (in callback function output buffer). Default is 1. Valid allowed range is 1 to 200. The max buffer size is 8KB, if this number is set high, it will return data only max of 8KB at a time, which may not match the number of records.

**int \*maxPendingOperations->**Maximum pending operation. Default is 0. Allowed range is 0 to 200.

**SQLWCHAR reserved[16]->**Reserved for future usage.

- **IFMX\_JOIN\_SMART\_TRIGGER structure:** This structure is used to attach/join already opened session. Each member of the structure is explained below:

**void(\*callback) (char const \*jsonOutBuf)->** This is user defined callback function. This will be called when response from IDS server is received on registered event/query(events/queries) for a session to be attached/joined. The data returned is read only for the user.

**int \*joinSessionID**-> This is to be assigned from the value received from the SQLGetStmtAttr() call. This session ID will be used to connect to prior registered session.

**BOOL \*ControlBackToApplication**-> Once control comes back to application (as part of callback call), if user decides to come out of the blocking call (event registration is blocking call), user can set this flag to TRUE.

**SQLWCHAR reserved[16]** -> Reserved for future usage.

- **IFMX\_REGISTER\_SMART\_TRIGGER structure:** This structure is used to register the event/query on already opened smart trigger/pushdata session. There could be as many as trigger/event user wants to register. Each trigger/event should be invoked with separate thread. Each member of the structure is explained below.

**void(\*callback) (char const \*jsonOutBuf)**-> This is user defined callback function. This will be called when response from IDS server is received on registered event/query. The data returned is read only for the user.

**int \*loFileDescriptor** -> This is input, which is received from SQLGetStmtAttr(SQL\_INFX\_ATTR\_GET\_LO\_FILE\_DESC\_SMART\_TRIGGER) call. For each event/query registration for a given session, this value will remain same.

**SQLWCHAR \*tableName** -> This is table name to be registered. The input to be provided in SQLWCHAR type. Max length as per IDS supported table length.

**SQLWCHAR \*ownerName** -> This is user/owner of table to be registered. The input to be provided in SQLWCHAR type. Max length as per IDS supported user/owner length.

**SQLWCHAR \*dbName** -> This is database name where the table belongs. The input to be provided in SQLWCHAR type. Max length as per IDS supported database length.

**SQLWCHAR \*sqlQuery** -> This is the SELECT query which is on registered table. The input to be provided in SQLWCHAR type. Max length of query 4KB.

**SQLWCHAR \*label** -> If NULL, internally, ODBC will create the label using table name, owner name database name & internal counter i.e. "%s\_%s\_%s\_%d". The input to be provided in SQLWCHAR type.

**BOOL \*isDeregister** -> Smart Trigger/Pushdata is blocking call, the control is back to application when ODBC calls user specified callback function. If user wants to come out of the blocking call, they can set this flag to TRUE.

**BOOL \*ControlBackToApplication**-> Once control comes back to application (as part of callback call), if user decides to come out of the blocking call without non-registering the event (event registration is blocking call), user can set this flag to TRUE.

**SQLWCHAR reserved[16]** -> Reserved for future usage.

## Compiling sample application on Linux

Use below compilation/linking steps, assuming C file name is SmartTrigger.c and application links directly to Informix driver (no Driver Manager).

```
gcc -g -c -fsigned-char -DNO_WIN32 -O -I$ONEDB_HOME/incl/cli -I$ONEDB_HOME/incl/esql -I$ONEDB_HOME/incl/dmi
SmartTrigger.c
```

```
gcc -g -o SmartTrigger SmartTrigger.o -L$ONEDB_HOME/lib/cli -L$ONEDB_HOME/lib/esql -lthcli -lifdms
-L$ONEDB_HOME/lib/esql -lifgls -lifglx -lm -lnsl
```

## Compiling sample application on Windows

Use below steps to compile the Smart Trigger application on Windows.

```
cl /Zi /DEBUG /MD /D_CRT_SECURE_NO_DEPRECATED /D_CRT_NON_CONFORMING_SWPRINTFS /D
_CRT_NONSTDC_NO_DEPRECATED /I%ONEDB_HOME%\incl\cli %ONEDB_HOME%\lib\iclit09b.lib odbc32.lib odbccp32.lib
SmartTrigger.c
```

The below example, supports two(2) trigger/event registration, hence two threads (NUM\_OF\_INSTANCE) has been used. There are following 3 functions which needs to be changed to provide appropriate input for your environment.

1. SetConnectionString() => This is for database connection, you can provide "DSN=<value>?" as well, depending on your choice. In this function, you need to provide your own connection string.
2. AssignOpenParams() => In this function, you may need to change timeout, number of records(1 is recommended) etc parameters for smart trigger/pushdata session opening.
3. AssignRegisterParams() => In this function, you need to change, callback function, table, owner, database and query values which suits your environment. This function uses two such events/inputs to be registered.

## ODBC sample application

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef NO_WIN32
#include <io.h>
#include <windows.h>
#include <conio.h>
#endif /*NO_WIN32*/

#define __REENTRANT
#include <signal.h>
#ifdef NO_WIN32
#include <sys/wait.h>
#include <pthread.h>
#endif
#include <time.h>
#include <infxcli.h>

#define ERRMSG_LEN 200
#define NAMELEN 300
#define NUM_OF_INSTANCE 2

SQLHDBC hdbc;
SQLHENV henv;
SQLHSTMT hstmt;
```

```

SQLINTEGER sesID = 0;
SQLWCHAR connStrInW[NAMELEN];
SQLCHAR connStrIn[NAMELEN];
BOOL delete = 0;
int gFileDesc = 0;
int fileDesc[NUM_OF_INSTANCE];
int timeout;
short maxRec;
int maxPend;
int detachable;
BOOL dregister[NUM_OF_INSTANCE];
BOOL ControlBack[NUM_OF_INSTANCE];

IFMX_REGISTER_SMART_TRIGGER gSmartTriggerRegister[NUM_OF_INSTANCE];
IFMX_OPEN_SMART_TRIGGER gopenSmartTrigger;
IFMX_JOIN_SMART_TRIGGER gJoinSmartTrigger;

