Altibase Application Development

ODBC Reference

Release 6.1.1

April 23, 2012

Altibase Application Development ODBC Reference

Release 6.1.1

Altibase Corporation

10F, Daerung PostTower II, 182-13,

Guro-dong Guro-gu Seoul, 152-847, Korea

Telephone: +82-2-2082-1000   Fax: 82-2-2082-1099

E-mail: support@altibase.com   www: http://www.altibase.com

Contents

Preface

About This Manual

This manual describes how to use the ODBC.

Types of Users

This manual is for ALTIBASE® HDB™ users as follows.

- Database administrator
- Performance manager
- Database user
- Application designers
- Technical support staff

This manual assumes that you have the following background :

- Basic knowledge required for computers, operating systems, and operating system command
- Experiences in using the relational database or understanding on database concepts.
- Computer programming experience
- Experiences in administration of a database server, operating system and network.

Software Dependencies

This manual assumes that your database server is ALTIBASE HDB 5.5.1.

How This Manual is Structured

This manual covers the following topics :

- Chapter1: ODBC Introduction

  This chapter shows you how to use ODBC, ODBC programming procedure, ODBC functions.
- Chapter2: ODBC Functions

  This chapter shows you the specifications of ODBC functions.
- Chapter3: LOB Interface

  This chapter shows functions and data types that can be used for handling LOB data.
- Appendix A. Sample Codes

  This appendix shows the entire sample codes used this documents.

- Appendix B. Data Types

  This appendix explains the ALTIBASE HDB SQL data types, C data types, and conversion between data types.

- Appendix C. ODBC Error Codes

  This appendix explains the ALTIBASE HDB ODBC Drive Error reference as defined in X/Open and SQL Access Group SQL CAE specification.

- Appendix D. ODBC Conformance Levels

  This appendix describes the conformance level of ALTIBASE HDB ODBC Driver.

- Appendix E. Upgrade

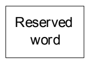
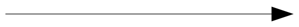
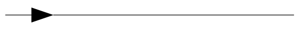
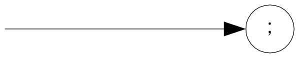
Documentation Conventions

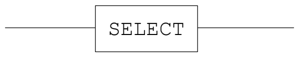
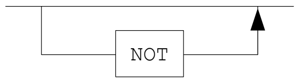
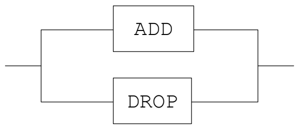
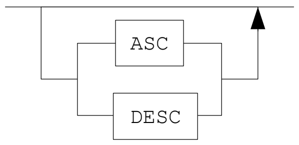
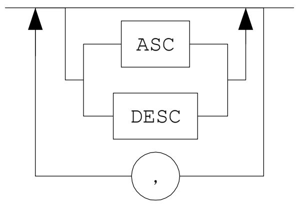
This section offers documentation conventions as follows. They make it easier to gather information from ALTIBASE HDB manuals.

- Command-Line Conventions
- Typographical Conventions

Command-Line Conventions

This section defines and illustrates the format of commands that are available in Altibase products. These commands have their own conventions, which might include alternative forms of a command, required and optional parts of the command, and so forth.

Element	Description
	The command starts. The syntax element which is not a complete command starts with an arrow.
	The command continues in the next line. The syntax element which is not a complete command terminates with this symbol.
	The command continues from the previous line. The syntax element which is not a complete command starts with this symbol.
	End of a statement.

Element	Description
	Mandatory
	Optional
	Mandatory field with optional items Only one field must be provided..
	Optional field with optional item
	Optional Multiple fields are allowed. The comma must be in front of every repetition.

Typographical Convetions

This manual uses the following standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning	Example
[]	Displays the optional fields.	`VARCHAR [(size)] [[FIXED |] VARIABLE]`
{}	Displays the mandatory fields. Specifies a field that requires selection of more than one item.	`{ ENABLE | DISABLE | COMPILE }`

Convention	Meaning	Example

|	Argument indicating optional or mandatory fields	`{ ENABLE | DISABLE | COMPILE }`
		`[ ENABLE | DISABLE | COMPILE ]`

.
.
.
	Repetition of the previous argumentSpecifies the omission of the example codes.	`iSQL> select e_lastname from employees;`
		`E_LASTNAME`
		`------------------------`
		`Moon`
		`Davenport`
		`Kobain`
		`.`
		`.`
		`.`
		`20 rows selected.`

Other symbols	Other Symbols	`EXEC :p1 := 1;`
		`acc NUMBER(11,2);`

Italics	Within text, new terms and emphasized words appear in italics. Within syntax, diagrams, values that you are to specify appear in italics.	`SELECT * FROM table_name;` CONNECT *userID/password*;
Lower Case Letters	Program elements provided by the user such as table names, column names, file names, etc.	`SELECT e_lastname FROM employees;`
Upper Case Letters	All elements provided by the system or keywords appear in uppercase letter.	`DESC SYSTEM_.SYS_INDICES_;`

Related Reading

For additional technical information, consult the following manuals:

- ALTIBASE HDB Getting Started
- ALTIBASE HDB Administrator's Manual
- ALTIBASE HDB Replication Manual
- ALTIBASE HDB Precompiler User's Manual
- ALTIBASE HDB ODBC Reference
- ALTIBASE HDB Application Program Interface User's Manual
- ALTIBASE HDB iSQL User's Manual
- ALTIBASE HDB Utilities Manual
- ALTIBASE HDB Error Message Reference

On-line Manuals

Manuals are available at Altibase Technical Center (http://atc.altibase.com/).

Altibase Welcomes Your Comments

Please let us know what you like or dislike about our manuals. To help us with future versions of our manuals, please tell us about any corrections or classifications that you would find useful.

Include the following information :

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following electronic mail address : support@altibase.com

When you need an immediate assistance regarding technical issues, please contact Altibase Technical Center.

Thank you. We appreciate your feedback and suggestions.

1 ODBC Introduction

The ODBC is a callable SQL programming interface. The callable SQL interface is used to access the database and execute a dynamic SQL statement through calling the function.

1.1 Open Database Connectivity

The ODBC is a standard open application program interface to access the database. It makes possible for applications to access data from a variety of database management systems (DBMSs). It provides calling level interfaces to access database servers and to execute SQL statements. That application will be independent of DBMS. This Guide is for the ALTIBASE HDB ODBC Driver's users.

1.1.1 Backgrounds of the ODBC

The ODBC was first created by SQL Access Group (SAG) and introduced in September 1992. Now various versions for UNIX, OS/2, Microsoft Windows and Macintosh are available also.

The ODBC is based on the callable standard SQL interface(CLIs). The ODBC enables the programs to use the SQL request to access the database although those programs do not know the independent interface of the database. The ODBC receives the SQL request and converts each request in a format comprehensible for the database system.

1.1.2 ODBC versus Embedded C/C++ Programming

The ODBC interface is designed for use C/C++ Languages. Then what's the difference between programs using ODBA and C/C++ precompiler.An application that uses C/C++ Precompiler interface requires a precompiler to convert an SQL statement into a code. The ODBC application uses the standard function that executes an SQL statement while it does not need pre-compiling.

The advantage is that it can use other database products that provide the standard functions. In other words, the ODBC supports independent development of an application for each database product.

1.1.3 Groups of ODBC Functions

The API of the ODBC consists of functions that define environment for running applications, managing connections, and processing SQL statements and transactions. Depending on the features of each function, these are groupped by followings:

- Managing Environments and Connections
- SQL Processing
- Setting and Retrieving Driver Attributes
- Meta data processing

1.1.3.1 Managing Environments and Connections

The APIs that allocate resources necessary for the connection to the database server and provide connection-related features. Releases the memory space after disconnected from datasources.

1.1.3.2 SQL Processing

The APIs that allocate resources and prepare commands for the processing, executing and retreiving results of SQL statements.

1.1.3.3 Setting and Retrieving Driver Attributes

The APIs that set the environment for the processing of the SQL, connection status, and statement attributes.

1.1.3.4 Metadata Processing

The APIs that provide features to define tables and columns and to retrieve database metadata.

1.2 Using the ODBC

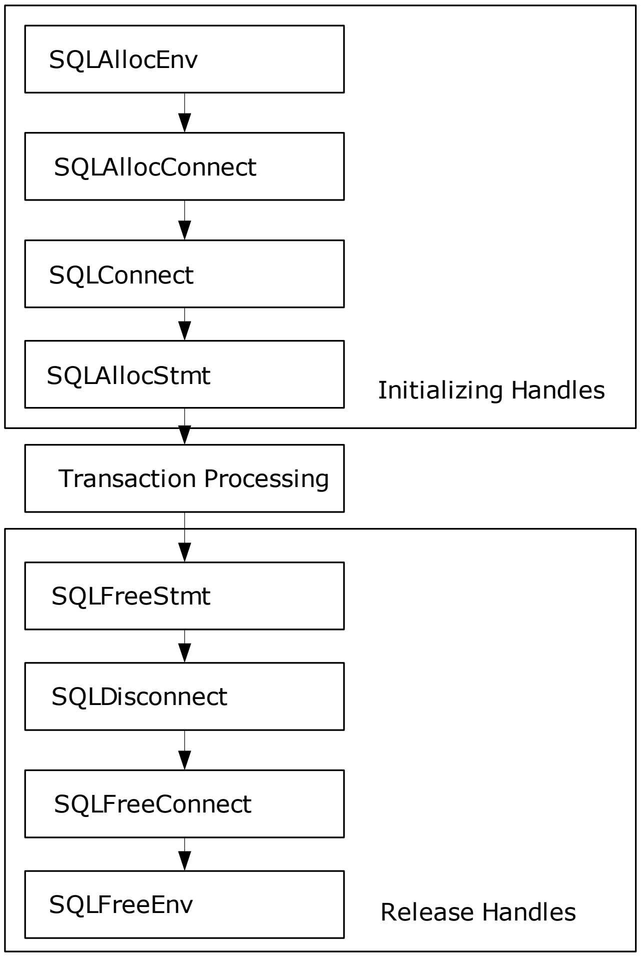
This chapter explains how to develop user application programs using the ODBC driver of ALTIBASE HDB.

1.2.1 Basic Usages

The ODBC application program generally consists of following three parts:

- Initializing Handles
- Transaction Processing
- Release Handles

Besides the above, The diagnosis messages is made through the all parts of an application.



1.2.2 Initializing Handles

This is a part to allocate and initialize the environment and connection handles.

Transition from one phase to the next phase is made through transmission of proper handles to send information about the execution results from the previous phase. Handle types provided by the ODBC are as follows:

1.2.2.1 Environment Handles

Tthe environment handles obtain general environment about an application environment. The environment handles must be allocated before the connection handle. In one application, multiple environment handles can be allocated.

1.2.2.2 Connection Handles

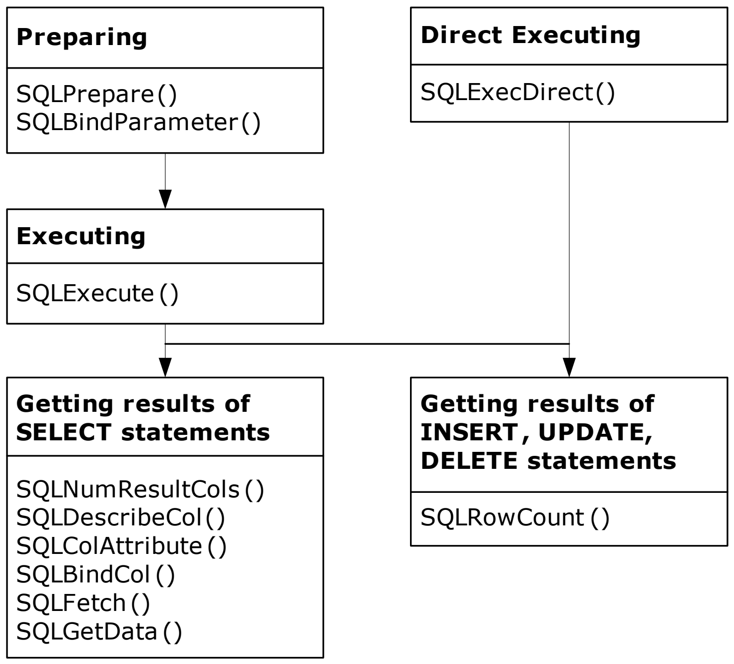
The connection handles obtain connection-related information that the ODBC manages. They Inclu-desconnection and transaction status, and diagnosis information. An application allocates the connection handle for each connection and attempts to connect to a database server.

1.2.2.3 Statement handles

The statement handles obtain the information of SQL statements. The statement handleare related to the connection handles. They allocate the statement handle to execute SQL statements. Maximum 1024 statements can be allocated to one connection.

1.2.3 Processing of Transactions

The following figure is a general procedure of calling functions to processing a transaction.



1.2.4 Release Handle

This step is for releasing the handles and meory allocated by an application, and finishing an application.

1.2.5 Managing Diagnosis Messages

Diagnosis is to handle the warning or error status occurred in an application. There are 2 Levels of disgnosis messages in the ODBC.

1.2.5.1 Application Return Values

Return Values	Description
SQL_SUCCESS	Successful completion of the function
SQL_SUCCESS_WITH_INFO	Successful execution with warning and other information
SQL_NO_DATA_FOUND	The function is successful, but there is no related data.
SQL_ERROR	Failure of the function
SQL_INVALID_HANDLE	The function failed due to invalid input handles.

The diagnosis messages are returned except the case of SQL_SUCCESS, SQL_NO_DATA_FOUND, SQL_INVALID_HANDLE. To check the diagnosis message, call SQLGetDiagRec(), SQLGetDiagField()

1.2.5.2 Diagnosis Messages

The diagnosis message is a five-bytes alphanumeric character string. The heading two characters refer to the class, and the next three character refer to the sub class. The ALTIBASE HDB ODBC diagnosis messages follow the standard of X/Open SQL CAE specifications.

1.2.6 Restriction

Client library of ALTIBASE HDB does not use signal processor. Therefore, if access to network terminates due to external factors, application can be shut down compulsorily by receiving signal of SIGPIPE. You may process it in user application to avoid forced shutdown. And you can not call functions of ALTIBASE HDB client library to process it because program can be stopped. However, you can after processing it.

1.3 Using ODBC

This chapter describes how to use ODBC in Unix and Windows.

1.3.1 Using UNIX ODBC versus Using Windows ODBC

All APIs are used in the same way but different strings(supported by SQLDriverConnect) are used to connect to a database server depending on kinds of API.

1.3.1.1 Unix

```
DSN=host_ip;PORT_NO=20300;UID=SYS;PWD=MANAGER;CONN-
TYPE=1;NLS_USE=US7ASCII
```

The name of shared library is libodbcinst.so (the extension of HP is sl.) by default in Unix-ODBC or iODBC where ODBC library of ALTIBASE HDB is installed. However, if wanting to set its name manually, you can specify it with UNIX_ODBC_INST_LIB_NAME.

ex) You should change setting of environment variable in iODBC.

Unix-ODBC

Extra setting is not a requisite for running Unix ODBC Manager.

iODBC

You should specify the following setting of evironment variable to make an iODBC manager provide an ODBC driver for ALTIBASE HDB.

```
export UNIX_ODBC_INST_LIB_NAME=libiodbcinst.so
```

You can specify the following setting of evironment variable under the necessity.

```
export LD_PRELOAD=/lib/libdemangle.so
```

In addition, if iODBC manager fails to provide an ODBC driver for AIX, you should specify additional setting after checking the following

- You should extract iodbcinst.so. from iodbcinst.a after searching for a path where iODBC library is installed.

  ```
  $> ar -x libiodbcinst.a
  ```

- You should change so.2 into so as file extension.

  ```
  $> mv libiodbcinst.so.2 libiodbcinst.so
  ```

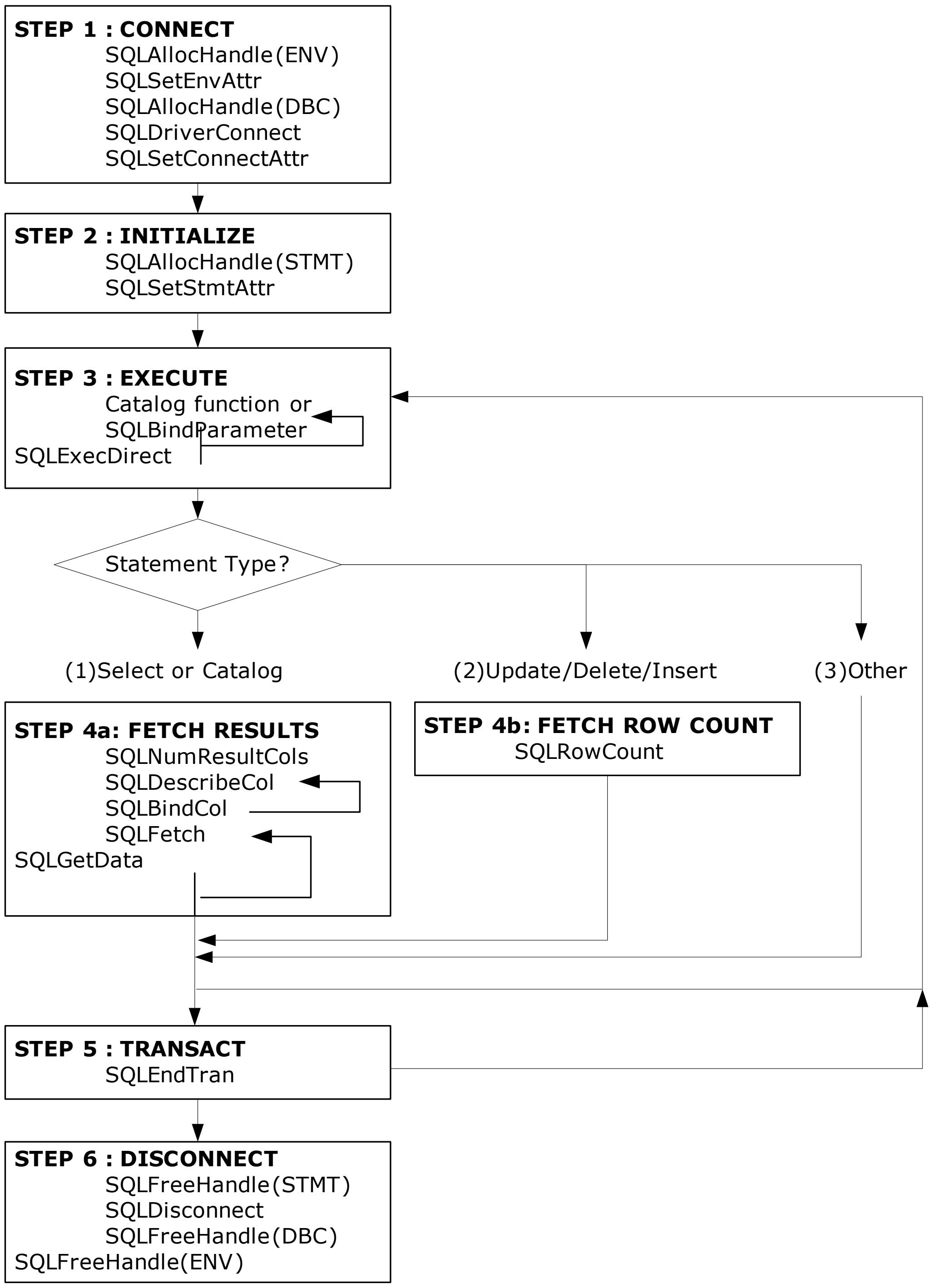
- iODBC shared library search path should be specified by setting the value of the environment variable LIBPATH.

  ```
  Ex) Shared library path is /usr/local/lib :
  export LIBPATH=/usr/local/lib:$LIBPATH
  ```

1.3.1.2 Windows

After installing ODBC Driver, you should specify data source name registered in Setting -> Administrative Tools -> Data Source as DSN=datasource_name.