SQLINTEGER checkError (SQLRETURN rc,
 SQLSMALLINT handleType,
 SQLHANDLE handle,
 SQLCHAR* errMsg)
{
 SQLRETURN retcode = SQL_SUCCESS;
 SQLSMALLINT errNum = 1;
 SQLWCHAR sqlStateW[6];
 SQLCHAR *sqlState;
 SQLINTEGER nativeError;
 SQLWCHAR errMsgW[ERRMSG_LEN];
 SQLCHAR *errMsg;
 SQLSMALLINT textLengthPtr;

 if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO))
 {
 while (retcode != SQL_NO_DATA)
 {
 retcode = SQLGetDiagRecW (handleType, handle, errNum, sqlStateW, &nativeError, errMsgW, ERRMSG_LEN,
&textLengthPtr);

 if (retcode == SQL_INVALID_HANDLE)
 {
 fprintf (stderr, "checkError function was called with an invalid handle!!\n");
 return 1;
 }

 if ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO))
 {
 sqlState = (SQLCHAR *) malloc (wcslen(sqlStateW) + sizeof(char));
 wcstombs((char *) sqlState, sqlStateW, wcslen(sqlStateW)
 + sizeof(char));

 errMsg = (SQLCHAR *) malloc (wcslen(errMsgW) + sizeof(char));
 wcstombs((char *) errMsg, errMsgW, wcslen(errMsgW)
 + sizeof(char));

 fprintf (stderr, "ERROR: %d: %s : %s \n", nativeError, sqlState, errMsg);
 }
 }
 }
}

```

```

 }

 errNum++;
}

fprintf(stderr, "%s\n", errmsg);
return 1; /* all errors on this handle have been reported */
}
else
 return 0; /* no errors to report */
}

void TriggerCallback1(char const *outBuf)
{
 dregister[0] = FALSE; //TRUE;
 ControlBack[0] = FALSE;
 printf("\nCallback #1");
 if(outBuf != NULL)
 printf("\nData received : %s\n", outBuf);
 else
 printf("\nReturned NULL data!!");

 gSmartTriggerRegister[0].isDeregister = &dregister[0];
 gSmartTriggerRegister[0].ControlBackToApplication = &ControlBack[0];
 gJoinSmartTrigger.ControlBackToApplication = &ControlBack[0];
 return;
}

void TriggerCallback2(char const *outBuf)
{
 dregister[1] = FALSE;
 ControlBack[1] = FALSE; //TRUE;
 printf("\nCallback #2");
 if(outBuf != NULL)
 printf("\nData received : %s\n", outBuf);
 else
 printf("\nReturned NULL data!!");

 gSmartTriggerRegister[1].isDeregister = &dregister[1];
 gSmartTriggerRegister[1].ControlBackToApplication = &ControlBack[1];
 gJoinSmartTrigger.ControlBackToApplication = &ControlBack[1];
 return;
}

DWORD ThreadRegisterPushDataQuery(void *lpParam)
{
 SQLRETURN rc = 0;
 SQLHSTMT tmpHstmt;
 SQLINTEGER dummy = 0;

 IFMX_REGISTER_SMART_TRIGGER temp;
 IFMX_REGISTER_SMART_TRIGGER SmartTriggerInstance;

 SmartTriggerInstance = *((IFMX_REGISTER_SMART_TRIGGER *)lpParam);

 rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &tmpHstmt);

```



```

 if (checkError (rc, SQL_HANDLE_DBC, hdbc, (SQLCHAR *) "Error(Thread) in Step 1 -- Statement Handle
Allocation failed\nExiting!!"))
 exit(-1);

 rc = SQLSetStmtAttrW(tmpHstmt, SQL_INFX_ATTR_REGISTER_SMART_TRIGGER, (IFMX_REGISTER_SMART_TRIGGER
*)&SmartTriggerInstance, SQL_IS_POINTER);
 if (checkError(rc, SQL_HANDLE_DBC, hdbc, (SQLCHAR *) "Error(Thread) in Step 2 -- SQLSetStmtAttr
failed\nExiting!!"))
 exit(-1);
 rc = SQLGetStmtAttrW(tmpHstmt, SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_LOOP, (void *)&dummy, SQL_NTS, NULL);
 //rc = SQLGetStmtAttrW(tmpHstmt, SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_NO_LOOP, (void *)&dummy, SQL_NTS,
NULL);
 if (checkError(rc, SQL_HANDLE_DBC, hdbc, (SQLCHAR *) "Error(Thread) in Step 2 -- SQLGetStmtAttr
failed\nExiting!!"))
 exit(-1);

 SQLFreeHandle(SQL_HANDLE_STMT, tmpHstmt);
 printf("\nFinished thread execution\n");
}

void SetConnectionString()
{
#ifdef NO_WIN32
 //sprintf((char *) connStrIn, "DRIVER={IBM INFORMIX ODBC
DRIVER};HOST=x.x.x.x;SERVER=ol_informix1210_2;SERVICE=8573;PROTOCOL=onsoctcp;DATABASE=sysadmin;UID=onedb;PWD=x
xx");
 wsprintf((SQLWCHAR *) connStrInW, "DSN=SmartTrigger");
#else
 //sprintf((char *) connStrIn, "DRIVER={IBM INFORMIX ODBC
DRIVER};HOST=x.x.x.x;SERVER=ol_informix1210_1;SERVICE=20195;PROTOCOL=onsoctcp;DATABASE=sysadmin;UID=onedb;PWD=
xxx");
 wsprintf((SQLWCHAR *) connStrInW, L"DSN=SmartTrigger");
#endif
 return;
}

void AssignOpenParams()
{
 timeOut = 5; //In seconds
 maxRec = 1; //Maximum number of records to get, (8192 Bytes limit)
 maxPend = 0;
 detachable = FALSE; //TRUE, If want to retain session for later usage
 delete = FALSE; //TRUE, if want to delete the detachable session

 gopenSmartTrigger.timeOut = &timeOut;
 gopenSmartTrigger.isDetachable = &detachable;
 gopenSmartTrigger.maxRecsPerRead = &maxRec;
 gopenSmartTrigger.maxPendingOperations = &maxPend;

 return;
}

void AssignRegisterParams(IFMX_REGISTER_SMART_TRIGGER *SmartTriggerInstance, int FileDesc, int i)
{
 fileDesc[i] = FileDesc;
 dregister[i] = FALSE; // Just initialize, should be changed in callback
 ControlBack[i] = FALSE; // Just initialize, should be changed in callback
 SmartTriggerInstance->loFileDescriptor = &fileDesc[i];
}

```

```

SmartTriggerInstance->tableName = (SQLWCHAR *)malloc(50 * sizeof(SQLWCHAR));
SmartTriggerInstance->ownerName = (SQLWCHAR *)malloc(50 * sizeof(SQLWCHAR));
SmartTriggerInstance->dbName = (SQLWCHAR *)malloc(50 * sizeof(SQLWCHAR));
SmartTriggerInstance->sqlQuery = (SQLWCHAR *)malloc(500 * sizeof(SQLWCHAR));
//SmartTriggerInstance->label = (SQLWCHAR *)malloc(50 * sizeof(SQLWCHAR));
SmartTriggerInstance->label = NULL;
SmartTriggerInstance->isDeregister = &dregister[i];
SmartTriggerInstance->ControlBackToApplication = &ControlBack[i];

//wcscopy((SQLWCHAR *)SmartTriggerInstance->ownerName, L"shesh");
//wcscopy((SQLWCHAR *)SmartTriggerInstance->dbName, L"sheshdb");
wsprintf((SQLWCHAR *)SmartTriggerInstance->ownerName, "shesh");
wsprintf((SQLWCHAR *)SmartTriggerInstance->dbName, "sheshdb");

if (i==0)
{
 SmartTriggerInstance->callback = TriggerCallback1;
 //wcscopy((SQLWCHAR *)SmartTriggerInstance->label, L"label1");
 //wcscopy((SQLWCHAR *)SmartTriggerInstance->tableName, L"tab1");
 //wcscopy((SQLWCHAR *)SmartTriggerInstance->sqlQuery, L"select * from tab1;");
 //wsprintf((SQLWCHAR *)SmartTriggerInstance->label, "label1");
 wsprintf((SQLWCHAR *)SmartTriggerInstance->tableName, "tab1");
 wsprintf((SQLWCHAR *)SmartTriggerInstance->sqlQuery, "select * from tab1;");
}
else
{
 SmartTriggerInstance->callback = TriggerCallback2;
 //wcscopy((SQLWCHAR *)SmartTriggerInstance->label, L"label2");
 //wcscopy((SQLWCHAR *)SmartTriggerInstance->tableName, L"tab2");
 //wcscopy((SQLWCHAR *)SmartTriggerInstance->sqlQuery, L"select * from tab2;");
 //wsprintf((SQLWCHAR *)SmartTriggerInstance->label, "label2");
 wsprintf((SQLWCHAR *)SmartTriggerInstance->tableName, "tab2");
 wsprintf((SQLWCHAR *)SmartTriggerInstance->sqlQuery, "select * from tab2;");
}
return;
}

void FreeMemory(IFMX_REGISTER_SMART_TRIGGER *SmartTriggerInstance)
{
 free(SmartTriggerInstance->tableName);
 free(SmartTriggerInstance->ownerName);
 free(SmartTriggerInstance->dbName);
 free(SmartTriggerInstance->sqlQuery);
 if(SmartTriggerInstance->label != NULL)
 free(SmartTriggerInstance->label);
 return;
}

int main (long argc,
 char* argv[])
{
 /* Miscellaneous variables */
 SQLRETURN rc = 0;
 SQLINTEGER i = 0;
 SQLINTEGER getSesID = 0;
 SQLWCHAR connStrOutW[NAMELEN];
 SQLSMALLINT connStrOutLen;
 SQLINTEGER stackSize = 40 * 1024;

```

```

HANDLE hThread_[NUM_OF_INSTANCE];
DWORD threadID_[NUM_OF_INSTANCE];
#ifdef NO_WIN32
pthread_t cpid[NUM_OF_INSTANCE];
#endif
DWORD dwThreadID=10;

printf("\nApplication : sizeof(SQLWCHAR) = %d", sizeof(SQLWCHAR));
/* Allocate the Environment handle */
rc = SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (rc != SQL_SUCCESS)
{
 fprintf (stdout, "Environment Handle Allocation failed\nExiting!!");
 exit (-1);
}

/* Set the ODBC version to 3.0 */
rc = SQLSetEnvAttr (henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, 0);
if (checkError (rc, SQL_HANDLE_ENV, henv, (SQLCHAR *) "Error(main) in Step 1 -- SQLSetEnvAttr
failed\nExiting!!"))
 exit (-1);

/* Allocate the connection handle */
rc = SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
if (checkError (rc, SQL_HANDLE_ENV, henv, (SQLCHAR *) "Error(main) in Step 2 -- Connection Handle
Allocation failed\nExiting!!"))
 exit (-1);

/* Establish the database connection */
SetConnectionString();
rc = SQLDriverConnectW(hdbc, NULL, connStrInW, SQL_NTS, connStrOutW, NAMELEN, &connStrOutLen,
SQL_DRIVER_NOPROMPT);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, (SQLCHAR *) "Error(main) in Step 3 -- SQLDriverConnect
failed\nExiting!!"))
 exit (-1);
printf("\nApplication : Database connection successful");

/* Allocate the statement handle */
rc = SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
if (checkError (rc, SQL_HANDLE_DBC, hdbc, (SQLCHAR *) "Error(main) in Step 4 -- Statement Handle Allocation
failed\nExiting!!"))
 exit (-1);

sesID = 0;
gJoinSmartTrigger.callback = TriggerCallback1;
gJoinSmartTrigger.joinSessionID = &sesID;

if(sesID != 0)
{
 rc = SQLSetStmtAttrW(hstmt, SQL_INFX_ATTR_JOIN_SMART_TRIGGER, (void *)&gJoinSmartTrigger, SQL_NTS);
 if (checkError(rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error(main) in Step 5 --
SQLSetStmtAttr(SQL_INFX_ATTR_JOIN_SMART_TRIGGER) failed\nExiting!!"))
 exit(-1);

 printf("\nJoin session was executed successfully... Exiting.\n");
 goto Exit; // Exit gracefully
}

```

```

AssignOpenParams();

rc = SQLSetStmtAttrW(hstmt, SQL_INFX_ATTR_OPEN_SMART_TRIGGER,&gopenSmartTrigger, SQL_NTS);
if (checkError(rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error(main) in Step 5 --
SQLSetStmtAttr(SQL_INFX_ATTR_OPEN_SMART_TRIGGER) failed\nExiting!!"))
 exit(-1);

rc = SQLGetStmtAttrW(hstmt, SQL_INFX_ATTR_GET_LO_FILE_DESC_SMART_TRIGGER, (int *)&gFileDesc, SQL_NTS,
NULL);
if (checkError(rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error(main) in Step 6 --
SQLGetStmtAttr(SQL_INFX_ATTR_GET_LO_FILE_DESC_SMART_TRIGGER) failed\nExiting!!"))
 exit(-1);

for (i = 0; i < NUM_OF_INSTANCE; i++)
{
 printf("\nStart Thread %d", i + 1);
 AssignRegisterParams(&gSmartTriggerRegister[i], gFileDesc, i);
#ifdef NO_WIN32
 hThread_[i] = CreateThread(
 0, // Security Attributes (no security restrictions)
 stackSize, // Stack Size
 ThreadRegisterPushDataQuery, // Start address
 (void *)&gSmartTriggerRegister[i], //&inputValues[i],
 0, // Creation Flags (create running)
 &(threadID_[i]) // Thread Id
);
#else
 rc = pthread_create(&cpid[i],NULL,(void *)ThreadRegisterPushDataQuery,&gSmartTriggerRegister[i]);
#endif
}