1.4 Basic Programming Steps



1.4.1 Step 1: Connecting to a Database

The first step is to connect to the database. The following functions are necessary for this step:

```
STEP 1: CONNECT
        SQLAllocHandle()
        SQLSetEnvAttr()
        SQLAllocHandle()
        SQLDriverConnect()
        SQLSetConnectAttr()
```

The first operation to access a database is to allocate the environment handle using SQLAllocHandle ().

An application sets the environmental attribute by calling SQLSetEnvAttr () .

Then, an application allocates the connection handle by using SQLAllocHandle () and calls SQLDriverConnect () to connect to a database.

After connecting successfully, an application can set the connection attributes by using SQLSetConnectAttr ().

1.4.2 Step 2: Initializing an application Status

The second phase is usually to initialize an application as described in the figure below. However each application can have different operations.

```
STEP 2: INITIALIZE
        SQLAllocStmt()
        SQLSetStmtAttr()
```

An application allocates the statement handle by using SQLAllocStmt (). Most applications sets the status of application attributes by using SQLSetStmtAttr ().

1.4.3 Step 3: Executing an SQL statements

The third phase is to build and execute the SQLSTATEments as shown below. The processing types in the phase can be different. An application creates or executes an SQL statement based on an SQL statement requested by users.

```
STEP 3: EXECUTE
        Catalog function or
        SQLBindParameter()
        SQLExecDirect()
```

The SQLExecDirect() function called to execute an SQL statement. In case to execute multiple times for the performance, preparation and execution will be made by SQLPrepare () and SQLExecute respectively.

If an SQL statement includes the arguments, an application will call SQLBindParameter () and bind each argument with an application variable. Before the argument is bound, SQLPrepare () must be executed. After binding is made, SQLExecute () can be executed.

An application can delay execution of an SQL statement and may call the function that returns the database metadata instead.We called those functions as catalog functions

The action of an application depends on the execution type of an SQL statements.

SQL Statement Types	Actions
SELECT or catalog function	Phase 4a : Get the results.
UPDATE, DELETE, or INSERT	Phase 4b : Get the number of affected rows.
Other SQL Statements	Phase 3 : Build and execute an SQL statement. or Phase 5 Commit the transaction.

1.4.4 Step 4a: Fetch the Results

This phase is the step to fetch the results.

```
STEP 4a: FETCH RESULTS
         SQLNumResultCols()
         SQLDescribeCol()
         SQLBindCol()
         SQLFetch()
         SQLGetData()
```

If a statements executed in the Step 3 is SELECT or catalog function, an application calls SQLNumResultCols () to find the number of the columns in the result set. This step is not needed if an application already knows the number of columns in the result set.

Then, an application brings the name of the result set, the data type, and the precision to SQLDescribeCol (). In the same way, if an application already knows this information, this step will not be necessary. Then, an application sends this information to SQLBindCol () that binds an application variables and the columns of the result set.

Then, an application calls SQLFetch () to fetch the first row of data and stores the data in the variables bound to SQLBindCol (). In case there is only one recored in the row, use the SQLGetData () to get the data. To fetch multiple rows of data, an application keeps calling SQLFetch () and SQLGetData ().

An application returns to Step 3 to execute other statements in the same transaction or goes to step 5 to commit or rollback the transaction.

1.4.5 Step 4b: Fetch the Affected Row Count

If a statements executed in Step 3 is UPDATE, INSERT, or DELETE, an application will bring the number of affected rows using SQLRowCount ().

An application goes back to Step 3 to execute other statements in the same transaction, or Step 5 to commit or rollback the transaction.

1.4.6 Step 5: Commit/Rollback a Transaction

In Step 5, an application commits the transaction or calls SQLEndTran () for rollback. An application performs this phase only when the transaction commit mode is in non-auto-commit mode. If the transaction commit mode is auto-commit, the transaction will be automatically reflected immediately when an SQL statements are successfully executed.

To execute an SQL statement in the new transaction, an application shoult to return to Step 3. An application goes to Step 6 to disconnect from the database.

1.4.7 Step 6: Disconnect from the Altibase Database

```
STEP 6: DISCONNECT
        SQLFreeHandle()
        SQLDisconnect()
        SQLFreeConnect()
        SQLFreeEnv()
```

In the final step, an application disconnects from the database. An application calls SQLFreeHandle () and release the handles and resources.

Then, an application disconnects from the database by using SQLDisconnect (), and returns the connection handle by using SQLFreeConnect ().

Lastly, an application returns the environment handle by using SQLFreeEnv () and ends the program.

1.5 Summary of ODBC Functions

Task	Function Name	Purpose
Managing environments and connections	SQLAllocConnect	Obtains an environment, connection, statement, or descriptor handle.
	SQLAllocEnv	Allocates an environment, connection, statement, or descriptor handle.
	SQLAllocStmt	Obtains statement handles and Allocates memory.
	SQLAllocHandle	Initializes of resources, environments, and statement handles and allocates memory
	SQLCloseCursor	This closes cursor and discards pending resuts.
	SQLConnect	Connects to a target database
	SQLDisconnect	Closes the connection. Releases an environment, connection, statement, or descriptor handle.
	SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
	SQLEndTran	Commits or rolls back a transaction.
	SQLFreeConnect	Closes the connection handle, and releases the memory.
	SQLFreeEnv	Closes the environment handle, and releases the memory.
	SQLFreeFailover	Frees the failover handle assigned to SQLAllocFailover
	SQLFreeHandle	Releases the memory allocated to the connection, the handle, and the command.
	SQLFreeStmt	Closes the statement handle, and releases the allocated memory.
	SQLTransact	Commits or releases all changes related to the database.

Task		Function Name	Purpose
SQL Pro-cessing	Reque sting	SQLBindParameter	Binds the parameter to an SQL statement.
		SQLExecDirect	Directly executes an SQL statement.
		SQLExecute	Executes a prepared SQL statement
		SQLNativeSql	This efficiently tests the syntax of SQL state-ments and converts it to that ODBC driver sup-prots.
		SQLParamData	This is used to supply data at statement execu-tion time.
		SQLPrepare	Prepares an SQL statement for later execution
		SQLPutData	This is used to supply data at statement execu-tion time.
	Retriev ing	SQLBindCol	Defines the buffer and the data type to receive the columns of the result set.
		SQLColAttribute	Defines the attributes about the columns of the result set.
		SQLDescribeCol	Checks the metadata about one column in the result set.
		SQLDescribeParam	Checks information related to the parameter marker (?) in the result set.
		SQLERROR	Checks diagnosis messages related to the recently called ODBC function.
		SQLFetch	Returns multiple result rows.
		SQLFetchScroll	The result set the cursor to the desired direction of progress, and get a column to bind
		SQLGetConnectAttr	Returns the properties setting of connection
		SQLGetData	Returns the data of a specified column in the result set
		SQLGetStmtAttr	Returns the attributes related to the current statement handle
		SQLGetTypeInfo	Returns the information about the data type supported by the database.
		SQLNumParams	Returns the number of parameters in an SQL statement.
		SQLNumResultCols	Returns the number of columns in the result set.
		SQLRowCount	Returns the number of rows affected by an insert, update, or delete request.
		SQLMoreResults	Returns whether there is more result

Task	Function Name	Purpose
Setting and retrieving driver attributes	SQLGetEnvAttr	This returns the current setting of an environment attribute.
	SQLGetFunctions	This returns information about whether a driver supports a specific ODBC function.
	SQLSetConnectAttr	Sets the connection attributes.
	SQLSetEnvAttr	Sets the environment attributes
	SQLSetStmtAttr	Sets the statement attributes.
Metadata Processing(catalog functions)	SQLColumns	Returns the list of column names in specified tables.
	SQLForeignKeys	Returns a list of column names that make up foreign keys, if they exist for a specified table.
	SQLGetDescField	This returns a single field of a descriptor record.
	SQLGetDescRec	This returns values of mutiple fields of a descriptor record.
	SQLGetDiagField	Diagnose the result after the function is used
	SQLGetDiagRec	This returns several commonly used fields of a diagnostic record after using the function.
	SQLPrimaryKeys	Returns the list of column names that make up the primary key for a table.
	SQLProcedureColumns	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.
	SQLProcedures	Returns the list of procedure names stored in a specific database.
	SQLSetDescField	This sets the descriptor field.
	SQLSpecialColumns	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
	SQLStatistics	Returns statistics about a single table and the list of indexes associated with the table.
	SQLTablePrivileges	Returns a list of tables and the privileges associated with each table.
	SQLTables	Returns the list of table names stored in a specific database.

2 ODBC Functions

This chapter describes the specifications of ODBC functions.

For each ODBC functions, the following information are described.

- Name of the function and purpose of use
- Function prototype for C/C++ Users
- Arguments list of the function
- Return Values
- Usages of function and notes
- Diagnosis message that can be displayed when an error occurrs in function
- Related Function list
- Example source codes

2.1 SQLAllocConnect

SQLAllocConnect allocates an environment, connection, statement, or descriptor handle. This function allocates the related resources in the environment identified by the connection handles and the input environment handles.

SQLAllocConnect () can be replaced by SQLAllocHandle ().

2.1.1 Syntax

```
SQLRETURN  SQLAllocConnect (
    SQLHENV    env,
    SQLHDBC    *dbc );
```

2.1.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV	env	Input	Environment Handle
SQLHDBC *	dbc	Output	Connection Handle Pointer

2.1.3 Return Values

```
SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_ERROR
```

2.1.4 Description

The ODBC uses the output connection handle to refer to all information related to the connection such as connection status, transaction status, and error information.

If the pointer (dbc) indicating the connection handle refers to the valid connection handle allocated by SQLAllocConnect (), the calling result will change the original value. If it is application programming error, it is not detected by ODBC.

* Call SQLAllocEnv() before calling this function.

2.1.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	Channel initialization error
HY001	Memory allocation error	Failed to allocate the memory for the explicit handle.
HY009	Invalid Arguments used (null pointer).	dbc is a NULL pointer.

2.1.6 Related Functions

`SQLAllocEnv`

`SQLConnect`

`SQLDisconnect`

`SQLFreeConnect`

2.1.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp.

```
/* Memory allocation for the environment */
if (SQLAllocEnv(&env) != SQL_SUCCESS)
{
 printf("SQLAllocEnv error!!\n");
 return SQL_ERROR;
}
/* Memory allocation for the connection */
if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
{
 printf("SQLAllocConnect error!!\n");
 return SQL_ERROR;
}
```

2.2 SQLAllocEnv

SQLAllocEnv allocates the resources related to the environment handles.

SQLAllocEnv () can be replaced by SQLAllocHandle ().

2.2.1 Syntax

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

2.2.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV *	env	Output	Environment handle pointer

2.2.3 Return Values

```
SQL_SUCCESS
```

```
SQL_ERROR
```

2.2.4 Description

One application can use various environment variables.

To use resources of the ODBC, the program that called SQLAllocEnv () must not terminate or get out of the stack. Otherwise, an application may lose the statement handles and other allocated resources.

Before calling SQLAllocConnect () or other ODBC functions, an application must call this function. Then, the env value will be sent to all functions that require the environment handles as input values.

2.2.5 Related Functions

```
SQLAllocConnect
```

```
SQLAllocStmt
```

```
SQLFreeEnv
```

2.2.6 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp.

```
/* Memory allocation for the environment */
if (SQLAllocEnv(&env) != SQL_SUCCESS)
{
 printf("SQLAllocEnv error!!\n");
 return SQL_ERROR;
}
```

2.3 SQLAllocHandle

SQLAllocHandle allocates and initializes the memory for the environment, connection, and statement handles.

2.3.1 Syntax

```
SQLRETURN  SQLAllocHandle (
    SQLSMALLINT    HandleType,
    SQLHANDLE      InputHandle,
    SQLHANDLE      *OutputHandlePtr );
```

2.3.2 Arguments

Data Type	Arguments	In/Out	Description
SQLSMALLINT	HandleType	Input	One of the following handle type is allocated: SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT
SQLHANDLE	InputHandle	Input	If the input HandleType is SQL_HANDLE_ENV, InputHandle will be SQL_Null_Handle. Or if the input HandleTyp is SQL_HANDLE_DBC, it will be the environment handle. In case of SQL_HANDLE_STMT, it will be the connection handle.
SQLHANDLE *	OutputHandlePtr	Output	The pointer of the allocated handle

2.3.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.3.4 Description

SQLAllocHandle () allocates the environment, connection and statement handles to be described in the next paragraph.

This function will replace SQLAllocEnv (), SQLAllocConnect () and SQLAllocStmt () functions. To request the environment handle, an application calls SQLAllocHandle () of which HandleTyp is SQL_HANDLE_ENV and input handle is SQL_Null_Handle. To request the connection handle, an application must call SQLAllocHandle () of which HandleTyp is SQL_HANDLE_DBC and the input handle must be a valid environment handle. To request the statement handle, an application must call SQLAllocHandle () of which HandleTyp is SQL_HANDLE_STMT and the input handle must be a valid connection handle.

One application can allocate multiple environment, connection, and statement handles at one time. However, several environment, connection or statement handle cannot be used at the same time on another thread of one process.

2.3.4.1 Allocation of the Environment Handles

The environment handle provides global information about the validity or activation of the connection handle.

To request an environment handle, an application must call SQLAllocHandle () of which HandleTyp is SQL_HANDLE_ENV and input handle is SQL_Null_Handle. The ODBC allocates the memory needed for environment information and returns the handle related to the *OutputHandle. An application sends the *OutputHandle value to the subsequent callings that require the environment handles.

2.3.4.2 Allocation of the Connection Handles

The connection handle provides information about the validity of the statement handle or activation of the transaction.

To request the connection handle, an application calls SQLAllocHandle () of which HandleTyp is SQL_HANDLE_DBC. InputHandle argument calls SQLAllocHandle () and is set as the returned environment handle. The ODBC allocates the memory necessary for the connection and returns the handle values related to the *OutputHandle. An application sends the *OutputHandle to the subsequent callings that require the connection handle.

2.3.4.3 Allocation of the Statement handles

The statement handle provides command information such as error messages about processing an SQL statement and status state.

To request the statement handle, an application connects to the database and calls SQLAllocHandle () before sending an SQL statement. For this calling, the HandleTyp must be set as SQL_HANDLE_STM and the InputHandle argument must be set as a connection handle to be returned by calling SQLAllocHandle (). The ODBC allocates the memory necessary for the command, connects the statement handle, and returns the handle related to *OutputHandle. An application sends *OutputHandle value to the subsequent callings that require the statement handle.

2.3.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	
HY001	Memory allocation error	Failed to allocate the memory for the explicit handle.
HY009	Invalid arguments used (null pointer).	OutputHandlePtr is a NULL pointer.

2.3.6 Related Functions

```
SQLExecDirect

SQLExecute

SQLFreeHandle

SQLPrepare

SQLSetConnectAttr

SQLSetEnvAttr

SQLSetStmtAttr
```

2.3.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta1.cpp

```
/* Memory allocation for the environment */
if (SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HENV, &env) != SQL_SUCCESS)
{
 printf("SQLAllocEnv error!!\n");
 return SQL_ERROR;
}

/* Memory allocation for the connection */
if (SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc) != SQL_SUCCESS)
{
 printf("SQLAllocConnect error!!\n");
 return SQL_ERROR;
}
```

2.4 SQLAllocStmt

SQLAllocStmt allocates and initializes the memory for the SQL statements. Up to 1024statements are allocated to one connection.

SQLAllocStmt () can be replaced by SQLAllocHandle ().

2.4.1 Syntax

```
SQLRETURN  SQLAllocStmt (
    SQLHDBC    dbc,
    SQLHSTMT   *stmt );
```

2.4.2 Arguments

Data Type	Arguments	In/Out	Description
SQLHDBC	dbc	Input	Connection Handle
SQLHSTMT *	stmt	Output	Pointer of statement handle

2.4.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

In case SQL_ERROR is returned, stmt arguments will be set as SQL_NULL_STMT. An application must set stmt arguments as SQL_NULL_STMT and calls SQLERROR ().