#ifdef NO_WIN32
if(NUM_OF_INSTANCE > 0)
 WaitForMultipleObjects(NUM_OF_INSTANCE, hThread_, TRUE, INFINITE);
#else
for (i = 0; i < NUM_OF_INSTANCE; ++i)
 pthread_join(cpid[i], NULL);
#endif

rc = SQLGetStmtAttrW(hstmt, SQL_INFX_ATTR_GET_SESSION_ID_SMART_TRIGGER, (int *)&getSesID, SQL_NTS, NULL);
if (checkError(rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error(main) in Step 7 --
SQLGetStmtAttr(SQL_INFX_ATTR_GET_SESSION_ID_SMART_TRIGGER) failed\nExiting!!"))
 exit(-1);
printf("\nSession ID received = %d\n", getSesID);

if(getSesID > 0 && delete == 1)
{
 rc = SQLSetStmtAttrW(hstmt, SQL_INFX_ATTR_DELETE_SMART_TRIGGER, (int *)&getSesID, SQL_NTS);
 if (checkError(rc, SQL_HANDLE_STMT, hstmt, (SQLCHAR *) "Error(main) in Step 8 --
SQLSetStmtAttr(SQL_INFX_ATTR_DELETE_SMART_TRIGGER) failed\nExiting!!"))
 exit(-1);
 printf("\nSession ID deleted = %d\n",getSesID);
}

for (i = 0; i < NUM_OF_INSTANCE; ++i)
{

```

```

#ifdef NO_WIN32
 printf("Close Thread Handle : %d\n", i);
 CloseHandle(hThread_[i]);
#endif
 FreeMemory(&gSmartTriggerRegister[i]);
}

Exit:

/* CLEANUP: Close the statement handle
** Free the statement handle
** Disconnect from the datasource
** Free the connection and environment handles
** Exit
*/

/* Close the statement handle */
SQLFreeStmt (hstmt, SQL_CLOSE);

/* Free the statement handle */
SQLFreeHandle (SQL_HANDLE_STMT, hstmt);

/* Disconnect from the data source */
SQLDisconnect (hdbc);

/* Free the environment handle and the database connection handle */
SQLFreeHandle (SQL_HANDLE_DBC, hdbc);
SQLFreeHandle (SQL_HANDLE_ENV, henv);

return (rc);
}

```

## Best Practices/Trouble shooting

You can link Smart Trigger ODBC application with ODBC Driver Manager(DM). Smart Trigger feature has been tested with Windows Driver Manager and unixODBC Driver Manager. Due to synchronization behaviour of certain Driver Manager, it may not allow more than one SQLSetStmtAttr(with Smart trigger) / SQLGetStmtAttr to be called until previous call is completed, Smart Trigger is blocking call (due to internal call to ifx\_lo\_read()). In multi-threaded Smart Trigger application, this could cause unexpected/hang behaviour, to avoid the same one of the two below available options could be used:

- For unixODBC DM configuration in **.odbcinst.ini** file, use "Threading = 0" (disables synchronization/mutex of DM)



**Note:** Synchronization/mutex continues to work from Informix ODBC driver

- You can use **SQL\_INFX\_ATTR\_GET\_DATA\_SMART\_TRIGGER\_NO\_LOOP** interface in application and manage the WHILE loop in application (example given below). If you use this option, you may not need to set "Threading = 0" in DM.

## SQL\_INFX\_ATTR\_GET\_DATA\_SMART\_TRIGGER\_LOOP and SQL\_INFX\_ATTR\_GET\_DATA\_SMART\_TRIGGER\_NO\_LOOP interfaces

These two interfaces are provided to choose the appropriate one depending on the usage of DM.

In SQL\_INFX\_ATTR\_GET\_DATA\_SMART\_TRIGGER\_LOOP interface, ODBC internally uses WHILE loop to wait for data for Smart Trigger from server until user deregister or wants control back to application (example below), this is suitable when application uses unixODBC DM and wants to register many events/queries from application by setting "Threading = 0" in DM's .odbcinst.ini file.

In SQL\_INFX\_ATTR\_GET\_DATA\_SMART\_TRIGGER\_NO\_LOOP interface, the responsibility of looping to get next Smart Trigger from server lies with application, this way you don not really have to set "Threading = 0" in unixODBC DM.

### Example

```
Thread Start
SQLSetStmtAttr(SQL_INFX_ATTR_REGISTER_SMART_TRIGGER)
SQLGetStmtAttr(SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_LOOP) \ While loop inside the ODBC code.
OR
While(TRUE)
{
 SQLGetStmtAttr(SQL_INFX_ATTR_GET_DATA_SMART_TRIGGER_NO_LOOP)
 //Break with some business condition
}
Thread End
```



**Note:** On Unix/Linux user libthcli.so and on Windows use iclit09b.lib, you can directly link Smart Trigger application with Informix ODBC driver

If you have multiple events/queries to be registered from single database connection, then application should be linked with multi-threaded ODBC library "libthcli.so". Multiple events/queries could be registered only with multi-threaded application, where each event/query could be registered from each thread.

In addition to above extensions, there are following structures provided in **infxccli.h** file. The allocation & deallocation of memory for these structures and its members is responsibility of application.

```
typedef struct tagIfmxJoinTrigger
{
 void(*callback)
 (char const *jsonOutBuf);
 int *joinSessionID;
 BOOL *ControlBackToApplication;
 SQLWCHAR reserved[16];
} IFMX_JOIN_SMART_TRIGGER;

typedef struct tagIfmxOpenTrigger
{
 BOOL *isDetachable;
 int *timeOut;
 short *maxRecsPerRead;
 int *maxPendingOperations;
 SQLWCHAR reserved[16];
}
```

```

} IFMX_OPEN_SMART_TRIGGER;

typedef struct tagIfmxRegisterTrigger
{
 void(*callback)
 (char const *jsonOutBuf);
 int *loFileDescriptor;
 SQLWCHAR *tableName;
 SQLWCHAR *ownerName;
 SQLWCHAR *dbName;
 SQLWCHAR *sqlQuery;
 SQLWCHAR *label;
 BOOL *isDeregister;
 BOOL *ControlBackToApplication;
 SQLWCHAR reserved[16];
} IFMX_REGISTER_SMART_TRIGGER;

```

Structure IFMX\_OPEN\_SMART\_TRIGGER should be allocated/filled with respective values and must call following SQLSetStmtAttr() API. If "isDetachable? member is TRUE, it will retain the session even after closing the connection / finishing the application execution.

```
SQLSetStmtAttr(hstmt, SQL_INFX_ATTR_OPEN_SMART_TRIGGER, &gopenSmartTrigger, SQL_NTS);
```

If session is opened with "isDetachable? member TRUE, then application can call following **SQLGetStmtAttr()** to get the session ID, which could be used later to JOIN (more info below) the session. If "isDetachable? member was FALSE while opening the session, if so calling **SQLGetStmtAttr()** may return 0 (zero) or negative number, which are invalid session ID.