2.4.4 Description

The ODBC relates the descriptors, results and status data with the processed SQL statements by using each statement handle. Each SQL statement must have a statement handle, but other commands can use the handles again.

When this function is called, the database connection used by the databasec must be referred to.

If the input pointer indicates the valid statement handle allocated by the previous calling of SQLAllocStmt (), the original value will be changed according to the result of this calling.

As an application programming error, it is not detected by ODBC.

* Call SQLAllocEnv() before calling this function. This function must be called before other functions

that have SQLPrepare (), SQLExecute (), SQLExecDirect () or statement handle as an input Arguments.

2.4.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	The number of stmt (1024) exceeds
HY001	Memory allocation error	Failed to allocate the memory for the stmt.
HY009	Invalid Arguments used (null pointer).	stmt is a NULL pointer.
HY010	Continuous function error (not connected or disconnected status)	dbc is not connected or disconnected.

2.4.6 Related Functions

```
SQLConnect

SQLFreeStmt
```

2.4.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
/* Memory allocation for a statement */
if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
{
 printf("SQLAllocStmt error!!\n");
 return SQL_ERROR;
}
```

2.5 SQLBindCol

SQLBindCol binds an application variables to the columns of the result sets for all data types.

2.5.1 Syntax

```
SQLRETURN  SQLBindCol (
    SQLHSTMT        stmt,
    SQLSMALLINT     col,
    SQLSMALLINT     cType,
    SQLPOINTER      value,
    SQLLEN          max,
    SQLLEN          *valueLength );
```

2.5.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	col	Input	Column position in the result set to bind. Starts with 1.
SQLSMALLINT	cType	Input/Output(Suspended)	C data type identifier of the *Value buffer. About the OODBC data types See the appendix of this document.
SQLPOINTER	value	Output	Pointer of the buffer to store the data. SQLFetch () returns the data to this buffer.If the value is a NULL pointer, the ODBC will unbind the data buffer for the result set columns. An application unbinds all columns by calling SQLFreeStmt () using SQL_UNBind option. However, if the ValueLength argument is valid even though the value argument is a NULL pointer, an application still have buffer for binding. of the length.

Data Type	Argument	In/Out	Description
SQLLEN	max	Input	Maximum size of the buffer (in bytes). When returning the character data to the *Value, the *Value argument must include space for the NULL-terminatior. Otherwise, the ODBC cuts out the data. In case a fixed length data (integer, date structure, etc) are returned, the ODBC will ignore max. Therefore, a sufficient buffer size must be allocated. Otherwise, the ODBC passes through the end of the buffer and saves the garbage data.
SQLLEN *	valueLength	Input/Output(Suspended)	Is a pointer for the data length or NULL. SQLFetch () function be able to return the length of data or SQL_NULL_DATA

2.5.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.5.4 Description

The pointer value and ValueLength are suspended output variables for this function. Thememory address indicated by this pointer will not be updated until SQLFetch () is called. The position referred to by this pointer must be valid till SQLFetch () is called.

SQLBindCol () binds application variables to the columns of the result set for all data types. When SQLFetch () is called, the data will be sent from the databaseMS to an application.

An application calls SQLBindCol () once for each column. When SQLFetch () is called, the data of each bound column is stored in the address allocated by the value or the ValueLength pointer.

An application can inquire the attributes such as the data type or length of the column by calling SQLDescribeCol () or SQLColAttribute (). This information can be used to indicate the proper data type or to convert the data into another data type.

The columns are identified in a series of numbers from the left to the right. The number of columns in the result set can be decided by setting SQL_DESC_Count in SQLNumResultCols () or fieldIdentifier argument and by calling SQLColAttribute ().

An application may not bind any column. The data in unbound column can be searched by SQLGet-

Data () after SQLFetch () is called. In usuall case SQLBindCol () is more efficient than SQLGetData ().

* To get the data from the buffer identified by this function, SQLBindCol () must be called before SQLFetch ().

2.5.5 Diagnosis

SQLSTATE	Description	Comments
07009	Invalid column number.	col Arguments exceeds the maximum number of columns in the result set.
HY000	General error	
HY001	Memory allocation error	Failed to allocate the memory for the explicit handle.
HY003	An application buffer type is not valid.	cType argument is not valid.

2.5.6 Related Functions

```
SQLDescribeCol

SQLFetch

SQLFreeStmt

SQLGetData

SQLNumResultCols
```

2.5.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex2.cpp

```
sprintf(query,"SELECT id,name,age FROM DEMO_EX2 WHERE id=?");
if (SQLPrepare(stmt,query, SQL_NTS) != SQL_SUCCESS)
{
 execute_err(dbc, stmt, query);
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
}
if (SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
 SQL_C_CHAR, SQL_CHAR,
 8, 0,
 id_in, sizeof(id_in), NULL) != SQL_SUCCESS)
{
 execute_err(dbc, stmt, query);
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
}

/* Set the variable to bring the result of Select. */
```

```
if (SQLBindCol(stmt, 1, SQL_C_CHAR,
 id, sizeof(id), NULL) != SQL_SUCCESS)
{
 printf("SQLBindCol error!!!\n");
 execute_err(dbc, stmt, query);
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
}
if (SQLBindCol(stmt, 2, SQL_C_CHAR,
 name, sizeof(name), NULL) != SQL_SUCCESS)
{
 printf("SQLBindCol error!!!\n");
 execute_err(dbc, stmt, query);
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
}
if (SQLBindCol(stmt, 3, SQL_C_SLONG,
 &age, 0, NULL) != SQL_SUCCESS)
{
 printf("SQLBindCol error!!!\n");
 execute_err(dbc, stmt, query);
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
}
To display the result while the result is available */
printf("id\tName\tAge\tbirth\t\sex\tetc\n");

printf("============================================================
=\n");
for ( i=1; i<=3; i++ )
{
 sprintf(id_in, "%d0000000", i);
 if ( SQLExecute(stmt) != SQL_SUCCESS )
 {
 execute_err(dbc, stmt, "SQLExecute : ");
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
 }
 if ( (rc = SQLFetch(stmt)) != SQL_NO_DATA && rc == SQL_SUCCESS )
 {
printf("%-10s%-20s%-5d%4d/%02d/%02d %02d:%02d:%02d\t%-2d\t",
 id, name, age, birth.year, birth.month, birth.day,
 birth.hour, birth.minute, birth.second, sex);
 if (etc_ind == SQL_NULL_DATA)
 {
 printf("NULL\n");
 }
 else
 {
 printf("%.3f\n", etc);
 }
 }
 else
 {
 execute_err(dbc, stmt, query);
 break;
 }
}
```

2.6 SQLCloseCursor

This closes cursor and discards the suspended results.

2.6.1 Syntax

```
SQLRETURN SQLCloseCurosr (
    SQLHSTMT    stmt);
```

2.6.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	In	Command Handle

2.6.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.6.4 Description

This closes cursor and discars the suspended results. This option has same functionality as using SQL_CLOSE option in SQLFreeStmt(). However, 240000 errors occur if cursor is not open in SQLClose-Corsor().

2.6.5 Diagnostics

SQLSTATE	Description	Comments
HY000	General Error	
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
24000	The state of cursor is incorrect.	No cursor is open in command handle.

2.6.6 Related Function

```
SQLFreeHandle
```

2.7 SQLBindParameter

SQLBindParameter binds the parameter marker of an SQL statement with an application variables. The data is transmitted from an application to the database when SQLExecute () is called.

2.7.1 Syntax

```
SQLRETURN  SQLBindParameter (
    SQLHSTMT        stmt,
    SQLSMALLINT     par,
    SQLSMALLINT     pType,
    SQLSMALLINT     cType,
    SQLSMALLINT     sqlType,
    SQLINTEGER      columnSize,
    SQLSMALLINT     scale,
    SQLPOINTER      value,
    SQLINTEGER      valueMax,
    SQLINTEGER      *valueLength );
```

2.7.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	par	Input	Parameter order. Starting with 1.
SQLSMALLINT	pType	Input	Parameter type. All parameters in an SQL statement must be input variables (SQL_PARAM_INPUT). When executing a stored procedure, arguments can be are input, output, or input/output type variables. (SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, SQL_PARAM_INPUT_OUTPUT)
SQLSMALLINT	cType	Input	C data type of the parameter (SQL_C_CHAR, SQL_C_SBIGINT, etc) See: Appendix of this document
SQLSMALLINT	sqlType	Input	SQL data type of the parameter (SQL_CHAR, SQL_VARCHAR, etc) See: Appendix of this document

Data Type	Argument	In/Out	Description
SQLINTEGER	columnSize	Input	An argument that indicates the precision of a parameter marker. Based on SQL type, it can be used as follows: • SQL_CHAR, SQL_VARCHAR: Indicates the max allowed length of a parameter marker. (If columnSize is 0, the default columnSize is used. For SQL_CHAR and SQL_VARCHAR, their columnSize is 32,000.) • SQL_DECIMAL, SQL_NUMERIC: Indicates the decimal significant digits of a parameter marker. (If columnSize is 0, the default columnSize is used. For both SQL_DECIMAL and SQL_NUMERIC, the columnSize is 38, which is the max number of decimal significant digits.) • SQL_BINARY, SQL_BYTES, SQL_NIBBLE, SQL_VARBIT: Indicates the max allowed length of a parameter marker.(If columnSize is 0, the default columnSize is used. The columnSize for each type is as follows: For SQL_BINARY, SQL_BYTE and SQL_VARBIT, their columnSize is 32000. For SQL_NIBBLE, its columnSize is 254.) • For other types, the user-defined columnSize argument is ignored and the following fixed value is used. SQL_SMALLINT 5 SQL_INTEGER 10 SQL_BIGINT 19 SQL_REAL 7 SQL_FLOAT 38 SQL_DOUBLE 15 SQL_TYPE_DATE 30 SQL_TYPE_TIME 30 SQL_TYPE_TIMESTAMP 30 SQL_INTERVAL 10 SQL_GEOMETRY 3200
SQLSMALLINT	scale	Input	The Scale of *value.
SQLPOINTER	value	Input (Suspended)	The pointer of the actual data about the parameter when SQLExecute () or SQLExecDirect () is called.

Data Type	Argument	In/Out	Description
SQLINTEGER	valueMax	Input/Output	Maximum length of the *Value buffer for the character or binary C data
SQLINTEGER *	valueLength	Input(Suspended)	Pointer of the input/output data length when SQLExecute () or SQLExecDirect () is called

2.7.3 Return Values

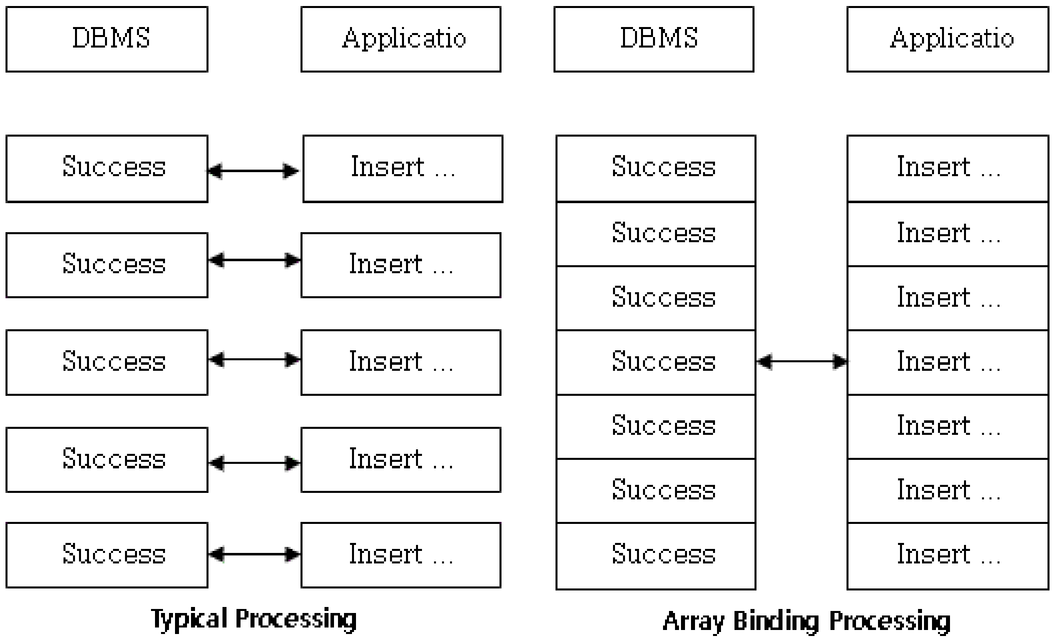
```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.7.4 Description

2.7.4.1 Binding Arrays

The array binding method reduces the network round-trip count and improves the speed by sending the parameter using array types.

The following figure briefly shows how works array binding. Larger amount of data can be sent in a shorter time due to reduced network paging count.



There are two array binding types:

2.7.4.2 Column-wise Parameter Binding

When using column-wise binding, an application binds one or two, or in some cases three, arrays to each column for which data is to be returned. To use a column-wise binding, do the following:

Set SQL_ATTR_PARAM_BIND_TYPE in Arguments Attribute of an application function SQLSetStm-tAttr()

Set SQL_PARAM_BIND_BY_COLUMN in param.

For each column to be bound, the applicatoin performs the following procedures.

1. Allocate the parameter buffer array.
2. Allocate the indicator buffer array.
3. Call SQLBindParameter () with arguments.

cType is C data type of the single element in the parameter buffer array.

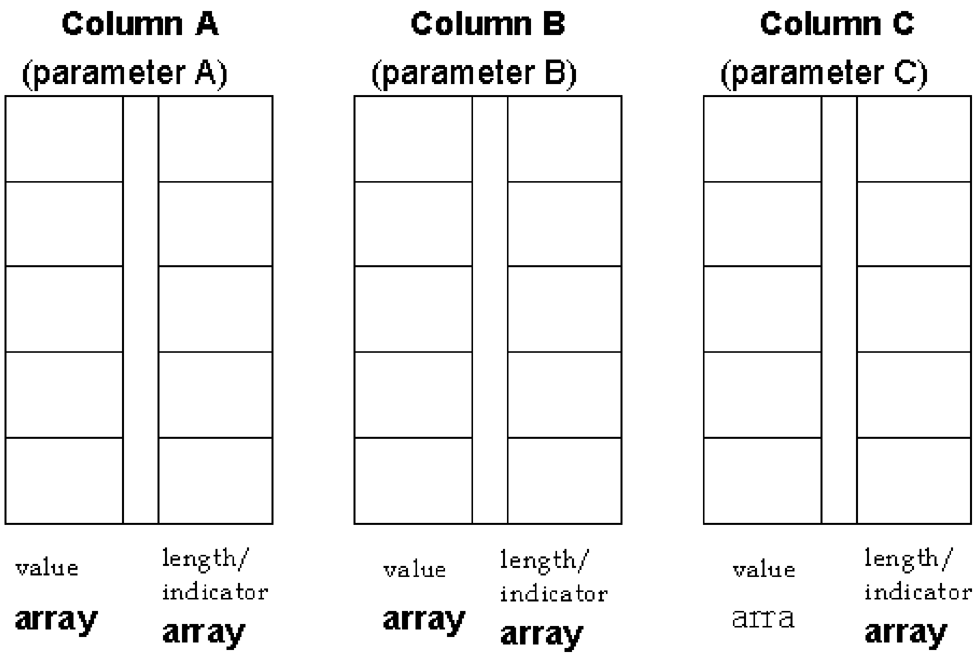
sqlType is the SQL data type of the parameter.

Value is the address of the parameter buffer array.

valueMax is the size of the single element in the parameter buffer array.

valueLength is the address of the length/indicator array.

The following figure shows how the column-wise binding operates for each column.



4. Example

```
#define DESC_LEN 51
#define ARRAY_SIZE 10

SQLCHAR * Statement = "INSERT INTO Parts (PartID, Description, Price) "
                                                     "VALUES (?, ?, ?)";
SQLUINTEGER    PartIDArray[ARRAY_SIZE];
SQLCHAR        DescArray[ARRAY_SIZE][DESC_LEN];
SQLREAL        PriceArray[ARRAY_SIZE];
SQLINTEGER     PartIDIndArray[ARRAY_SIZE], DescLenOrIndArray[ARRAY_SIZE],
 PriceIndArray[ARRAY_SIZE];
SQLUSMALLINT   i, ParamStatusArray[ARRAY_SIZE];
SQLUINTEGER ParamsProcessed;

// Set the SQL_ATTR_PARAM_BIND_TYPE statement attribute to use
// column-wise binding.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_BIND_TYPE,
SQL_PARAMETER_BIND_BY_COLUMN, 0);

// Specify the number of elements in each parameter array.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);

// Specify an array in which to return the status of each set of
// parameters.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);

// Specify an SQLUINTEGER value in which to return the number of sets of
// parameters processed.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);

// Bind the parameters in column-wise fashion.
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG, SQL_INTEGER, 5, 0,
 PartIDArray, 0, PartIDIndArray);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, DESC_LEN -
1, 0,
 DescArray, DESC_LEN, DescLenOrIndArray);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_REAL, 7, 0,
 PriceArray, 0, PriceIndArray);
```

2.7.4.3 Row-wise Binding

When using row-wise binding, an application defines a structure containing one or two, or in some cases three, elements for each column for which data is to be returned.An application performs the next procedures to use the row-wise binding.

Define the array to include the single set of the parameters (including parameters and the length/ indicator buffers).

Set SQL_ATTR_PARAM_BIND_TYPE in argument attributes of function SQLSetStmtAttr (), and set the size of the array including program variables in the argument parameter, and binds the address of each element to the first element of the array.

Call SQLBindParameter () with following arguments.

cType is the component type of the parameter buffer.