```
SQLGetStmtAttr(hstmt, SQL_INFX_ATTR_GET_SESSION_ID_SMART_TRIGGER, (int *)&getSesID, SQL_NTS, NULL);
```

# Index

## Special Characters

- .h files. 3
- .netrc file 48

## A

- Allocating handles. 3
- Architecture 5
- Arguments 3
  - null pointers 13
  - pointers 13
- Arrays. 3
- Attributes, columns 227
- Auto-commit mode. 3
- AUTOFREE feature 194

## B

- BIGINT data type 60, 60
- BIGSERIAL data type 60, 60
- Binary data
  - C data type 72
  - converting to C 87
  - transferring 75
- Binary data, converting to SQL 96
- Binding columns 3, 143, 220
- Binding parameters. 3
- Bit data, converting to SQL 97
- BLOB data type 64
- Bookmarks, support 35
- Boolean data
  - C data type 72
  - converting to C 87
- BOOLEAN data type 60
- Boundaries, segment 13
- Buffers 3
  - allocating 13
  - input 13, 13
  - interoperability 13
  - maintaining pointers 13
  - NULL data 14
  - null pointers 13
  - null-termination 14
  - output 13, 14
  - segment boundaries 13
  - truncating data 14
- Bulk operations 244
- BYTE data type 60

## C

- C data type 3, 72
  - binary 72
  - boolean 72
  - character 72
  - conversion examples 94, 102
  - converting from SQL data type 59, 84
  - converting to SQL data types 95
  - date 72
  - default conversions 85
  - HCL OneDB
  - ODBC Driver 59
  - numeric 72
  - SQL\_C\_BINARY 72
  - SQL\_C\_BIT 72
  - SQL\_C\_CHAR 72
  - SQL\_C\_DATE 72
  - SQL\_C\_DOUBLE 72
  - SQL\_C\_FLOAT 72
  - SQL\_C\_LONG 72

- SQL\_C\_SHORT 72
- SQL\_C\_SLONG 72
- SQL\_C\_SSHORT 72
- SQL\_C\_STINYINT 72
- SQL\_C\_TIMESTAMP 72
- SQL\_C\_TINYINT 72
- SQL\_C\_ULONG 72
- SQL\_C\_USHORT 72
- SQL\_C\_UTINYINT 72
- standard 59
- timestamp 72
- typedefs 59
- Calls, executing with SQLPrepare and SQLExecute 197
- Canceling, connection browsing 227
- Challenge and response buffer pointers 29
- CHAR data type 60
- Character data 72
  - converting to C 88
  - converting to SQL 97
  - empty string 13
- CHARACTER data type 60
- CHARACTER VARYING data type 60, 60
- Client functions, calling 161, 161
- Client locale 25
- CLIENT\_LOCALE environment variable 25, 40
- CLOB data types 64
- Code, example 3
- Collections 64
  - buffers 143
  - converting SQL data 144
  - creating 153, 183
  - current position 160
  - deleting 160, 185
  - inserting 160, 188
  - local fetch 144
  - modifying 160
  - retrieving 187
  - retrieving information 161
  - transferring 143
  - updating 160, 191
- Columns 3
  - attributes 227
  - binding. 3
  - precision. 3
  - procedure 246
- Concurrency 37
- Configuring a DSN on UNIX 37
- Configuring a DSN on Windows 48
- Configuring data sources. 3
- connection attribute
  - SQL\_INFX\_ATTR\_DELIMITED 192
- Connection handles
  - defined connection handles
    - HDBC variable type 13
    - SQLFreeConnect 236
  - connection string
    - enabling delimited identifiers in 192
  - connection string keywords
    - DELIMITED 192
- Connections, SQLDisconnect 227
- Converting data 3
  - C to SQL 95
  - default conversions 85
  - examples 94, 102
  - SQL to C 84
- Create-time flags 106

## Cursor

- automatically freeing 194
- enable insert cursor 48
- insert 194
- position errors 232
- Report KeySet 48
- scrollable 48

## D

- Data
  - committing 202
  - converting 3
  - length 65
  - transferring in binary form 75
  - translating. 3
  - truncating. 3
  - updating 201
- Data sources
  - configuring on UNIX 37
  - configuring on Windows 48
- Data transfer
  - error checking 192
- Data types. 3
- Data-source specification 40
- Database locale 25
- Date data
  - C data type 72
  - converting to C 90
  - converting to SQL 99
- DATE data type 60
- DATE\_STRUCT typedef 72
- DATETIME data type 60
- DB\_LOCALE environment variable 25, 40
- DBCENTURY environment variable 8
- DEC data type 60
- DECIMAL data type 60
- Default fetch type for UDTs 77
- DELETE statements 3, 241, 248
  - affected rows 248
- DELIMITED
  - connection string keyword 192
- DELIMITED environment variable
  - in ODBC 192, 192
- delimited identifiers
  - in ODBC 192
- Descriptors, columns 227
- Diagnostics 205
- Disk-storage information 105
- Display size 65
- DISTINCT data type 64
- DOUBLE PRECISION data type 60
- Driver manager, described 5
- Driver,
  - HCL OneDB
  - ODBC 5
- Drivers, allocating handles 219
- DSN settings 77

## E

- Empty strings 13
- Environment handles
  - defined 13
  - SQLAllocEnv 219
  - SQLFreeEnv 236
- environment variable
  - DELIMITED
    - in ODBC 192



- Environment variables 8
  - CLIENT\_LOCALE 25, 40
  - DB\_LOCALE 25, 40
  - DBCENTURY 8
  - GL\_DATE 8
  - IFX\_LOB\_XFERSIZE 192
  - ODBCINI 9
  - ONEDB\_SQLHOSTS 9
  - ONEDB\_HOME 8
  - PATH 8
  - TRANSLATION\_OPTION 25
  - TRANSLATIONDLL 25, 40
  - VMBCHARLENEXACT 25
- Error descriptions
  - HCL OneDB
    - 33
  - ISAM 33
- Error handling
  - checking during data transfer 192
- Error messages
  - HCL OneDB
    - mapping SQLSTATE values 205
- Errors 3
  - diagnostic SQLSTAGE values 205
  - error messages 205
  - handling with OPTMSG 204
  - mapping
    - HCL OneDB
    - to SQLSTATE values 205
  - rowsets 232
- Examples, data conversion 94, 102
- Extended data types 4
- Extensive error detection 4
- External authentication 28

## F

- fCType 85
- Fetch simple large object data 196
- Fetch type 77
- Fetching data. 3
- Files
  - .h files. 3
  - .netrc 48
  - infxcli.h 9, 64
  - odbc.ini 39
  - odbcinst.ini 37
- FLOAT data type 60
- Floating point data
  - converting to C 91
  - converting to SQL 100
- Freeing handles. 3
- Functions
  - rows and collections
    - ifx\_rc\_count() 183
    - ifx\_rc\_create() 183
    - ifx\_rc\_delete() 185
    - ifx\_rc\_describe() 185
    - ifx\_rc\_fetch() 187
    - ifx\_rc\_free() 188
    - ifx\_rc\_insert() 188
    - ifx\_rc\_isnull() 189
    - ifx\_rc\_setnull() 190
    - ifx\_rc\_typespec() 190
    - ifx\_rc\_update() 191
  - smart large objects 163
    - ifx\_lo\_alter() 163
    - ifx\_lo\_close() 164
    - ifx\_lo\_col\_info() 165
    - ifx\_lo\_create() 165
    - ifx\_lo\_def\_create\_spec() 166

- ifx\_lo\_open() 167
- ifx\_lo\_read() 168
- ifx\_lo\_readwithseek() 169
- ifx\_lo\_seek() 170
- ifx\_lo\_specget\_estbytes() 171
- ifx\_lo\_specget\_extsz() 171
- ifx\_lo\_specget\_flags() 172
- ifx\_lo\_specget\_maxbytes() 172
- ifx\_lo\_specget\_sbspace() 173
- ifx\_lo\_specset\_estbytes() 173
- ifx\_lo\_specset\_extsz() 174
- ifx\_lo\_specset\_flags() 175
- ifx\_lo\_specset\_maxbytes() 176
- ifx\_lo\_specset\_sbspace() 176
- ifx\_lo\_stat\_atime() 177
- ifx\_lo\_stat\_cspec() 178
- ifx\_lo\_stat\_ctime() 178
- ifx\_lo\_stat\_refcnt() 179
- ifx\_lo\_stat\_size() 179
- ifx\_lo\_stat() 177
- ifx\_lo\_tell() 180
- ifx\_lo\_truncate() 180
- ifx\_lo\_write() 181
- ifx\_lo\_writewithseek() 182

## G

- GLS feature
  - data types 4, 64
- GLS. 3

## H

- Handles. 3
- hdbc. 3
- Header files
  - required 9
  - sqlxcli.h C data type 10
- henv. 3
- hstmt. 3

## I

- identifiers
  - delimited
    - enabling/disabling using ODBC 192
- IDSSECURITYLABEL data type 60
- ifx\_lo\_alter() 163
- ifx\_lo\_close() 164
- ifx\_lo\_col\_info() 165
- ifx\_lo\_create() 165
- ifx\_lo\_def\_create\_spec() 166
- ifx\_lo\_open() 167
- ifx\_lo\_read() 168
- ifx\_lo\_readwithseek() 169
- ifx\_lo\_seek() 170
- ifx\_lo\_specget\_estbytes() 171
- ifx\_lo\_specget\_extsz() 171
- ifx\_lo\_specget\_flags() 172
- ifx\_lo\_specget\_maxbytes() 172
- ifx\_lo\_specget\_sbspace() 173
- ifx\_lo\_specset\_estbytes() 173
- ifx\_lo\_specset\_extsz() 174
- ifx\_lo\_specset\_flags() 175
- ifx\_lo\_specset\_maxbytes() 176
- ifx\_lo\_specset\_sbspace() 176
- ifx\_lo\_stat\_atime() 177
- ifx\_lo\_stat\_cspec() 178
- ifx\_lo\_stat\_ctime() 178
- ifx\_lo\_stat\_refcnt() 179
- ifx\_lo\_stat\_size() 179
- ifx\_lo\_stat() 177
- ifx\_lo\_tell() 180
- ifx\_lo\_truncate() 180
- ifx\_lo\_write() 181

- ifx\_lo\_writewithseek() 182
- IFX\_LOB\_XFERSIZE
  - environment variable 192
- ifx\_rc\_count() 183
- ifx\_rc\_create() 183
- ifx\_rc\_delete() 185
- ifx\_rc\_describe() 185
- ifx\_rc\_fetch() 187
- ifx\_rc\_free() 188
- ifx\_rc\_insert() 188
- ifx\_rc\_isnull() 189
- ifx\_rc\_setnull() 190
- ifx\_rc\_typespec() 190
- ifx\_rc\_update() 191
- IN parameters
  - used during execution of SPL 198
- Include files. 3
- infxcli.h file 64
- Initializing data sources 37
- Input buffers 13
- Insert cursor 194
  - enabling 48
- INSERT statements
  - affected rows 248, 248
  - SQLParamOptions 244
- INSERT statements. 3
- INT data type 60
- INT8 data type 60
- Integer data
  - converting to C 91
  - converting to SQL 100
- INTEGER data type 60
- Internet Protocol Version 6 4
- Interoperability
  - buffer length 13
  - default C data type 85
  - transferring data 75

## L

- LDAP authentication on Windows 29
- Length
  - data 65
- Length, buffers
  - input 13
  - maximum 13
  - output 14
- Length, defined 65
- Length, unknown
  - precision 65
- Libraries 10
- Library
  - HCL OneDB
  - ODBC Driver
    - 10
    - translation 25
    - shared 25
- LIST data type 64
- LO\_APPEND 121
- LO\_BUFFER 121
- LO\_DIRTY\_READ 121
- LO\_KEEP\_LASTACCESS\_TIME 106
- LO\_NOBUFFER 121
- LO\_NOKEEP\_LASTACCESS\_TIME 106
- LO\_NOLOG 106
- LO\_RDONLY 121
- LO\_RDWR 121
- LO\_SEEK\_CUR 169, 170, 182
- LO\_SEEK\_END 169, 170, 182
- LO\_SEEK\_SET 169, 170, 182
- LO\_WRONLY 121
- Locales

- client 25
- database 25
- lofd 104
- Login authorization. 3
- Logon ID 40
- Long identifiers 4
- loptr 104
- lospec 104
- lostat 104
- LVARCHAR data type 60

## M

- Manual-commit mode. 3
- Memory. 3
- Message chaining 203
- Message transfer optimization 202
- Messages, error. 3
- Microsoft Transaction Server 4
- Migrating to
  - HCL OneDB
- ODBC
  - DSN connection on UNIX 77
  - DSN connection on Windows 77
- Modes
  - auto-commit. 3
  - manual commit. 3
- MONEY data type 60
- MTS. 3
- MULTISET data type 64
- Multithreading, with environment handles 219

## N

- Named rows 64
- NCHAR data type 64
- NULL data
  - output buffers 14
- Null pointers
  - input buffers 13
  - output buffers 14
- Null-termination byte
  - embedded 13
  - examples 94, 102
  - input buffers 13
  - output buffers 14
- Numeric data 3
  - C data type 72
    - converting to C 91
    - converting to SQL 100
  - TIMESTAMP\_STRUCT 72
  - UCHAR 72
  - UWORD 72
- NUMERIC data type 60
- NVARCHAR data type 64

## O

- odbc.ini file 39
  - Data Source Specification section 40
  - ODBC Data Sources section 40
- odbc.ini tracing options 47
- ODBCINI environment variable 9
- odbcinst.ini file 37
- ONEDB\_ SQLHOSTS environment variable 9
- ONEDB\_HOME environment variable 8
- OPAQUE data type 64
- Optimistic concurrency control. 3
- OPTMSG 202
- OUT parameters