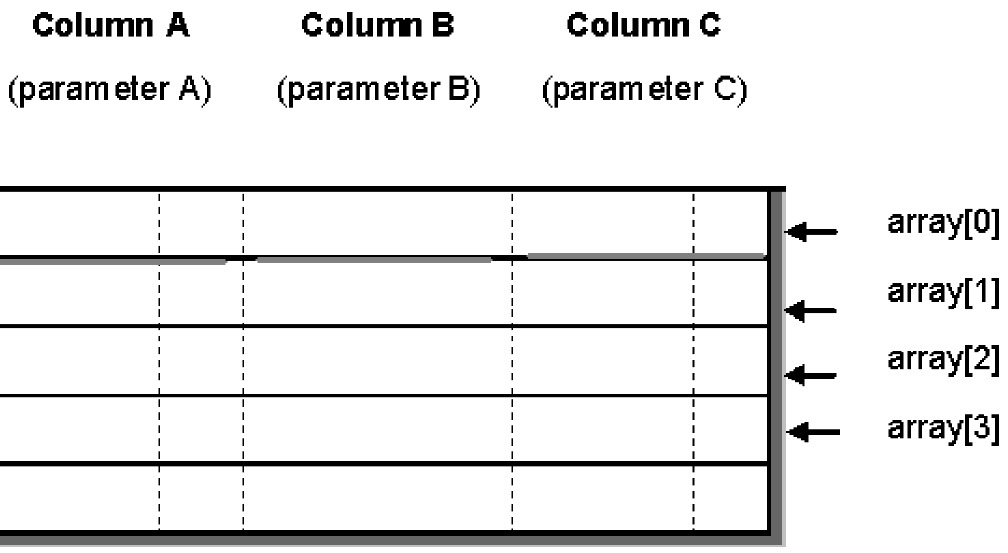
sqlType is the SQL data type of the parameter.

Value is the address of the parameter buffer component in the first array element.

valueMax is the size of the parameter buffer component.

valueLength is the address of the length/indicator to be bound.

The following figure shows how row-wise binding operates.



2.7.5 Example

```
#define DESC_LEN 51
#define ARRAY_SIZE 10

typedef tagPartStruct {
 SQLREAL        Price;
 SQLUINTEGER    PartID;
 SQLCHAR        Desc[DESC_LEN];
 SQLINTEGER     PriceInd;
 SQLINTEGER     PartIDInd;
 SQLINTEGER     DescLenOrInd;
} PartStruct;

PartStruct PartArray[ARRAY_SIZE];
SQLCHAR *       Statement = "INSERT INTO Parts (PartID, Description,
 Price) "
 "VALUES (?, ?, ?)";
SQLUSMALLINT    i, ParamStatusArray[ARRAY_SIZE];
SQLUINTEGER ParamsProcessed;

// Set the SQL_ATTR_PARAM_BIND_TYPE statement attribute to use
// column-wise binding.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_BIND_TYPE, sizeof(PartStruct), 0);

// Specify the number of elements in each parameter array.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);

// Specify an array in which to return the status of each set of
// parameters.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
```

```
// Specify an SQLUINTEGER value in which to return the number of sets of
// parameters processed.
SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);

// Bind the parameters in row-wise fashion.
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG, SQL_INTEGER, 5, 0,
 &PartArray[0].PartID, 0, &PartArray[0].PartIDInd);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, DESC_LEN -
1, 0,
 PartArray[0].Desc, DESC_LEN, &PartArray[0].DescLenOrInd);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_REAL, 7, 0,
 &PartArray[0].Price, 0, &PartArray[0].PriceInd);
```

2.7.6 Constraints

For SQL_BINARY, SQL_BYTES, SQL_NIBBLE and SQL_VARBIT types, the buffer size and column size must be specified.

For SQL_CHAR and SQL_VARCHAR types, the default precision is the max size that a column can have. For SQL_NUMERIC and SQL_NUMBER types, the precision is 38.

2.7.7 Diagnosis

SQLSTATE	Description	Comments
07006	Violation of the limited data type attributes	A cType data type cannot be converted into a sqlType data type.
07009	Invalid number	Indicated par value is smaller than 1.
HY000	General error	
HY001	Memory allocation error	Failed to allocate the memory for the explicit handle.
HY003	An application buffer type is not valid.	A cType value is invalided C data type.
HY009	Invalid pointer used (null pointer)	valueLength is a NULL pointer and pType is not SQL_PARAM_OUTPUT.
HY090	Invalid buffer length	valueMax value is smaller than 0 or higher than 64K
HY105	Wf73 Invalid parameter type	pType is invalided value (in, out, inout)

2.7.8 Related Functions

`SQLExecDirect`

`SQLExecute`

`SQLFreeStmt`

2.7.9 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex2.cpp

```
sprintf(query,"INSERT INTO DEMO_EX2 VALUES( ?, ?, ?, ?, ?, ? )");

/* Prepare for a statement and bind the variable. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
if (SQLBindParameter(stmt,1, /* the sequence of host variables indicated by
?, starting with 1 */
   SQL_PARAM_INPUT, /* Indicates in, out, and inout. */
   SQL_C_CHAR,      /* c type of the variable to bind */
   SQL_CHAR,         /* Data type of the corresponding column in the database
char (8)*/
 8,             /* precision of the column type upon creation of the table */
 0,             /* Scale of the column type upon creation of the table */
 id,            /* Pointer of the buffer to be bound */
 sizeof (id),   /* Size of the buffer to be bound */
 &id_ind        /* indicator */
 ) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
if (SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
    SQL_C_CHAR, SQL_VARCHAR,
    20, /* varchar(20) */
    0,
    name, sizeof(name), &name_ind) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
if (SQLBindParameter(stmt, 3, SQL_PARAM_INPUT,
    SQL_C_SLONG, SQL_INTEGER,
    0, 0, &age,
    0,/* Not used when the buffer to be bound is the fixed size type. */
    NULL) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
if (SQLBindParameter(stmt, 4, SQL_PARAM_INPUT,
    SQL_C_TYPE_TIMESTAMP, SQL_DATE,
    0, 0, &birth, 0, NULL) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
if (SQLBindParameter(stmt, 5, SQL_PARAM_INPUT,
    SQL_C_SSHORT, SQL_SMALLINT,
    0, 0, &sex, 0, NULL) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
if (SQLBindParameter(stmt, 6, SQL_PARAM_INPUT,
    SQL_C_DOUBLE, SQL_NUMERIC,
    10, 3, &etc, 0, &etc_ind) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
/* Execute the prepared statement. */

sprintf(id, "10000000");
sprintf(name, "name1");
age = 28;
      birth.year=1980;birth.month=10;birth.day=10;
      birth.hour=8;birth.minute=50;birth.second=10;
sex = 1;
etc = 10.2;
id_ind = SQL_NTS;          /* id => NULL terminated string */
name_ind = 5;              /* name => length=5 */
/* etc is the fixed size type. Therefore, it will be ignored unless the indi-
cator is SQL_NULL_DATA. */
if (SQLExecute(stmt) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
```

2.8 SQLColAttribute

SQLColAttribute brings the attributes for the column of the result set, and judges the count of columns.

SQLColAttributeW() as a Unicode string supports same execution as SQLColAttribute().

2.8.1 Syntax

2.8.1.1 64 bit Windows

```
SQLRETURN  SQLColAttribute (
    SQLHSTMT        stmt,
    SQLSMALLINT     columnNumber,
    SQLSMALLINT     fieldIdentifier,
    SQLCHAR         *charAttributePtr,
    SQLSMALLINT     bufferLength,
    SQLSMALLINT     *stringLengthPtr,
    SQLLEN          *numericAttributePtr );
```

2.8.1.2 Other Platforms

```
SQLRETURN  SQLColAttribute (
    SQLHSTMT        stmt,
    SQLSMALLINT     columnNumber,
    SQLSMALLINT     fieldIdentifier,
    SQLCHAR         *charAttributePtr,
    SQLSMALLINT     bufferLength,
    SQLSMALLINT     *stringLengthPtr,
    SQLPOINTER      *numericAttributePtr );
```

2.8.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	columnNumber	Input	The column position in the result set. Starts with 1.

Data Type	Argument	In/Out	Description
SQLSMALLINT	fieldIdentifier	Input	Information identifier to know: SQL_DESC_CASE_SENSITIVE, SQL_DESC_CATALOG_NAME, SQL_DESC_COUNT, SQL_DESC_DISPLAY_SIZE, SQL_DESC_LABEL, SQL_DESC_LENGTH, SQL_DESC_NAME, SQL_DESC_NULLABLE, SQL_DESC_PRECISION, SQL_DESC_SCALE, SQL_DESC_SCHEMA_NAME, SQL_DESC_TABLE_NAME, SQL_DESC_TYPE, SQL_DESC_TYPE_NAME, SQL_DESC_UNSIGNED
SQLCHAR *	charAttrib-utePtr	Output	Buffer pointer to store data to be returned when fieldIdentifier in column-Number is the character string. If field value is an integer, it is not used.
SQLSMALLINT	bufferLength	Input	The character number of *charAttributeP-trIf *charAttributePtr is an integer, this field is ignored.
SQLSMALLINT *	stringLength-Ptr	Output	Pointer to a buffer in whih to return the total number of bytes (excluding the null-termination byte) available to return in *charAttributePtr.
SQLINTEGER *	numericAt-tributePtr	Output	Pointer of the integer buffer to which the value of fieldIdentifier field in column-Number row is returned.

2.8.3 Return Values

```
SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_ERROR
```

2.8.4 Description

Instead of returning a specified arguments set such as SQLDescribeCol (),using SQLColAttribute () the attributes for a specified column can be defined. In case the required information is a string type, it will be returned to charAttributePtr. In case the required information is numeric type, it will be returned to numericAttributePtr.

The column is identified by its position (from left to the right, starting with 1).

Call SQLNumResultCols () before calling SQLColAttribute () to check whether the result set exists.

SQLDescribeCol () must be called before SQLBindCol () in case an application does not know about column attributes such as data types, length, etc.

2.8.4.1 fieldIdentifier Descriptor Types

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

Descriptor	Data Type	Description
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	Discrimination of upper and lower characters
SQL_DESC_CATALOG_NAME	SQLCHAR *	Catalog of the table including columns
SQL_DESC_COUNT	SQLINTEGER	The column number of the result set is returned.
SQL_DESC_DISPLAY_SIZE	SQLINTEGER	The maximum number of characters to dis-play the column data
SQL_DESC_LABLE	SQLCHAR *	Column label or title
SQL_DESC_LENGTH	SQLINTEGER	Data bytes related to the column
SQL_DESC_NAME	SQLCHAR *	Name of the column
SQL_DESC_NULLABLE	SQLINTEGER	Whether or not to NULL Yes – SQL_NULLABLE No – SQL_NO_NULLS
SQL_DESC_PRECISION	SQLINTEGER	Precision of the column
SQL_DESC_SCALE	SQLINTEGER	Decimal point attributes of the column
SQL_DESC_SCHEMA_NAME	SQLCHAR *	Schema of the table including the columns
SQL_DESC_TABLE_NAME	SQLCHAR *	Table Name
SQL_DESC_TYPE	SQLINTEGER	SQL data type
SQL_DESC_TYPE_NAME	SQLCHAR *	Database type name
SQL_DESC_UNSIGNED	SQLINTEGER	Inspection of column items

2.8.5 Diagnosis

SQLSTATE	Description	Comments
07009	Invalid column number	columnNumber is 0 or higher than the number of columns in the result set.

SQLSTATE	Description	Comments
HY000	General error	

2.8.6 Related Functions

SQLBindCol

SQLDescribeCol

SQLFetch

2.8.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta8.cpp

```
sprintf(query,"SELECT * FROM DEMO_META8");
if (SQLExecDirect(stmt,query, SQL_NTS) != SQL_SUCCESS)
{
   execute_err(dbc, stmt, query);
   SQLFreeStmt(stmt, SQL_DROP);
   return SQL_ERROR;
}
   SQLNumResultCols(stmt, &columnCount);
   for ( i=0; i<columnCount; i++ )
   {
      SQLColAttribute(stmt, i+1,
                      SQL_DESC_NAME,
                      columnName, sizeof(columnName), &columnNameLength,
                      NULL);
      SQLColAttribute(stmt, i+1,
                      SQL_DESC_TYPE,
                      NULL, 0, NULL,
                      &dataType);
      SQLColAttribute(stmt, i+1,
                      SQL_DESC_PRECISION,
                      NULL, 0, NULL,
                      &columnPrecision);
      SQLColAttribute(stmt, i+1,
                      SQL_DESC_SCALE,
                      NULL, 0, NULL,
                      &scale);
      SQLColAttribute(stmt, i+1,
                      SQL_DESC_NULLABLE,
                      NULL, 0, NULL,
                      &nullable);
   }
```

2.9 SQLColumns

SQLColumns retrieves column information of a specified table as an result set format.

SQLColumnsW() as a Unicode string supports same execution as SQLColumns().

2.9.1 Syntax

```
SQLRETURN  SQLColumns (
    SQLHSTMT       stmt,
    SQLCHAR        *cName,
    SQLSMALLINT    cNameLength,
    SQLCHAR        *sName,
    SQLSMALLINT    sNameLength,
    SQLCHAR        *tName,
    SQLSMALLINT    tNameLength,
    SQLCHAR        *colName,
    SQLSMALLINT    colNameLength );
```

2.9.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR*	cName	Input	Catalog Name
SQLSMALLINT	cNameLength	Input	The character number of **cName*
SQLCHAR *	sName	Input	Name of the schema to retrieve
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	tName	Input	Table name to retrieve
SQLSMALLINT	tNameLength	Input	The length, in bytes, of **tName*
SQLCHAR *	colName	Input	Column to retrieve
SQLSMALLINT	colName-Length	Input	The length, in bytes, of **colName*

2.9.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE
```

```
SQL_ERROR
```

2.9.4 Description

This function is usually used before execution of the command to get column information in the database catalog. SQLColumns () can be used to retrieve all data types returned by SQLTables (). On the contrary, SQLColAttribute () and SQLDescribeCol () functions describe columns of the result set and SQLNumResultCols() returns the number of columns in the result set.

SQLColumns () returns the results in the standard result set format sorted by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and ORDINAL_POSITION.

Some of columns returned by SQLStatistics () are not returned by SQLColumns (). For example, SQL-Columns () does not return index columns created by expressions such as “Salary + Benefits” or “DEPT = 0012” and the filter.

2.9.4.1 Columns Returned by SQLColumns ()

The following table lists the columns of the result sets.

Name	No.	Data Type	Description
TABLE_CAT	1	VARCHAR	Always return NULL
TABLE_SCHEM	2	VARCHAR	Schema name; NULL in case not suitable for the database
TABLE_NAME	3	VARCHAR (NOT NULL)	Table Name
COLUMN_NAME	4	VARCHAR (NOT NULL)	Column Name.As for the unnamed string, the ODBC driver returns the empty character string.
DATA_TYPE	5	SMALLINT (NOT NULL)	SQL data type
TYPE_NAME	6	VARCHAR (NOT NULL)	Character string representing the name of the data type corresponding to DATA_TYPE.
COLUMN_SIZE	7	INTEGER	String Size. NULL will be returned when the string size is not proper.
BUFFER_LENGTH	8	INTEGER	The maximum buffer length to store the data
DECIMAL_DIGITS	9	SMALLINT	NULL will be returned when the data type cannot apply the decimal points of the string and the decimal points.

Name	No.	Data Type	Description
NUM_PREC_RADIX	10	SMALLINT	In case of the numeric data type, it is 10: For COLUMN_SIZE and DECIMAL_DIGIT, decimal digits allowable in this string is given. For example, DECIMAL(12,5) string can return NUM_PREC_RADIX 10, COLUMN_SIZE 12, and DECIMAL_DIGITS 5.
NULLABLE	11	SMALLINT (NOT NULL)	SQL_NO_NULLS when the column is not allowed NULL or SQL_NULLABLE when NULL is allowed.
REMARKS	12	VARCHAR	Description of the column
COLUMN_DEF	13	VARCHAR	Default value of the column
SQL_DATA_TYPE	14	SMALLINT (NOT NULL)	SQL data type
SQL_DATETIME_SUB	15	SMALLINT	Subtype code for the data type. NULL is returned for other data types.
CHAR_OCTET_LENGTH	16	INTEGER	Maximum digits of the character of binary data-type string. For other data types, NULL will be returned.
ORDINAL_POSITION	17	INTEGER (NOT NULL)	Column order of the table. The first column number is 1 in the table.
IS_NULLABLE	18	VARCHAR	NO : When the column does not include NULL:YES : When the column includes NULL:

2.9.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel error before the function processing is completed between the ODBC and the database.
HY000	General error	

2.9.6 Related Functions

```
SQLBindCol

SQLFetch

SQLStatistics
```

```
SQLTables
```

2.9.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta2.cpp

```
if (SQLColumns(stmt,NULL, 0,
               NULL, 0,
               "DEMO_META2", SQL_NTS,
               NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLColumns");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
SQLBindCol(stmt, 1, SQL_C_CHAR, szCatalog, STR_LEN,&cbCatalog);
SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, STR_LEN,&cbTableName);
SQLBindCol(stmt, 4, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
SQLBindCol(stmt, 5, SQL_C_SSHORT, &DataType, 0, &cbDataType);
SQLBindCol(stmt, 6, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
SQLBindCol(stmt, 7, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
SQLBindCol(stmt, 8, SQL_C_SLONG, &BufferLength, 0, &cbBufferLength);
SQLBindCol(stmt, 9, SQL_C_SSHORT, &DecimalDigits, 0, &cbDecimalDigits);
SQLBindCol(stmt, 10, SQL_C_SSHORT, &NumPrecRadix, 0, &cbNumPrecRadix);
SQLBindCol(stmt, 11, SQL_C_SSHORT, &Nullable, 0, &cbNullable);
SQLBindCol(stmt, 17, SQL_C_SLONG, &OrdinalPosition, 0, &cbOrdinalPosition);
SQLBindCol(stmt, 18, SQL_C_CHAR, szIsNullable, STR_LEN, &cbIsNullable);

while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
{
   if ( rc == SQL_ERROR )
   {
      execute_err(dbc, stmt, "SQLColumns:SQLFetch");
      break;
   }
  printf(...);
 }
```

2.10 SQLConnect

SQLConnect connects the ODBC with the database. The connection handle refers to all data related to the database connection including connection status, transaction status, and error data.

SQLConnectW() as a Unicode string supports same execution as SQLConnect().

2.10.1 Syntax