  - used during execution of SPL 198
- Output buffers 14

## P

- PAM. 29

- Parameters 3
  - arrays 244
  - binding 197
    - SQLBindParameter 220
  - number 242
    - used during execution of SPL 198
- Passwords 48
- PATH environment variable 8
- Pluggable Authentication Module
  - Connect functions 17, 29
  - Connection pooling 29
  - Intermediate Code 31
  - SQLSetConnectAttr() function 29
  - Third party connections 31
- Pointers, maintaining 13
- Pointers, null. 3
- Position, cursor. 3
- Positioned
  - DELETE statements 201
  - UPDATE statements 201
- Precision 65
- Procedure
  - defined 247
- Procedure columns 246
- Procedures, SQL 247
  - SQLSTAGE and error values 247
- pwd 40

## Q

- Queries. 3

## R

- REAL data type 60
- Report KeySet cursors 48
- Report sets
  - SQLDescribeCol 226
- Report standard ODBC data type
  - DSN settings 77
- Result sets 3
  - arrays. 3
  - defined 11
  - SQLNumResultCols 243
  - SQLRowCount 248, 248
- Retrieving data
  - arrays. 3
  - binding columns. 3
  - rows. 3
- Row status array, errors 232
- Rows 3, 64
  - affected 248, 248
  - and collections 143
  - buffers 143
  - converting SQL data 144
  - creating 153, 183
  - current position 160
  - deleting 160, 185
  - errors in 232
  - inserting 160, 188
  - local fetch 144
  - modifying 160
  - retrieving 187
  - retrieving information 161
  - transferring 143
  - updating 160, 191
- Rowsets
  - errors 232

## S

- SBSPACENAME 108
- Scale, defined 65
- SCHAR typedef 72
- Scrollable cursor 48

- SDOUBLE typedef 72
- SDWORD typedef 72
- Segment boundaries 13
- SELECT statements 3
  - affected rows 248, 248
  - bulk 244
- SERIAL data type 60
- SERIAL8 data type 60
- SET data type 64
- Setting GLS options
  - UNIX 25
  - Windows 25
- setup.odbc 9
- SFLOAT typedef 72
- Simple large object fetches 196
- Size, display 65
- SMALLFLOAT data type 60
- SMALLINT data type 60
- Smart large objects 64
  - access modes 121
  - accessing 118
  - allocation extent size 105
  - altering 163
  - closing 124, 164
  - creating 109, 165
  - data structures 104
  - disk-storage information 105
  - estimated size 105
  - file descriptor 104
  - functions 163
  - getting file position 180
  - ifx\_lo functions 120
  - inheritance hierarchy 107
  - inserting 118
  - last access-time 106
  - lightweight I/O 122
  - locks 123
  - logging indicator
    - LO\_LOG 106
  - maximum size 105
  - modifying 124
  - ODBC API 119
  - opening 120, 167
  - pointer structure 104
  - reading 168, 169
  - retrieving status 133
  - sbspace name 105
  - selecting 120
  - setting file position 170, 180
  - specification structure 104
  - status structure 104
  - storage characteristics 105, 165
  - transferring 118
  - updating 118
  - writing 181, 182
- smart trigger 258
- SQL data types 3
  - BLOB 64
  - BOOLEAN 60
  - BYTE 60
  - CHAR 60
  - CHARACTER 60
  - CHARACTER VARYING 60
  - CLOB 64
    - collection 64
    - conversion examples 94, 102
    - converting from C data type 95
    - converting to C data type 84
  - DATE 60
  - DATETIME 60
  - DEC 60

DECIMAL 60  
 default C data type 85  
 display size 65  
 DISTINCT 64  
 DOUBLE PRECISION 60  
 FLOAT 60  
 HCL OneDB  
   59  
 HCL OneDB  
 ODBC Driver  
   59  
 IDSSECURITYLABEL 60  
 INT 60  
 INT8 60  
 INTEGER 60  
   length 65  
 LIST 64  
 LVARCHAR 60  
 MONEY 60  
 MULTISSET 64  
 NCHAR 64  
 NUMERIC 60  
 NVARCHAR 64  
 OPAQUE 64  
   precision 65  
 REAL 60  
   row 64  
   scale 65  
 SERIAL 60  
 SERIAL8 60  
 SET 64  
 SMALLFLOAT 60  
 SMALLINT 60  
 smart large object 64  
 SQL\_BIGINT 60  
 SQL\_BIT 60  
 SQL\_CHAR 60  
 SQL\_DATE 60  
 SQL\_DECIMAL 60  
 SQL\_DOUBLE 60  
 SQL\_IFMX\_UDT\_BLOB 64  
 SQL\_IFMX\_UDT\_CLOB 64  
 SQL\_INFX\_BIGINT 60  
 SQL\_INFX\_UDT\_FIXED 64  
 SQL\_INFX\_UDT\_VARYING 64  
 SQL\_INTEGER 60  
 SQL\_LONGVARBINARY 60  
 SQL\_LONGVARCHAR 60  
 SQL\_REAL 60  
 SQL\_SMALLINT 60  
 SQL\_TIMESTAMP 60  
 SQL\_VARCHAR 60  
 TEXT 60  
 VARCHAR 60  
 SQL statements  
   native 241  
 SQL\_ATTR\_ROW\_ARRAY\_SIZE 196  
 SQL\_BIGINT data type 60  
 SQL\_BIT data type 60  
 SQL\_C\_BINARY data type 72  
 SQL\_C\_BIT data type 72  
 SQL\_C\_CHAR data type 72  
 SQL\_C\_DATE data type 72  
 SQL\_C\_DOUBLE data type 72  
 SQL\_C\_FLOAT data type 72  
 SQL\_C\_LONG data type 72  
 SQL\_C\_SHORT data type 72  
 SQL\_C\_SLONG data type 72  
 SQL\_C\_SSHORT data type 72  
 SQL\_C\_STINYINT data type 72  
 SQL\_C\_TIMESTAMP data type 72  
 SQL\_C\_TINYINT data type 72  
 SQL\_C\_ULONG data type 72  
 SQL\_C\_USHORT data type 72  
 SQL\_C\_UTINYINT data type 72  
 SQL\_CHAR data type 60  
 SQL\_DATE data type 60  
 SQL\_DECIMAL data type 60  
 SQL\_DESC\_OCTET\_LENGTH, and  
   bookmarks 35  
 SQL\_DIAG\_ISAM\_ERROR attribute 33  
 SQL\_DOUBLE data type 60  
 SQL\_ENABLE\_INSERT\_CURSOR 194  
 SQL\_IFMX\_UDT\_BLOB data type 64  
 SQL\_IFMX\_UDT\_CLOB data type 64  
 SQL\_INFX\_ATTR\_AUTO\_FREE 194  
 SQL\_INFX\_ATTR\_DEFAULT\_UDT\_FETCH\_TYPE 77  
 SQL\_INFX\_ATTR\_DEFERRED\_PREPARE 195  
 SQL\_INFX\_ATTR\_DELIMITIDENT connection  
   attribute 192  
 SQL\_INFX\_ATTR\_ENABLE\_INSERT\_CURSORS 48  
 SQL\_INFX\_ATTR\_ENABLE\_SCROLL\_CURSORS 48  
 SQL\_INFX\_ATTR\_IDSISAMERRMSG  
   attribute 33  
 SQL\_INFX\_ATTR\_LO\_AUTOMATIC 76, 76, 119  