```
SQLRETURN  SQLConnect (
    SQLHDBC        dbc,
    SQLCHAR        *db,
    SQLSMALLINT    dbLength,
    SQLCHAR        *usr,
    SQLSMALLINT    usrLength,
    SQLCHAR        *pwd,
    SQLSMALLINT    pwdLength );
```

2.10.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	Input	Connection Handle
SQLCHAR *	Db	Input	Host IP
SQLSMALLINT	dbLength	Input	The character number of * *db*
SQLCHAR *	usr	Input	User Identifier
SQLSMALLINT	usrLength	Input	The character number of * *usr*
SQLCHAR *	pwd	Input	Authentication character string(password)
SQLSMALLINT	pwdLength	Input	The character number of * *pwd*

2.10.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.10.4 Description

The input length Arguments (dbLength, usrLength, pwdLength) can be set to actual length of the related data. SQL_NTS to indicate that the related data terminated with NULL, or a length value that does not include a NULL termination character can be set.

* SQLAllocConnect () must be called before this function.

This function must be called before SQLAllocStmt ().

Informations such as IP address, user name, and password can be set by SQLSetConnectAttr () as an argument string.

You set the parameters excepting dbc on null or zero in the distributed transaction(Reference the SQLSetConnectAttr).

2.10.5 Diagnosis

SQLSTATE	Description	Comments
08001	Cannot be connected to the server.	ODBC cannot establish connection with the database.
08002	The connection name is already in use.	The corresponding dbc is already connected to the database.
08S01	Communication channel error	Communication channel error before the function processing is completed between the ODBC and the database.
HY000	General error	The character set does not exist.
HY001	Memory allocation error	Cannot allocate the requested memory for the ODBC to execute the function and complete execution.

2.10.6 Related Functions

`SQLAllocHandle`

`SQLDisconnect`

`SQLDriverConnect`

`SQLSetConnectAttr`

2.11 SQLDescribeCol

SQLDescribeCol returns the name of the column, data type, decimal value, and NULLability of columns from the result set.

SQLDescribeColW() as a Unicode string supports same execution as SQLDescribeCol().

2.11.1 Syntax

```
SQLRETURNSQLDescribeCol (
    SQLHSTMT        stmt,
    SQLSMALLINT     col,
    SQLCHAR         *name,
    SQLSMALLINT     nameMax,
    SQLSMALLINT     *nameLength,
    SQLSMALLINT     *type,
    SQLINTEGER      *precision,
    SQLSMALLINT     *scale,
    SQLSMALLINT     *nullable );
```

2.11.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	col	Input	Order of the parameter marker. Starts with 1
SQLCHAR*	name	Output	Pointer of the column name
SQLSMALLINT	nameMax	Input	The character number of the * *Name*
SQLSMALLINT *	nameLength	Output	The length of the *name (excluding NULL-termination byte)
SQLSMALLINT *	type	Output	Pointer of the SQL data type in the column
SQLINTEGER *	precision	Output	Pointer of column size in databaseThe ODBC returns 0 ,when pointer column size cannot be decided
SQLSMALLINT *	scale	Output	Pointer of the number of decimal values in databaseIf the number of decimal values in the database cannot be decided or is not proper, the ODBC will return 0.

Data Type	Argument	In/Out	Description
SQLSMALLINT *	nullable	Output	The pointer of the value that indicates whether the column allows NULL. SQL_NO_NULLS: The column does not allow NULL data. SQL_NULLABLE: The column allows NULL data.

2.11.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.11.4 Description

An application usually calls SQLPrepare (), and calls SQLDescribeCol () before SQLExecute (). An application can call also SQLDescribeCol () after calling SQLExecDirect ().

SQLDescribeCol () searches names, types, and lengths of the columns created by SELECT statements. If the column is an expression, *name will be also an expression.

2.11.5 Diagnosis

SQLSTATE	Description	Comments
01004	String data, right-truncated.	If the buffer *name is not long enough to return the entire column name, the length of the full column name will be returned as **nameLength*.
07009	Invalid descriptor index	The col value is 0. The identified col value is higher than the number of columns in the result set.
HY000	General error	
HY090	Invalid string or buffer length	The identified nameMax is smaller than 0.

If SQLDescribeCol () is called after SQLPrepare () and before SQLExecute (), all SQLSTATE that can be returned by SQLPrepare () or SQLExecute () can be returned.

2.11.6 Related Functions

SQLBindCol

SQLColAttribute

SQLFetch

SQLNumResultCols

SQLPrepare

2.11.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta2.cpp

```
sprintf(query,"SELECT * FROM DEMO_EX1");
if (SQLExecDirect(stmt,query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
SQLNumResultCols(stmt, &columnCount);
for ( i=0; i<columnCount; i++ )
{
    SQLDescribeCol(stmt, i+1, columnName, sizeof(columnName),
                   &columnNameLength, &dataType,
                   &columnSize, &scale, &nullable);
}
```

2.12 SQLDescribeParam

SQLDescibe returns the SQL data types of the columns related to the parameter marker (?) of the dynamic SQL statements, size, data types, expressions of the corresponding parameter markers, number of decimal values, and the NULLability.

2.12.1 Syntax

```
SQLRETURNSQLDescribeParam (
    SQLHSTMT        stmt,
    SQLSMALLINT     iparam,
    SQLSMALLINT     *type,
    SQLINTEGER      *size,
    SQLSMALLINT     *decimaldigit,
    SQLSMALLINT     *nullable );,
```

2.12.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	iparam	Input	Order of the parameter marker, starting with 1
SQLSMALLINT *	type	Output	SQL data type pointer of the parameter
SQLINTEGER *	size	Output	SQL data type pointer of the parameter. Column size or expression pointer of the corresponding parameter
SQLSMALLINT *	decimaldigit	Output	Number of decimal values of the column, expression pointer of the corresponding parameter
SQLSMALLINT *	- NULLable	Output	Pointer of the value that shows whether NULL is allowed for the parameter or not.

2.12.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.12.4 Description

Parameter *iparam* is identified by the number. It is numbered from the left to the right starting with 1.

* SQLPrepare () must be called before this function.

Before SQLBindParameter (), SQLDescribeParam () must be called.

* *Types, sizes, decimal digits, and NULLable* of the parameter have the following limitations:

type: SQL_VACHAR

size: 4000

decimaldigit: 0

nullable: SQL_NULLABLE_UNKNOWN , The ODBC Drive cannot decide whether the parameter allows NULL data.

2.12.5 Diagnosis

SQLSTATE	Description	Comments
07009	Invalid column number	*iparam* is out of the entire argument range.
HY010	Error in function-calling order	Called before SQLPrepare () / SQLExecDirect ().

2.12.6 Related Functions

```
SQLExecDirect

SQLNumParams

SQLPrepare
```

2.12.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_info2.cpp

```
SQLPrepare(hstmt, Statement, SQL_NTS);

// Check to see if there are any parameters. If so, process them.
SQLNumParams(hstmt, &NumParams);
if (NumParams) {
 // Allocate memory for three arrays. The first holds pointers to buffers in w
hich
 // each parameter value will be stored in character form. The second con-
tains
```

```
 the
 // length of each buffer. The third contains the length/indicator value for
each
 // parameter.
 PtrArray = (SQLPOINTER *) malloc(NumParams * sizeof(SQLPOINTER));
 BufferLenArray = (SQLINTEGER *) malloc(NumParams * sizeof(SQLINTEGER));
 LenOrIndArray = (SQLINTEGER *) malloc(NumParams * sizeof(SQLINTEGER));

 for (i = 0; i < NumParams; i++) {
 // Describe the parameter.
 SQLDescribeParam(hstmt, i + 1, &DataType, &ParamSize, &DecimalDigits,
&Nullable);

 // Call a helper function to allocate a buffer in which to store the parame-
ter
 // value in character form. The function determines the size of the buffer fr
om
 // the SQL data type and parameter size returned by SQLDescribeParam and
returns
 // a pointer to the buffer and the length of the buffer.
 PtrArray[i] = (char*)malloc(ParamSize);
 BufferLenArray[i] = SQL_NTS;
 // Bind the memory to the parameter. Assume that we only have input parame-
ters.
 SQLBindParameter(hstmt, i + 1, SQL_PARAM_INPUT, SQL_C_CHAR, DataType, Param-
Size,
 DecimalDigits, PtrArray[i], BufferLenArray[i],
 &LenOrIndArray[i]);

 // Prompt the user for the value of the parameter and store it in the memory
 // allocated earlier. For simplicity, this function does not check the value
 // against the information returned by SQLDescribeParam. Instead, the driver
does

 // this when a statement is executed.
 strcpy((char*)PtrArray[i], "AAAAAAA");
 BufferLenArray[i] = 7;
 }
}
```

2.13 SQLDisconnect

SQLDisconnect colses a connection and releases the handles for the connection.

2.13.1 Syntax

```
SQLRETURN  SQLDisconnect (
    SQLHDBC    dbc );
```

2.13.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	Input	Connection Handle

2.13.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.13.4 Description

If an application calls SQLDisconnect () before releasing the statement handles related to the connection, the connection with the database will be closed.

When SQL_SUCCESS_WITH_INFO is returned, it means that database is successfully disconnected but additional errors or specified implementation program data exist. The cases are as follows:

An error occurred after disconnection. When connection is not established due to other problems such as communication failure

An application can use the databasec to request another SQLConnect () after successfully calling SQLDisconnect ().

* To connect another database after calling this function, call SQLConnect () or SQLDriverConnect ().

2.13.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.13.6 Related Functions

```
SQLAllocHandle

SQLConnect

SQLDriverConnect

SQLEndTran

SQLFreeConnect
```

2.13.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
SQLDisconnect( dbc );
```

2.14 SQLDriverConnect

SQLDriverConnect () is alternative to SQLConnect (). This function supports the connection string that requires more information than three arguments (DSN, user ID, and password) of SQLConnect ().

SQLDriverConnect () provides connection attributes as follows:

host IP or host name
one or more user IDs
one or more passwords
connection method
port number
NLS_USE
TIMEOUT setting

SQLDriverConnectW() as a Unicode string supports same execution as SQLDriverConnect().

2.14.1 Syntax

```
SQLRETURN  SQLDriverConnect (
    SQLHDBC         dbc,
    SQLPOINTER      windowHandle,
    SQLCHAR         *InConnectionString,
    SQLSMALLINT     length1,
    SQLCHAR         *OutConnectionString,
    SQLSMALLINT     bufferLength,
    SQLSMALLINT     *strLength2Ptr,
    SQLSMALLINT     DriverCompletion );
```

2.14.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDBC	Dbc	Input	Connection Handle
SQLPOINTER	windowHandle	Input	Not used.
SQLCHAR*	InConnectionString	Input	A complete connection string, a partial connection string, or an empty character string. For more information see the following description section
SQLSMALLINT	length1	Input	The length, in bytes, of the contents of the *InConnectionString* argument
SQLCHAR *	OutConnectionString	Output	Not used.
SQLSMALLINT	bufferLength	Input	Not used.

Data Type	Argument	In/Out	Description
SQLSMALLINT *	strLength2Ptr	Output	Not used.
SQLSMALLINT	DriverCompletion	Input	Not used.

2.14.3 Return Values

`SQL_SUCCESS`

`SQL_SUCCESS_WITH_INFO`

`SQL_NO_DATA_FOUND`

`SQL_INVALID_HANDLE`

`SQL_ERROR`

2.14.4 Description

This connection string is used to transmit one or more values needed for completion of the connection. The contents of the connection string and DriverCompletion determine the connection method.

Each keyword has following attributes.

- DSN

  It can be a host name, an IPv4 address, or an IPv6 address. An IPv6 address must be enclosed by a left square bracket([) and a right square bracket(]). For example, in the case of localhost (meaning this computer), `localhost` can be specified as the host name, `127.0.0.1` as the IPv4 address, or `[::1]` as the IPv6 address. For more information about the IPv6 address notation, please refer to the *ALTIBASE HDB Administrator's Manual*.

- UID

  User Id

- PWD

  Password .If there is no password for the user id, no data will be defined.

- CONNTYPE

  Connection methods (1 : TCP/IP, 2 : UNIX DOMAIN, 3 : IPC)

- PRIVILEGE

  sys account could be granted sysdba privilege for remote access.

  Access with (sysdba) sysdba privilege is available through TCP/IP and UNIX DOMAIN, but remote access is only through TCP/IP.

- PORT_NO

  Connection port number

- NLS_USE

  NLS language. such as US7ASCII for English, KO16KSC5601 for Korean

- NLS_NCHAR_LITERAL_REPLACE

  This checks to use NCHAR with analyzing SQL statements. (0: doesn't analyze SQL statments, 1: analyzes SQL statements.

  This causes worse performance.)

- TIMEOUT

  Time waiting for server connection attemption. The default value is 3 seconds.

- CONNECTION_TIMEOUT

  Set timeout value to prevent blocking that may occur in select() or poll() in an unstable network

- DATE_FORMAT

  Date Format. The default date format is YYYY/MM/DD HH:MI:SS.

- ALTERNATESERVER

  Specifys the IP addresses and the connection port numbers of the alternative servers to use the connection failover feature. For example, the format is (192.168.1.2:20300,192.168.1.3:20300).

- IpcFilePath

  Client can't connect to server through IPC in Unix because they have different socket paths if having different ALTIBASE_HOMEs each other. You can communicate with Unix domain by using ALTIBASE_HOME/trc/cm-ipc, and then you can get information of shared memory.

- APP_INFO

  This sets information of program you connect to, and you can check this with the following statement.select CLIENT_APP_INFO from v$session;

- AUTOCOMMIT

  This denotes to set AUTOCOMMIT mode (ON or OFF).

- LONGDATACOMPAT

  This enables BLOB and CLOB to be types for ODBC when you connect to ODBC with data whose type is BLOB or CLOB (YES or No).

- PREFER_IPV6

  This attribute determines the IP address to be connected first when a host name is given for DSN attribute.

If this attribute is set to TRUE and a host name is given for DSN attribute, this means that resolving the host name to the IPv6 address is prefered. If this attribute is omitted or it is set to FALSE, a client application connects to the IPv4 address by default. If it fails to connect to the prefered IP version address, an attempt is made to connect using the other IP version address.

If alternative servers in ALTERNATESERVER attribute are specified as a host name, the value set for the PREFER_IPV6 attribute is applied to the alternative servers.

InConnectionString :

DSN=192.168.1.11;UID=SYS;PWD=MANAGER;CONNTYPE=1;NLS_USE=KO16KSC5601;PORT_NO=20202;TIMEOUT=5;CONNECTION_TIMEOUT=10;DATE_FORMAT=DD-MON-YYYY;IPCFILEPATH="…/cm-ipc;PREFER_IPV6=FALSE”

2.14.5 Restriction

- You can access to database remotely with sysdba privilege, but can't start up database.
- When you try to contact local server with sysdba privilege through TCP/IP in the state of no remote access specified in REMOTE_SYSDBA_ENABLE and no loop back (127.0.0.1), local server can regard this trial as remote access and then can't allow it.

```
$ isql -u sys -p manager -s 192.168.3.91 -port 11515 -sysdba
ISQL_CONNECTION = TCP, SERVER = 192.168.3.91, PORT_NO = 11515

[ERR-410C9 : Remote access as SYSDBA not allowed]
```

- If the values of PORT_NO and NLS_USE aren't specified in connection string, you should set same value of environment variable below as the value specified in property file. However, if you want to set national-character figurative constant instead of environment variable, you should specify ALTIBASE_NLS_NCHAR_LITERAL_REPLACE as 1. This causes parsing additionally.

```
export ALTIBASE_PORT_NO=20300
export ALTIBASE_NLS_USE=US7ASCII
export ALTIBASE_NLS_NCHAR_LITERAL_REPLACE=0
```

2.14.6 Diagnosis

SQLSTATE	Description	Comments
08001	Unable to establish connection.	The client is unable to connect to a server.
08002	The connection name is in use.	The connection handle dbc is already connected to the database and still open.

SQLSTATE	Description	Comments
08S01	Communication channel error	The communication channel between the driver and the database to which the driver was attempting to connect failed before the function completed processing
HY000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined.
HY001	Memory allocation error	Cannot allocate the requested memory for the ODBC to execute the function and complete execution.

2.14.7 Related Functions

```
SQLAllocHandle

SQLConnect

SQLDisconnect

SQLFreeHandle

SQLSetConnectAttr
```

2.14.8 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
sprintf(connStr,
 "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s",
/* ;PORT_NO=20300", */
                       USERNAME, PASSWD, 2, NLS);
/* Establish a connection. */
if (SQLDriverConnect( dbc, NULL, connStr, SQL_NTS,
                      NULL, 0, NULL,
                      SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
{
    execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
    return SQL_ERROR;
}
```

2.15 SQLEndTran

SQLEndTran requests a commit or rollback operation for all active operations on all statements associated with a connection. SQLEndTran can also request that a commit or rollback operation be performed for all connections associated with an environment.

2.15.1 Syntax

```
SQLRETURN  SQLEndTran (
    SQLSMALLINT    handleType,
    SQLHENV        handle,
    SQLSMALLINT    type );
```

2.15.2 Arguments

Data Type	Argument	In/Out	Description
SQLSMALLINT	handleType	Input	Handle type identifier. It should be either SQL_HANDLE_ENV or SQL_HANDLE_DBC.
SQLHENV	handle	Input	The handle.
SQLSMALLINT	type	Input	One of the following two values: SQL_COMMIT SQL_ROLLBACK

2.15.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.15.4 Description

If the *handleType* is SQL_HANDLE_ENV and the handle is effective environment handle, the ODBC will call SQLEndTran () for each connection handle related to the environment. The handle argument to call the ODBC must be the environment handle of the ODBC. In this case, the ODBC may commit the transaction or attempt to rollback depending on the type in the connected status.

If the type is SQL_COMMIT, SQLEndTran () will send commit command to the session related to the connection. If the type is SQL_ROLLBACK, SQLEndTran () will give rollback request to the connection-related session.

In case of the manual commit mode, by calling SQLSetConnectAttr () that can set SQL_ATTR_AutoCommit statement attribute as SQL_AutoCommit_OFF, the new transaction will internally start when an SQL statement to be included in the transaction is executed.

2.15.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.15.6 Related Functions

```
SQLFreeHandle
```

```
SQLFreeStmt
```

2.15.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_tran1.cpp

```
SQLEndTran(SQL_HANDLE_DBC, dbc, SQL_COMMIT);
```

2.16 SQLError

SQLError returns error or status information.

SQLErrorW() as a Unicode string supports same execution as SQLError().

2.16.1 Syntax

```
SQLRETURN  SQLError(
    SQLHENV        env,
    SQLHDBC        dbc,
    SQLHSTMT       stmt,
    SQLCHAR        *state,
    SQLINTEGER     *err,
    SQLCHAR        *msg,
    SQLSMALLINT    msgMax,
    SQLSMALLINT    *msgLength );
```

2.16.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV	env	Input	Environment Handle
SQLHDBC	dbc	Input	Connection Handle
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	state	Output	Pointer of SQLSTATE
SQLINTEGER *	err	Output	Pointer of unique ALTIBASE HDB error code
SQLCHAR *	msg	Output	Pointer of the diagnosis message
SQLSMALLINT	msgMax	Input	The character number of the * msg buffer
SQLSMALLINT *	msgLength	Output	Pointer of the total length of return buffer. Excludes NULL termination character.

2.16.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.16.4 Description

SQLSTATE is the same as defined by X/OPEN SQL CAE and X/OPEN SQLCLI snapshot.

SQLERROR() gets diagnosis information as followings:

To gain environment-related diagnosis information, send a valid environment handle. Set dbc and stmt as SQL_NULL_DBC and SQL_NULL_STMT.

To acquire connection-related diagnosis information, send the valid database connection handle and set stmt as SQL_NULL_STMT. Arguments env will be ignored.

To acquire diagnosis information related to a command, send the valid statement handle. env and dbc Arguments will be ignored.

If diagnosis information created by one ODBC function is not catched before functions other than SQLERROR () are called by the same handle, information related to calling of the previous functions will be lost. Always true regardless whether there is diagnosis information created by the second calling of the ODBC function.

To prevent the error message from being cut, the buffer length will be declared as SQL_MAX_MESSAGE_LENGTH + 1. The message text cannot be longer than this.

2.16.5 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
SQLINTEGER errNo;
SQLSMALLINT msgLength;
SQLCHAR errMsg[MSG_LEN];

if (SQLERROR ( SQL_NULL_HENV, aCon, aStmt,
               NULL, &errNo,
               errMsg, MSG_LEN, &msgLength ) == SQL_SUCCESS)
{
    printf(" Error : # %ld, %s\n", errNo, errMsg);
}
```

2.17 SQLExecDirect

If an SQL statement includes parameters, the given SQL statement will be directly executed using the current value of the parameter marker. When Using SQLExecDirect(), the SQL statement can be executed only once and is the fastest method to submit for an one-time execution.

SQLExecDirectW() as a Unicode string supports same execution as SQLExecDirect().

2.17.1 Syntax

```
SQLRETURN  SQLExecDirect (
    SQLHSTMT       stmt,
    SQLCHAR        *sql,
    SQLINTEGER     sqlLength );
```

2.17.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	sql	Input	SQL statement to be executed
SQLINTEGER	sqlLength	Input	The length, in bytes, of the contents of the *sql* argument

2.17.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_NO_DATA_FOUND

SQL_INVALID_HANDLE

SQL_ERROR
```

2.17.4 Description

The parameter marker can be included in an SQL statement string. Tthe parameter marker specified by "?" and designates the place of the parameters to be replaced as an application variable when SQLExecDirect() is called. SQLBindParameter () binds an application variable with the parameter marker, and displays whether data must be converted during data transmission. All parameters must be bound before SQLExecDirect () is called.

To select the row in the result set received from the server when an SQL statement is SELECT, the buffer must be bound by SQLBindCol () after SQLExecDirect () is successfully returned and SQLFetch () must be called to refer to the bound buffer.

If SQLExecDirect () executes an UPDATE or DELETE statement that does not affect any row, calling SQLExecDirect () will return SQL_NO_DATA. Use SQLRowCount() to check the number of affecting records.

2.17.5 Diagnosis

SQLSTATE	Description	Comments
08003	Invalid connection handle used	
08S01	Communication failure	Communication channel error
HY000	General error	
HY001	Memory allocation error	Failed to allocate the memory for the explicit handle.
HY009	Invalid Arguments used (null pointer).	*sql is a NULL pointer

2.17.6 Related Functions

```
SQLBindCol

SQLEndTran

SQLExecute

SQLFetch

SQLGetData

SQLPrepare

SQLStmtAttr
```

2.17.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
sprintf(query,"SELECT * FROM DEMO_EX1");
if (SQLExecDirect(stmt,query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
       SQLFreeStmt(stmt, SQL_DROP);
       return SQL_ERROR;
 }
```

2.18 SQLExecute

SQLExecute submits a prepared statement, using the current values of the parameter marker variables if any parameter markers exist in the statement.

2.18.1 Syntax

```
SQLRETURN  SQLExecute (
    SQLHSTMT    stmt );
```

2.18.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle

2.18.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_NO_DATA_FOUND

SQL_INVALID_HANDLE

SQL_ERROR
```

2.18.4 Description

The parameter marker can be included in a SQL statement string. The parameter marker specified by '?' and designates the place of the parameters to be replaced as an application variable when SQLExecute () is called. SQLBindParameter () must bind each parameter marker with a corresponding application variable, and displays whether the data must be converted for transmission. All parameters must be bound before SQLExecute () is called.

SQLExecute() executes a statement prepared by SQLPrepare(). After the application processes or discards the results from a call to SQLExecute, the application can call SQLExecute() again with new parameter values.

The command executed by SQLExecDirect () cannot be execute again by SQLExecute (). SQLPrepare () must be called first.

In case SQLExecute () executes an UPDATE and DELETE statement that does not affect any row in the database, calling SQLExecute () will return SQL_NO_DATA. Use SQLRowCount() to check the number of record.

If SQL_ATTR_PARAMSET_SIZE statement attribute is higher than 1 and the SQL statement has at least one parameter marker, SQLExecute () will execute an SQL statement once for a series of the parameters in the array indicated by the *Value argument upon calling of SQLBindParameter ().

2.18.5 Diagnosis

SQLSTATE	Description	Comments
07006	Restricted data type attribute violation	The column data within the result set must not be converted to the data type expressed in cType of SQLBindCol().
08003		When stmt is not connected or the connection is released
08S01	Communication channel error	The communication link between the driver and the database to which the driver was connected failed before the function completed processing.
HY000	General error	
HY090	Invalid string or buffer length	If SQLBindParameter () and the related parameters are NULL pointers, the parameter length is not 0 or SQL_NULL_DATA. If the parameter designated with SQLBindParameter () is not the NULL pointer, C data type will be SQL_C_BINARY or SQL_C_CHAR and the parameter length will be smaller than 0.

SQLExecute () can return all SQLSTATE data that are returned by SQLPrepare ().

2.18.6 Related Functions

`SQLBindCol`

`SQLEndTran`

`SQLExecDirect`

`SQLFetch`

`SQLFreeStmt`

`SQLGetData`

`SQLPrepare`

`SQLSetStmtAttr`

2.18.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex2.cpp

```
See the examples of SQLBindParameter ().
```

2.19 SQLFetch

SQLFetch() fetches the next rowset of data from the result set and returns data for all bound columns.

The user can separately get columns by directly receiving the data for the variables specified in SQLBindCol () using or by calling SQLGetData () to fetch. In case SQLFetch () is called while conversion is set upon binding of the column, the data will be converted.

2.19.1 Syntax

```
SQLRETURN  SQLFetch (
    SQLHSTMT    stmt);
```

2.19.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle

2.19.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR

SQL_NO_DATA_FOUND
```

2.19.4 Description

SQLFetch() can be called only when the recently executed command in stmt is a SELECT statement.

SQLBindCol () and the number of binding variables of the application must not exceed the number of columns in the result set. Otherwise, SQLFetch () will fail.

2.19.4.1 Positioning the Cursor

When the result set is created, the cursor is positioned before the start of the result set. SQLFetch () returns the next row set in the result set. The SQL_ATTR_ROW_ARRAY_SIZE statement attribute specifies the number of rows in the rowset. For example, if the number of rows is 5 when the result set has total 100 rows, the result row set of initial SQLFetch () is from 1 to 5. Also, if the result set is from the 52nd row to 56th row of the current row set, 57th rows to 61st rows will be returned by

SQLFetch (). SQL_SUCCESS will be returned, and the number of released rows will be 5. The following table shows the row set and returns the code that was returned by SQLFetch ().

Current row set	Returned code	New row set	Number of the fetched rows
Post-start	SQL_SUCCESS	1 to 5	5
1 to 5	SQL_SUCCESS	6 to 10	5
52 to 56	SQL_SUCCESS	57 to 61	5
91 to 95	SQL_SUCCESS	96 to 100	5
93 to 97	SQL_SUCCESS	98 to 100	3
96 to 100	SQL_NO_DATA	None	0
99 to 100	SQL_NO_DATA	None	0
After end	SQL_NO_DATA	None	0

After SQLFetch () is returned, the current row will become the first row of the row set.

2.19.4.2 Returning the Data in Bound Columns.

As SQLFetch() returns each row, it places the data for each bound column in the buffer bound to that column. If no columns are bound, SQLFetch does not return any data but does move the block cursor forward.

For each bound column, SQLFetch () does the following:

1. When the data is NULL, set SQL_NULL_DATA in the length/indicator buffer and go to the next column. If the data for the column is not NULL, SQLFetch proceeds to step 2

2. Converts the data of the type specified in the type argument of SQLBindCol ().

3. If the data is converted into the flexible length data type, SQLFetch () will inspect whether the data length (including NULL-terminatior when converted into SQL_C_CHAR) exceeds the data buffer length. If the character data length exceeds the data buffer length, SQLFetch () will cut the NULL-terminatior according to the data buffer length. In this way, finish the data composed of NULL characters. If the binary data length exceeds the data buffer length, SQLFetch () will cut the data according to the data buffer. The length of the data buffer is specified in SQLBindCol () length.

4. Positions the converted data in the data buffer.

5. Positions the data length in the length/indicator buffer. If the indicator pointer and the length pointer are set as the same buffer, the length will be recorded for the valid data and SQL_NULL_DATA will be recorded for the NULL data. If there is no bound length/indicator buffer, SQLFetch () will not return the length.

   For CLOB data, if the driver cannot determine the number of available bytes, as is sometimes the case with long data, it sets the length to SQL_NO_TOTAL and returns SQLSTATE

01004(Data truncated) and SQL_SUCCESS_WITH_INFO.

The contents of the bound data buffer and the length/indicator buffer are not defined unless SQLFetch () returns SQL_SUCCESS or SQL_SUCCESS_WITH_NOINFO. When the result of SQLFetch () is SQL_ERROR, it is invalid value.

2.19.4.3 Row Status Array

The row status array is used to return the status of each row in the rowset. The address of this array is specified with the SQL_ATTR_ROW_STATUS_PTR statement attribute. The array is allocated by the application and must have as many elements as are specified by the SQL_ATTR_ROW_ARRAY_SIZE statement attribute. If the value of SQL_ATTR_ROW_STATUS_PTR is a NULL pointer, SQLFetch () will not return the row status.

2.19.4.4 Rows Fetched Buffer

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

If SQLFetch () does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO except SQL_NO_DATA, the contents of the row fetched buffer will not be determined. In this case, the row fetched buffer will be set to 0.

2.19.4.5 Error Handling

Errors and warnings can apply to individual rows or to the entire function.

Errors and warnings for the entire function.

If a random warning is applied to the entire function, SQLFetch () will return SQL_SUCCESS_WITH_INFO and proper SQLSTATE. The warning status records applied to the function must be returned before the status records are applied to each row.

2.19.5 Diagnosis

SQLSTATE	Description	Comments
01004	String data, right truncated	String or binary data returned for a column resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated.

SQLSTATE	Description	Comments
07006	Restricted data type attribute violation	The column data within the result set must not be converted to the data type expressed in cType of SQLBindCol().
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	

2.19.6 Related Functions

```
SQLBindCol

SQLDescribeCol

SQLExecDirect

SQLExecute

SQLFreeStmt

SQLGetData

SQLNumResultCols

SQLPrepare
```

2.19.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex2.cpp

```
See example of SQLBindCol()
```

2.20 SQLFetchScroll

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

2.20.1 Syntax

```
SQLRETURN SQLFetchScroll(SQLHSTMT       stmt,
                         SQLSMALLINT    fOrient,
                         SQLINTEGER     fOffset)
```

2.20.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	fOrient	Input	Type of fetch, This argument determines scroll direction. SQL_FETCH_NEXT SQL_FETCH_PRIOR SQL_FETCH_FIRST SQL_FETCH_LAST SQL_FETCH_ABSOLUTE SQL_FETCH_RELATIVE
SQLINTEGER	fOffset	Input	Number rows to fetch

2.20.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR

SQL_NO_DATA_FOUND
```

2.20.4 Description

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

You can set the cursor direction like followings;

SQL_FETCH_NEXT

Return the next rowset. This is equivalent to calling SQLFetch(). SQLFetchScroll() ignores the value of FetchOffset.

SQL_FETCH_PRIOR

Return the prior rowset. SQLFetchScroll() ignores the value of FetchOffset.

SQL_FETCH_RELATIVE

Return the rowset FetchOffset from the start of the current rowset.

SQL_FETCH_ABSOLUTE

Return the rowset starting at row FetchOffset.

SQL_FETCH_FIRST

Return the first rowset in the result set. SQLFetchScroll() ignores the value of FetchOffset.

SQL_FETCH_LAST

Return the last complete rowset in the result set. SQLFetchScroll() ignores the value of FetchOffset.

2.20.5 Diagnosis

SQLSTATE	Description	Comments
01004	String data, right truncated	String or binary data returned for a column resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated.
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	

2.20.6 Related Functions

```
SQLFetch
```

2.20.7 Example

```
SQLFetchScroll(stmt , SQL_FETCH_RELATIVE, 10);
```

2.21 SQLForeignKeys

SQLForeignkeys () can return the following:

- A list of foreign keys of a specified table (columns of a specified table referring to the primary keys of other tables)
- A list of foreign keys of other tables referring to the primary keys of a specified table

SQLForeignKeysW() as a Unicode string supports same execution as SQLForeignKeys().

2.21.1 Syntax

```
SQLRETURN  SQLForeignKeys (
    SQLHSTMTstmt,
    SQLCHAR        *pkcName,
    SQLSMALLINT    pkcNaneLength,
    SQLCHAR        *pksName,
    SQLSMALLINT    pksNameLength,
    SQLCHAR        *pktName,
    SQLSMALLINT    pktNameLength,
    SQLCHAR        *fkcName,
    SQLINTEGER     fkcNaneLength,
    SQLCHAR        *fksName,
    SQLSMALLINT    fksNameLength,
    SQLCHAR        *fktName,
    SQLSMALLINT    fktNameLength);
```

2.21.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR*	pkcName	Input	Primary key table catalog name
SQLSMALLINT	pkcName-Length	Input	The length, in bytes, of **pkcName*
SQLCHAR *	pksName	Input	Primary key table schema name
SQLSMALLINT	pksName-Length	Input	The length, in bytes, of **pksName*
SQLCHAR *	pktName	Input	Primary key table
SQLSMALLINT	pktName-Length	Input	The length, in bytes, of **pktName*
SQLCHAR *	fkcName	Input	Foreign key table catalog name

Data Type	Argument	In/Out	Description
SQLSMALLINT	fkcName-Length	Input	The length, in bytes, of **fkcName*
SQLCHAR *	fksName	Input	Foreign key table schema name
SQLSMALLINT	fksName-Length	Input	The length, in bytes, of **ksName*
SQLCHAR *	fktName	Input	Foreign key table name
SQLSMALLINT	fktName-Length	Input	The length, in bytes, of **fktName*

2.21.3 Return Values

SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR

2.21.4 Description

If **pktName* contains a table name, SQLForeignKeys ()returns a result set containing the primary key of the specified table and all of the foreign keys that refer to it. The list of foreign keys in other tables does not include foreign keys that point to unique constraints in the specified table.

If **fktName* has a table name, SQLForeignKeys () will returns a result set containing all of the foreign keys in the specified table that point to primary keys in others tables, and the primary keys in the other tables to which they refer. The list of foreign keys in the specified table does not contain foreign keys that refer to unique constraints in other tables.

If both **pktName* and **fktName* have table names, SQLForeignKeys will return the foreign keys of the table specified by **fktName*. For **fktName*, refer to the primary keys of the table specified in **pktName*.

SQLForeignKeys () returns the result in the standard result set form sorted by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_Name, and KEY_SEQ in case the foreign keys related to the primary keys are requested. If the primary keys related to the foreign keys are requested, the result in the standard result set sorted by PKTABLE_CAT, PKTABLE_SCHEM, PKTABLE_Name, and KEY_SEQ will be returned. The following table lists the strings of the result sets.

2.21.4.1 Columns Returned by SQLForeignKeys ()

String Name	String No.	Data Type	Description
PKTABLE_CAT	1	VARCHAR	Always NULL Return
PKTABLE_SCHEM	2	VARCHAR	Foreign key table schema name; NULL if not applicable to the database.
PKTABLE_NAME	3	VARCHAR (NOT NULL)	Primary key table name
PKCOLUMN_NAME	4	VARCHAR (NOT NULL)	Primary key column name.As for the unnamed string, ODBC makes the empty character string return.
FKTABLE_CAT	5	VARCHAR	Always NULL Return
FKTABLE_SCHEM	6	VARCHAR	Primary key table schema name; NULL if not applicable to the database
FKTABLE_NAME	7	VARCHAR (NOT NULL)	Foreign key table name
FKCOLUMN_NAME	8	VARCHAR (NOT NULL)	Foreign key column name.As for the unnamed string, ODBC makes the empty character string return.
KEY_SEQ	9	SMALL-INT(NOT NULL)	Column number sequence (starting with 1)
UPDATE_RULE	10	SMALLINT	Application of SQL_NO_ACTION to the foreign key upon UPDATE operation
DELETE_RULE	11	SMALLINT	Application of SQL_NO_ACTION to the foreign key upon DELETE operation
FK_NAME	12	VARCHAR	Foreign key name. NULL not proper for the database
PK_NAME	13	VARCHAR	Primary key name. NULL not proper for the database
DEFERRABILITY	14	SMALLINT	SQL_INITIALLY_IMMEDIATE

2.21.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing between the ODBC and the database is completed
HY009	Invalid Arguments used (null pointer).	Argument pktName and fktName are NULL pointer.
HY090	Invalid string or buffer length	The value of one of the name length arguments was less than 0 but not equal to SQL_NTS.

2.21.6 Related Functions

SQLBindCol

SQLFetch

SQLPrimaryKeys

SQLStatistics

2.21.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta9.cpp