 SQL\_INFX\_ATTR\_ODBC\_TYPES\_ONLY 76, 76  
 SQL\_INFX\_ATTR\_OPTIMIZE\_AUTOCOMMIT 48  
 SQL\_INFX\_ATTR\_OPTMSG 202  
 SQL\_INFX\_ATTR\_OPTOFC 48  
 SQL\_INFX\_ATTR\_REPORT\_KEYSET\_CURSORS 48  
 SQL\_INFX\_ATTR\_UPDATE\_DESCRIBE 76  
 SQL\_INFX\_BIGINT data type 60, 60  
 SQL\_INFX\_UDT\_FIXED data type 64  
 SQL\_INFX\_UDT\_VARYING data type 64  
 SQL\_INTEGER data type 60  
 SQL\_LONGVARBINARY data type 60  
 SQL\_LONGVARCHAR data type 60, 60  
 SQL\_REAL data type 60  
 SQL\_SMALLINT data type 60  
 SQL\_TIMESTAMP data type 60  
 SQL\_VARCHAR data type 60  
 SQLAllocConnect 3  
   function description 219  
 SQLAllocEnv 3  
   function description 219  
 SQLAllocStmt 3  
   function description 219  
 SQLBindCol  
   function description 220  
 SQLBindParameter  
   function description 220  
 SQLBrowseConnect  
   function description 221  
 SQLBulkOperations 36  
   bookmarks 35  
   function description 35  
 SQLCancel, function description 222  
 SQLColAttributes, function description 223  
 SQLColumnPrivileges, function  
   description 223  
 SQLColumns, function description 224  
 SQLConnect, function description 225  
 SQLDataSources, function description 226  
 SQLDescribeCol, function description 226  
 SQLDescribeParam 36  
 SQLDisconnect, function description 227  
 SQLDriverConnect, function description 227  
 SQLDrivers, function description 229  
 SQLError 3  
   function description 229  
 SQLExecDirect, function description 230  
 SQLExecute, function description 231  
 SQLExtendedFetch  
   bookmarks 35  
   function description 232  
 SQLFetch, function description 234  
 SQLFetchScroll, bookmarks 35  
 SQLForeignKeys, function description 235  
 SQLFreeConnect  
   function description 236  
 SQLFreeEnv 3  
   function description 236  
 SQLFreeHandle 3  
 SQLFreeStmt 3  
   function description 237  
 SQLGetConnectOption, function  
   description 237  
 SQLGetCursorName, function description 237  
 SQLGetData, function description 238  
 SQLGetDiagRec API 33  
 SQLGetFunctions, function description 239  
 SQLGetInfo, function description 239  
 SQLGetStmtOption, function description 240  
 SQLGetTypeInfo  
   function description 240  
   supported data types 10  
 SQLMoreResults, function description 241  
 SQLNativeSql, function description 241  
 SQLNumParams, function description 242  
 SQLNumResultCols, function description 243  
 SQLParamData  
   function description 243  
   SQLPutData 243  
 SQLParamOptions  
   function description 244  
   multiple parameter values 244  
 SQLPrepare  
   deferring execution 195  
   function description 244  
 SQLPrimaryKeys, function description 245  
 SQLProcedureColumns, function  
   description 246  
 SQLProcedures, function description 247  
 SQLPutData  
   function description 248  
   SQLParamData 243  
 SQLRowCount, function description 248  
 SQLSetConnectOption 3  
   function description 249  
 SQLSetCursorName, function description 249  
 SQLSetPos  
   column binding 220  
   error messages 205  
   LockType argument  
     SQL\_CA1\_LOCK\_NO\_CHANGE 15  
   operation argument  
     SQL\_CA1\_POS\_DELETE 15  
     SQL\_CA1\_POS\_POSITION 15  
     SQL\_CA1\_POS\_REFRESH 15  
     SQL\_CA1\_POS\_UPDATE 15  
   positioned UPDATE and DELETE  
   statements 201  
   scroll cursors 196, 201  
   SQLGetData 196, 238  
   SQLParamData 243  
   SQLRowCount 248  
 SQLSetStmtOption  
   function description 250  
 SQLSpecialColumns, function description 251  
 SQLSTATE  
   naming conventions 205  
   values 205  
   values. 3

- SQLStatistics 251
  - function description 251
- SQLTablePrivileges, function description 252
- SQLTables, function description 253
- SQLTransact, function description 254
- SqlType 60
- Statement handles
  - defined 13
  - HSTMT variable type 13
  - SQLAllocStmt 219
  - SQLFreeStmt 237
- Status array, errors 232
- Status information. 3
- Storage characteristics
  - create-time flags 106
  - disk-storage information 105
  - inheritance hierarchy 107
- String data. 3
- SWORD typedef 72
- syscolattribs 109

## T

- Table
  - columns. 3
  - indexes. 3
  - rows. 3
- Termination byte, null. 3
- TEXT data type 60
- Threads, multiple
  - with environment handles 219
- Time-stamp data 72
  - converting to C 92
  - converting to SQL 101
- TIMESTAMP\_STRUCT typedef 72
- Tracing values in ODBC 47
- Transactions
  - concurrency 37
  - incomplete 227
- Transferring binary data 75
- Translation
  - library 25
  - options 25
  - shared library 25
    - error 205, 221, 225, 227, 249
- TRANSLATION\_OPTION environment variable 25
- TRANSLATIONDLL environment variable 25, 40
- Truncating data 3, 3
  - output buffers 14
  - SQLBindCol 220
- Typedefs
  - DATE\_STRUCT 72
  - SCHAR 72
  - SDOUBLE 72
  - SDWORD 72
  - SFLOAT 72
  - SWORD 72
  - UDWORD 72

## U

- UCHAR typedef 72
- UDT fetch type 77
- UDWORD typedef 72
- Unicode 4
- Unnamed rows 64
- UPDATE statements 3
  - affected rows 248, 248
  - bulk 244
- UPDATE\_DESCRIBE 57
- User ID 48
- UWORD typedef 72

## V

- VARCHAR data type 60
- Variables, binding
  - estbytes input argument 163
  - SQL\_INFX\_UDT\_FIXED 86
  - SQL\_INFX\_UDT\_VARYING 86
- Visual Basic client-side cursors 60
- VMBCHARLENEXACT environment variable 25

## W

- Window handles. 3

## X

- XA 4, 9, 28