```
if (SQLForeignKeys(stmt,
                        NULL, 0,
                        "SYS", SQL_NTS,
                        "ORDERS", SQL_NTS,
                        NULL, 0,
                        NULL, 0,
                        NULL, 0) != SQL_SUCCESS)
   {
      execute_err(dbc, stmt, "SQLForeignKeys");
      SQLFreeStmt(stmt, SQL_DROP);
      return SQL_ERROR;
   }
  SQLBindCol(stmt, 2, SQL_C_CHAR, szPKSchema, NAME_LEN, &cbPKSchema);
  SQLBindCol(stmt, 3, SQL_C_CHAR, szPKTableName, NAME_LEN,&cbPKTableName);
  SQLBindCol(stmt, 4, SQL_C_CHAR, szPKColumnName, NAME_LEN, &cbPKColumnName);
  SQLBindCol(stmt, 6, SQL_C_CHAR, szFKSchema, NAME_LEN, &cbFKSchema);
  SQLBindCol(stmt, 7, SQL_C_CHAR, szFKTableName, NAME_LEN,&cbFKTableName);
  SQLBindCol(stmt, 8, SQL_C_CHAR, szFKColumnName, NAME_LEN, &cbFKColumnName);
  SQLBindCol(stmt, 9, SQL_C_SSHORT, &KeySeq, 0, &cbKeySeq);
```

2.22 SQLFreeConnect

SQLFreeConnect() frees resources associated with a specific connection.

SQLFreeConnect () has been replaced by SQLFreeHandle ().

2.22.1 Syntax

```
SQLRETURN  SQLFreeConnect (
    SQLHDBC    dbc );
```

2.22.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	Input	Connection Handle Pointer

2.22.3 Return Values

```
SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_ERROR
```

2.22.4 Description

When this function is called in the connected status, SQL_ERROR will be returned but the connection handle is still valid.

2.22.5 Related Functions

```
SQLDisconnect

SQLFreeEnv
```

2.22.6 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp .>

```
if ( conn_flag == 1 )
{
    SQLDisconnect( dbc );
}
if ( dbc != NULL )
```

```
{
    SQLFreeConnect( dbc );
}
if ( env != NULL )
{
    SQLFreeEnv( env );
}
```

2.23 SQLFreeEnv

SQLFreeEnv frees resources associated with a specific environment. SQLFreeEnv () has been replaced by SQLFreeHandle ().

2.23.1 Syntax

```
SQLRETURN  SQLFreeEnv (
    SQLHENV    env );
```

2.23.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV	env	Input	Environment Handle

2.23.3 Return Values

```
SQL_SUCCESS
```

```
SQL_INVALID_HANDLE
```

```
SQL_ERROR
```

2.23.4 Description

In case this function is called in a valid connection handle, SQL_ERROR will be returned but the environment handle is still valid.

* SQLFreeConnect () must be called before this function.

2.23.5 Related Functions

```
SQLFreeConnect
```

2.23.6 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp >

SQLFreeConnect()>

```
See example of SQLFreeConnect()
```

2.24 SQLFreeHandle

SQLFreeHandle frees resources associated with a specific environment, connection, statement, or descriptor handle

2.24.1 Syntax

```
SQLRETURN  SQLFreeHandle (
    SQLSMALLINT    handleType,
    SQLHANDLE      handle );
```

2.24.2 Arguments

Data Type	Argument	In/Out	Description
SQLSMALLINT	handleType	Input	Handle type to be freed: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT If the *handleType* is not the one of these values, SQLFreeHandle() free SQL_INVALID_HANDLE.
SQLHANDLE	handle	Input	Handle to be freed

2.24.3 Return Values

```
SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_ERROR
```

If SQLFreeHandle () returns SQL_ERROR, the handle is valid.

2.24.4 Description

SQLFreeHandle () can replace with SQLFreeEnv (), SQLFreeConnect (), and SQLFreeStmt () function.

Once the handle is released, an application cannot use the released handle.

2.24.4.1 Freeing an Environment Handle

Before handleType calls SQLFreeHandle (), SQL_HANDLE_ENV, an application must call SQLFreeHandle () of which *handleType* is SQL_HANDLE_DBC for all connections allocated in the corresponding environment. Otherwise, calling SQLFreeHandle () will return SQL_ERROR and the corresponding environment and the random activated connection remains still vaild.

2.24.4.2 Freeing a Connection Handle

If an error is detected on this handle before *handleType* calls SQLFreeHandle (), SQL_HANDLE_DBC, an application must call SQLDisconnect () for the connection. Otherwise, calling of SQLFreeHandle () will return SQL_ERROR and connection will be still vaild.

2.24.4.3 Freeing a Statement Hanle

Calling SQLFreeHandle () of which *handleType* is SQL_HANDLE_STMT will release all resources allocated by calling SQLAllocHandle of which handleType is SQL_HANDLE_STMT. Calling SQLFreeHandle () to release the command with the result suspended by an application will delete the suspended results.

2.24.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	
HY001	Memory allocation error	Failed to allocate the memory for the explicit handle.

2.24.6 Related Functions

```
SQLAllocHandle
```

2.24.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta1.cpp .>

```
if ( dbc != NULL )
{
    SQLFreeHandle( SQL_HANDLE_DBC, dbc );
}
if ( env != NULL )
{
    SQLFreeHandle( SQL_HANDLE_ENV, env );
}
```

2.25 SQLFreeStmt

SQLFreeStmt stops processing associated with a specific statement, closes any open cursors associated with the statement, discards pending results, or, optionally, frees all resources associated with the statement handle.

2.25.1 Syntax

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

2.25.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	fOption	Input	Handle Control Method SQL_CLOSE, SQL_DROP, SQL_UNBIND, SQL_RESET_PARAMS

2.25.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.25.4 Description

Calls SQLFreeStmt () with the one of following options:

SQL_CLOSE: Closes the cursor related to stmt, and discards all pending results. An application can open this cursor again by executing SELECT statement again using the same or different variables. However, if no cursor is open, this option will not effect for the application.

SQL_DROP: The resources related to the input statement handle will be released, and the handle will be freed. In case there is an open cursor, the cursor will be closed and all pending results will be deleted.

SQL_UNBIND: Releases all column buffers bound by SQLBindCol for the given StatementHandle.

SQL_RESET_PARAMS: Releases all parameter buffers set by SQLBindParameter (). The relation between an application variable or file reference and an SQL statement parameter marker of the statement handle will be released.

2.25.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.25.6 Related Functions

```
SQLAllocHandle
```

```
SQLFreeHandle
```

2.25.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
SQLFreeStmt(stmt, SQL_DROP);
```

2.26 SQLGetConnectAttr

SQLGetConnectAttr gets the current setting of connection.

SQLGetConnectAttrW() as a Unicode string supports same execution as SQLGetConnectAttr().

2.26.1 Syntax

```
SQLRETURN SQLGetConnectAttr (
       SQLHDBC        dbc,
       SQLINTEGER     Attribute,
       SQLPOINTER     ValuePtr,
       SQLINTEGER     BufferLength,
       SQLINTEGER     *StringLengthPtr );
```

2.26.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	Input	Connection Handle
SQLINTEGER	Attribute	Input	Attribute to retrieve
SQLPOINTER	ValuePtr	Output	Memory pointer to bring the value corresponding to the attribute
SQLINTEGER	BufferLength	Input	If *ValuePtr is the pointer of character string, this has the byte length of string or the value of SQL_NTS. If *ValuePtr is the pointer of binary buffer, its value is the binary length of data. If *ValuePtr is the pointer with fixed-length data type like integer, its value is ignored.
SQLINTEGER	StringLength-Ptr	Output	This returns bytes (excluding the null-termination characater) available to return in *ValuePtr.

2.26.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.26.4 Description

In case the attribute returns the string, ValuePtr will be a pointer indicating the string buffer.

The maximum length of the returned string including the NULL-terminatior will be the BufferLength bytes.

Depending on the attribute, an application must establish connections before calling SQLGetConnectAttr ().

2.26.4.1 List of Connection Attributes

Attribute	Contents
SQL_ATTR_AUTOCOMMIT	32-bit value to set reflection for the connection. SQL_AUTOCOMMIT_ON: Each SQL statement is automatically committed. SQL_AUTOCOMMIT_OFF: Not automatically committed.
SQL_ATTR_CONNECTION_TIMEOUT	Set timeout value to prevent blocking that may occur in select() or poll() in an unstable network
SQL_ATTR_PORT	Server port number (32-bit Integer)
SQL_ATTR_TXN_ISOLATION	32-bit value that sets the transaction isolation level for the current connection referred to by dbc.
SQL_ATTR_LOGIN_TIMEOUT	SQLINTEGER value corresponding to the waiting time (in seconds) for logging in before the data is returned to an application. The default depends on the driver. If ValuePtr is 0, timeout will be disabled and connection attempt will be permanently awaited.
SQL_ATTR_CONNECTION_DEAD	SQL_CD_TRUE Disconnected status SQL_CD_FALSE: Connected status

2.26.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC driver and the database
HYC00	Optional feature not implemented.	Not supported by the driver specified in the argument Attribute.

2.26.6 Related Functions

```
SQLSetConnectAttr
```

2.26.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_dead.cpp

```
rc = SQLGetConnectAttr( dbc, SQL_ATTR_CONNECTION_DEAD, &isDead, 0, NULL );
…
if ( rc == SQL_SUCCESS )
 {
 if ( isDead == SQL_CD_TRUE )
 {
 printf("The Connection has been lost.\n");
 }
 else
 {
 printf("The Connection is active.\n");
 }
 }
```

2.27 SQLGetData

SQLGetData retrieves data for a specified column in the result set. It can be called multiple times to retrieve variable-length data in parts.

SQLGetData () will be called for each column, and SQLFetch () will be called to retrieve one or more rows:

2.27.1 Syntax

```
SQLRETURN  SQLGetData (
    SQLHSTMT        stmt,
    SQLSMALLINT     col,
    SQLSMALLINT     cType,
    SQLPOINTERV     alue,
    SQLINTEGERV     alueMax,
    SQLINTEGER      *pcbValue );
```

2.27.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	col	Input	Column sequence. Starting with 1
SQLSMALLINT	cType	Input	The type identifier of C data type in the * Value buffer
SQLPOINTER	Value	Output	Pointer of the buffer to return the data
SQLINTEGER	ValueMax	Input	Length of *Value buffer, in bytesWhen returning the character data to the *Value, the *Value argument must include a NULL-terminatior. Otherwise, the ODBC cuts out the data. In case of a fixed length data such as integer, date structure, etc, are returned, the ODBC will ignore ValueMax. Therefore, a sufficient buffer size must be allocated. Otherwise, the ODBC passes through the end of the buffer and saves the data. If ValueMaxis smaller than 0, SQLGetData () will return SQLSTATE HY090. If the Value is not set as a NULL pointer, ValueMax will be ignored by the ODBC.

Data Type	Argument	In/Out	Description
SQLINTEGER *	pcbValue	Output	Pointer of buffer that will return the length or indicator variable. If this variable is a NULL pointer, no length or indicator will be returned. When the fetched data is NULL, this variable will return the error (SQL_SUCCESS_WITH_INFO). SQLGetData () will return the following to the length/indicator buffer: The length of the data available to return SQL_NO_TOTAL SQL_NULL_DATA

2.27.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_NO_DATA_FOUND

SQL_ERROR

SQL_INVALID_HANDLE
```

2.27.4 Description

SQLGetData () returns the data of a specified column. SQLGetData () can be called only after one or more rows are fetched by SQLFetch (). Although there are some exceptions, the user can bind a several columns in the row and call SQLGetData () for the other columns.

2.27.4.1 Using SQLGetData ()

This function returns only the data of unbound columnsWhen SQLGetData () is called, the col value must be higher than or the equal to previously called column. In other words, the data must be searched in ascending order.

2.27.4.2 Retrieving Data with SQLGetData ()

To return the data to a specified column, SQLGetData () must execute the following series of procedures.

1. Returns SQL_NO_DATA if it has already returned all of the data for the column.
2. If the data is NULL, SQL_NULL_DATA will be set in *pcbValue.

   If the data for the corresponding column is not NULL, SQLGetData () will proceed to the next phase.

3. If the data is converted into flexible data types such as character type or binary type, SQLGetData () will check whether the data length exceeds ValueMax. If the length of the data including NULL-terminatior exceeds ValueMax, SQLGetData () will cut the data to ValueMax length with the NULL-terminatior length deducted. In this way, finish the data composed of NULL characters. If the binary data length exceeds the data buffer length, SQLGetData () will cut the data according to the ValueMax.

   If the data buffer is small and does not include the NULL-terminatior, SQLGetData () will return SQL_SUCCESS_WITH_INFO.

   SQLGetData () does not cut the data converted into the fixed-length data type. SQLGetData () always assumes that the *Value length is the size of the data type.

4. Places the converted (or cut) data in **Value*.

5. Places the data length in **pcbValue*. If **pcbValue* is a NULL pointer, SQLGetData () will not return the length.

   For CLOB data, if the driver cannot determine the number of available bytes, as is sometimes the case with long data, it sets the length to SQL_NO_TOTAL and returns SQLSTATE 01004(Data truncated) and SQL_SUCCESS_WITH_INFO. The last call to SQLGetData() returns the length of the data available(including the null-termination character) at the current call; that is , the length decreases with each subsequent call.

6. If the data is cut during conversion although there is no loss of significant information (for example, 1.234 is converted into 1) or if *valueMax* is too small (for example, character string "abcde" i placed in the 4-byte buffer), SQLGetData () will return SQLSTATE 01004 (Data truncated) and SQL_SUCCESS_WITH_INFO.

If SQLGetData () does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, contents of the bound data buffer and the length/indicator buffer will not be defined.

2.27.5 Diagnosis

SQLSTATE	Description	Comments
01004	String data, right truncated	When the size of the value to be returned is higher than the size of the given buffer. For CLOB data, when the length of the data remaining in the specified column is greater than the size of the given buffer, SQL_NO_TOTAL is returned in **pcbValue*, and the function returns SQL_SUCCESS_WITH_INFO.

SQLSTATE	Description	Comments
07009	Invalid descriptor index	The col value is 0. The col value is higher than the column value in the result set. An application has already called SLQGetData () for the current row. The number of columns in the current calling is smaller than the number of columns in the previous calling.
HY000	General error	
HY010	Continuous function error	The given stmt cannot execute this function. This function can be called after the result set creation phase.
HY090	Invalid string or buffer length	valueMax is smaller than 0.

2.27.6 Related Functions

```
SQLBindCol

SQLExecDirect

SQLExecute

SQLFetch
```

2.27.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_info1.cpp

```
See example of SQLGetTypeInfo()
```

2.28 SQLGetDescField

This retrieves an attribute of descriptor. Unicode SQLGetDescFieldW() supports same execution as SQLGetDescField().

2.28.1 Syntax

```
SQLRETURN SQLGetDescField (
    SQLHDESC       desc,
    SQLSMALLINT    recNumber,
    SQLSMALLINT    fieldIdentifier,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     *bufferLength,
    SQLINTEGER     *stringLengthPtr);
```

2.28.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDESC	desc	In	Descriptor Handle
SQLSMALLINT	recNumber	In	This starts from 1 of column number.
SQLSMALLINT	fieldIdentifier	In	This specifies the attribute of column to retrieve.
SQLPOINTER	ValuePtr	Out	Buffer pointer where attributes are saved
SQLINTEGER *	bufferLength	In	ValuePtr Size
SQLINTEGER *	stringLength-Ptr	Out	Size speciified in ValuePtr

2.28.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_NO_DATA

SQL_INVALID_HANDLE

SQL_ERROR
```

2.28.4 Description

This retrieves one column information with descriptor handle.

2.28.5 Diagnostics

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY010	Function Sequence Error	
01004	Cutted Resource	The size of ValuePtr buffer is lesser than the size of returned data
07009	Invalid Descriptor Index	The value of recNumber is incorrect.

2.28.6 Related Function

`SQLGetDescRec`

`SQLSetDescField`

`SQLSetDescRec`

2.29 SQLGetDescRec

This retrieves multiple number of descriptor attributes. Unicode SQLGetDescRecW() supports same execution as SQLGetDescRec().

2.29.1 Syntax

```
SQLRETURN SQLGetDescRec (
    SQLHDESC       desc,
    SQLSMALLINT    recNumber,
    SQLCHAR        *name,
    SQLSMALLINT    bufferLength,
    SQLSMALLINT    *stringLength,
    SQLSMALLINT    *type,
    SQLSMALLINT    *subType,
    SQLLEN         *lengthPtr,
    SQLSMALLINT    *precision,
    SQLSMALLINT    *scale,
    SQLSMALLINT    *nullable);
```

2.29.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDESC	desc	In	Descriptor Handle
SQLSMALLINT	recNumber	In	This starts from 1 of column number.
SQLCHAR *	name	Out	Pointer recieving column name
SQLSMALLINT	bufferLength	In	Size of name buffer
SQLSMALLINT *	stringLength	Out	Size specified in name
SQLSMALLINT *	type	Out	Pointer recieving column type
SQLSMALLINT *	subType	Out	Pointer recieving subtype of column
SQLLEN *	lengthPtr	Out	Pointer recieving column length
SQLSMALLINT *	precision	Out	Pointer recieving column precision
SQLSMALLINT *	scale	Out	Pointer recieving column scale
SQLSMALLINT *	nullable	Out	Pointer recieving whether to specify null in column

2.29.3 Return Value

```
SQL_SUCCESS
```

```
SQL_SUCCESS_WITH_INFO

SQL_NO_DATA

SQL_INVALID_HANDLE

SQL_ERROR
```

2.29.4 Description

You can retrieve several information of column with descriptor handle.

2.29.5 Diagnostics

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY010	Function Sequence Error	
01004	Cutted Resource	The size of name buffer is lesser than that of retunred data.
07009	Invalid Descriptor Index	The value of recNumber is incorrect.

2.29.6 Related Function

```
SQLBindCol

SQLBindParameter

SQLGetDescFiled
```

2.30 SQLGetDiagField

SQLGetDiagField is Used to diagnose the result after an ODBC function is used.

SQLGetDiagFieldW() as a Unicode string supports same execution as SQLGetDiagField().

2.30.1 Syntax

```
SQLRETURN SQLGetDiagField(
    SQLSMALLINT    HandleType,
    SQLHANDLE      Handle,
    SQLSMALLINT    RecNumber,
    SQLSMALLINT    DiagIdentifier,
    SQLPOINTER     DiagInfoPtr,
    SQLSMALLINT    BufferLength,
    SQLSMALLINT    *StringLengthPtr)
```

2.30.2 Arguments

Data Type	Argument	In/Out	Description
SQLSMALLINT	HandleType	Input	A handle type to be assigned. It can be any of the followings: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT
SQLHANDLE	Handle	Input	If input HandleType is SQL_HANDLE_ENV, InputHandle should be SQL_NULL_HANDLE. If SQL_HANDLE_DBC, it should be an environment handle. If SQL_HANDLE_STMT, it should be a connection handle.
SQLSMALLINT	DiagIdentifier	Input	A type to be diagnosed. For now, SQL_DIAG_NUMBER is only supported.
SQLPOINTER	DiagInfoPtr	Output	A pointer to the buffer to which diagnostic information is returned.
SQLINTEGER	BufferLength	Input	If *DiagInfoPtr is the pointer of character string, this has the byte length of string or the value of SQL_NTS. If *DiagInfoPtr is the pointer of binary buffer, this has the binary length of data. If *DiagInfoPtr is the pointer with fixed-length data type, its value is ignored.
SQLINTEGER*	StringLength-Ptr	Output	This returns bytes (excluding the null-termination character) available to return in *ValuePtr.

2.30.3 Result Value

An error message for each handle type.

2.30.4 Description

This function is used to diagnose the execution result for an error. It is used in the following case:

In ODBC functions, when SQL_ERROR or SQL_SUCCESS_WITH_INFO is returned, error and warning information is gathered. This function is used to determine the gathered information.

Any ODBC function can call this function for diagnosis after its execution. This function shows the most recently stored diagnosis information.

Currently, it works for the following handle types:

`SQL_HANDLE_ENV`

`SQL_HANDLE_DBC`

`SQL_HANDLE_STMT`

For an input Argument, if HandleType is SQL_HANDLE_ENV, InputHandle should be SQL_NULL_HANDLE, or if SQL_HANDLE_DBC, it should be an environment handle, or if SQL_HANDLE_STMT, it should be a connection handle. That is, a value should be set properly for each input handle.

2.30.5 Related Function

`SQLError`

2.31 SQLGetDiagRec

You can use this when retrieving serveral attributes of diagnostic message after using ODBC function. Unicode SQLGetDiagRecW() supports same execution as SQLGetDiagRec().

2.31.1 Syntax

```
SQLRETURN SQLGetDiagRec (
    SQLSMALLINT    handleType,
    SQLHANDLE      handle,
    SQLSMALLINT    recNumber,
    SQLCHAR        *sqlstatus,
    SQLINTEGER     *nativeError,
    SQLCHAR        *messageText,
    SQLSMALLINT    bufferLength,
    SQLSMALLINT    *stringLength);
```

2.31.2 Argument

Data Type	Argument	In/Out	Description
SQLSMALLINT	handleType	In	Handle Type
SQLHANDLE	handle	In	Handle
SQLSMALLINT	recNumber	In	This starts from 1 of diagnostic message.
SQLCHAR *	sqlstatus	Out	SQLSTATE
SQLINTEGER *	nativeError	Out	Unique Error Number
SQLCHAR *	messageText	Out	Buffer pointer recieving message
SQLSMALLINT	bufferLength	In	Size of message text
SQLSMALLINT	stringLength	Out	Size specified in message text

2.31.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.31.4 Description

This retrieves several attributes of diagnostic message.

2.31.5 Related Function

`SQLGetDiagField`

2.32 SQLGetEnvAttr

This retrieves attribute value of environment handle.

2.32.1 Syntax

```
SQLRETURN SQLGetEnvAttr (
    SQLHENV        env,
    SQLINTEGER     attribute,
    SQLPOINTER     value,
    SQLINTEGER     bufferLength,
    SQLINTEGER     *stringLength);
```

2.32.2 Argument

Data Type	Argument	In/Out	Description
SQLHENV	env	In	Environment Handle
SQLINTEGER	attribute	In	This inserts attribute of envionment handle.
SQLPOINTER	value	Out	Buffer pointer where attributes are saved
SQLINTEGER	bufferLength	In	Size of Attribute Value
SQLINTEGER *	stringLength	Out	Size specified in the value of attribute

2.32.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_NO_DATA

SQL_INVALID_HANDLE

SQL_ERROR
```

2.32.4 Description

This retrieves attribute value of environment handle.

2.32.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY092	Invalid Attribute or Option	The value specified in attribute is not valid one supported by this driver.
01004	Cutted Resource	The size of value buffer is lesser than the size of returned data.
HYC00	Unsupported Attribute Use	The value specified in attribute is unsupported in driver.

2.32.6 Related Function

`SQLSetEnvAttr`

2.33 SQLGetFunctions

This retrieves function list supported by ODBC driver.

2.33.1 Syntax

```
SQLRETURN SQLGetFunctions (
    SQLHDBC          dbc,
    SQLUSMALLINT     functionId,
    SQLUSMALLINT     *supported);
```

2.33.2 Argument

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	In	Connection Handle
SQLUSMALLINT	functionId	In	Function ID
SQLUSMALLINT *	supported	Out	Array pointer recieving the supported function list

2.33.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.33.4 Description

This retrieves function list supported by ODBC driver. One list can be retrieved at a time or all lists can be retrieved with SQL_API_ALL_FUNCTIONS and SQL_API_ODBC3_ALL_FUNCTIONS.You should pinpoint the location suited for ID value of function in argumnet Supported. If this function is supported, SQL_TRUE is returned. Otherwise, SQL_FALSE is returned.If using SQL_API_ALL_FUNCTIONS, you shoould apply pointer of array whose size is 100 to Supported pointer and pinpoint the location suited for the value of function ID. If using SQL_API_ODBC3_ALL_FUNCTIONS, you should apply pointer of array whose size is the value of SQL_API_ODBC3_ALL_FUNCTIONS_SIZE to Supported pointer and you can check supported functions by using SQL_FUNC_EXISTS.

2.33.5 Diagnostics

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY010	Function Sequence Error	

2.33.6 Related Function

`SQLGetInfo`

2.34 SQLGetInfo

This indicates to return general information of DBMS accessed to application.

SQLGetInfoW() as a Unicode string supports same execution as SQLGetInfo().

2.34.1 Syntax

```
SQLRETURN SQLGetInfo(
    SQLHANDLE       ConnectionHandle,
    SQLUSMALLINT    InfoType,
    SQLPOINTER      InfoValuePtr,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT     *StringLengthPtr );
```

2.34.2 Arguments

Data Type	Arguments	In/Out	Description
SQLHANDLE	Connection-Handle	In	Database Connection Handle
SQLUSMALLINT	InfoType	In	Type of information which you want to search. The following values are available in addition to the values provided by the ODBC standard. ALTIBASE_PROTO_VER: A character string that indicates the protocol version of the ALTIBASE HDB connected by the driver.
SQLPOINTER	InfoValuePtr	Out	5 kinds of data are outputted according to data types of buffer pointer which makes data returned16 bit integer value32 bit integer value32 bit binary value32 bit maskNull Determination Character String
SQLSMALLINT	BufferLength	In	The byte size of buffer which InfoValuePtr indicates
SQLSMALLINT *	StringLength-Ptr	Out	The byte length of result values which InfoValuePtr indicates

2.34.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO
```

```
SQL_ERROR

SQL_INVALID_HANDLE
```

2.34.4 Description

SQLGetInfo() is used to get general information of DBMS. You can get relevant information of each type according to InfoType with this function, and ALTIBASE HDB follows the standard of ODBC.

2.34.5 Diagnosis

SQLSTATE	Description	Comments
01004	String data, right truncated	The size of returned values is bigger than that of the given buffer.
08003	Disconnection	Disconnection Status
08S01	Communication channel error(Data Sending/Recieving Failure)	Communication channel error before the function processing is completed between the ODBC and the database.
HY000	General Error	
HY090	Invalid Arguments used	One value among name length arguments must be under 0 or not be equal to SQL_NTS.
HY096	Out of InfoType Range	The values specified in InfoType are invalid in the version which ODBC provides.

2.35 SQLGetPlan

This is nonstandard function returning execution information.

2.35.1 Syntax

```
SQLRETURN SQLGetPlan (
    SQLHSTMT    stmt,
    SQLCHAR     **aPlan);
```

2.35.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	In	Input Statement Handle
SQLCHAR**	aPlan	Out	Pointer to store output execution information

2.35.3 Returned Values

```
SQL_SUCCESS

SQL_ERROR
```

2.35.4 Description

SQLGetPlan is nonstandard function, but you can use it when retrieving information of execution plan. At this time, you shouldn't modify information returned by aPlan.

2.35.5 Related Function

```
SQLSetConnectAttr
```

2.35.6 Example

< See : $ALTIBASE_HOME/sample/SQLCLI/demo_plan.cpp >

```
if( SQLSetConnectAttr( dbc, ALTIBASE_EXPLAIN_PLAN,
 (SQLPOINTER) 1, 0) != SQL_SUCCESS)
.
.
.
 if ( SQLGetPlan(stmt, (SQLCHAR**)&plan) != SQL_SUCCESS )
```

2.36 SQLGetStmtAttr

SQLGetStmtAttr retrieves the attributes related to the current statement handle .

SQLGetStmtAttrW() as a Unicode string supports same execution as SQLGetStmtAttr().

2.36.1 Syntax

```
SQLRETURN  SQLGetStmtAttr (
    SQLHSTMT       stmt,
    SQLINTEGER     Attribute,
    SQLPOINTER     param,
    SQLINTEGER     StringLength
    SQLINTEGER     *StringLengthPtr );
```

2.36.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLINTEGER	Attribute	Input	Attribute to set, Supported attributes SQL_ATTR_MAX_ROWS, SQL_ATTR_ROW_ARRAY_SIZE, SQL_ATTR_ROW_BIND_TYPE, SQL_ATTR_ROW_STATUS_PTR ALTIBASE_STMT_ATTR_ATOMIC_ARRAY
SQLPOINTER	param	Output	Pointer of the value related to the attrib-uteDepending on the Attribute, the param will be a 32-bit integer, the pointer of the Null-terminator, the binary pointer, or the value defined in the ODBC. If Attribute is the unique value of the ODBC, param is the integral number with a sign.
SQLINTEGER	StringLength	Input	If the Attribute has been defined in the ODBC and if the param indicates the character string or binary buffer, this argument must be the length of *param. If the Attribute is defined in the ODBC and param is an integer, this Arguments will be ignored.
SQLINTEGER *	StringLength-Ptr	Output	Returns the length (excluding the NULL-terminatior) of the value returned to Val-uePtr.

2.36.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_ERROR

SQL_INVALID_HANDLE
```

2.36.4 Description

SQLGetStmtAttr () returns the attribute related to the statement handle. For the statement attribute or more information, see SQLSetStmtAttr ().

2.36.5 Diagnosis

SQLSTATE	Description	Comments
HY090	Invalid string or buffer length	StringLength is smaller than 0.
HY092	Invalid attribute or option	The value specified in the attribute argument is not supported by this driver.
HYC00	Option not implemented	The value specified in the attribute argument of ODBC is valid but not supported by this driver.

2.36.6 Related Functions

```
SQLGetConnectAttr

SQLSetConnectAttr

SQLSetStmtAttr
```

2.36.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_plan.cpp

```
SQLGetStmtAttr(stmt, SQL_ATTR_EXPLAIN_PLAN, plan, sizeof(plan), &plan_ind)
```

2.37 SQLGetTypeInfo

SQLGetTypeInfo returns information related to the data types that the database supports. The driver returns the information in the form of an SQL result set. The data type is used in the DDL statement.

2.37.1 Syntax

```
SQLRETURN  SQLGetTypeInfo (
    SQLHSTMT       stmt,
    SQLSMALLINT    type );
```

2.37.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	type	Input	SQL data type

2.37.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.37.4 Description

SQLGetTypeInfo () converts data type information that ALTIBASE HDB provides into the standard result set type. The current result set is sorted by TYPE_NAME in ascending orders to be returned.

2.37.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing between the ODBC and the database is completed
HY000	General error	

SQLSTATE	Description	Comments
HY001	Memory allocation error	Cannot allocate the requested memory for the ODBC to execute the function and complete execution.

2.37.6 Related Functions

`SQLBindCol`

`SQLColAttribute`

`SQLFetch`

2.37.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_info1.cpp

```
if (SQLGetTypeInfo(stmt, SQL_ALL_TYPES) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLGetTypeInfo");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
{
    if ( rc == SQL_ERROR )
    {
       execute_err(dbc, stmt, "SQLGetTypeInfo:SQLFetch");
       break;
    }
    SQLGetData(stmt, 1, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
    SQLGetData(stmt, 2, SQL_C_SSHORT, &DataType, 0, &cbDataType);
    SQLGetData(stmt, 3, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
    SQLGetData(stmt, 9, SQL_C_SSHORT, &Searchable, 0, NULL);
    SQLGetData(stmt, 14, SQL_C_SSHORT, &MinScale, 0, &cbMinScale);
    SQLGetData(stmt, 15, SQL_C_SSHORT, &MaxScale, 0, &cbMaxScale);
    SQLGetData(stmt, 16, SQL_C_SSHORT, &SQLDataType, 0, &cbSQLDataType);
    SQLGetData(stmt, 18, SQL_C_SLONG, &NumPrecRadix, 0, &cbNumPrecRadix);

    printf("%-20s%10d%10d%10d\t",
           szTypeName, DataType, ColumnSize, SQLDataType);
    if ( Searchable == SQL_PRED_NONE )
    {
       printf("SQL_PRED_NONE\n");
    }
    else if ( Searchable == SQL_PRED_CHAR )
    {
       printf("SQL_PRED_CHAR\n");
    }
    else if ( Searchable == SQL_PRED_BASIC )
    {
       printf("SQL_PRED_BASIC\n");
    }
    else if ( Searchable == SQL_SEARCHABLE )
    {
        printf("SQL_SEARCHABLE\n");
    }
}
```

2.38 SQLMoreResult

SQLMoreResult returns whether there is more result.

2.38.1 Syntax

```
SQLRETURN SQLMoreResults (
    SQLHSTMT    stmt);
```

2.38.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Instruction Handle

2.38.3 Result Values

```
SQL_SUCCESS

SQL_NO_DATA_FOUND
```

2.38.4 Description

When a statement is fetched to obtain its result, the corresponding cursor is located at the first record of the result set. Depending on an application, when its result set is used, a further check is performed to see if there is more result left. If there are more result values left, SQL_SUCCESS is returned, and if not, SQL_NO_DATA_FOUND is returned.

If the result set is fetched, the cursor is located at the first result set. Depending on an application, when its result set is used, a further check is performed to see if there is more result left. If there are more result values left, SQL_SUCCESS is returned, and if not, SQL_NO_DATA_FOUND is returned.

2.38.5 Related Function

```
SQLFetch
```

2.39 SQLNativeSql

This coverts SQL statements to statements supported by ODBC driver. Unicode SQLNativeSqlW() supports same execution as SQLNativeSql().

2.39.1 Syntax

```
SQLRETURN SQLNativeSql (
    SQLHDBC        dbc,
    SQLCHAR        *InstatementText,
    SQLINTEGER     textLength,
    SQLCHAR        *OutStstementText,
    SQLINTEGER     bufferLength,
    SQLINTEGER     textLength);
```

2.39.2 Argument

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	In	Connection Handle
SQLCHAR *	Instatement-Text	In	SQL Statements to Convert
SQLINTEGER	textLength	In	The length, in bytes, of the contents of the *InstatementText* argument
SQLCHAR *	OutStste-mentText	Out	Buffer Pointer recieving the converted SQL statements
SQLINTEGER	bufferLength	In	Size of OutStstementText
SQLINTEGER	textLength	Out	Size specified in OutStstementText

2.39.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.39.4 Description

This converts SQL statements to statements supported by ODBC driver.

2.39.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
01004	Cutted Source	Buffer size of OutStstementText is lesser than the size of returned data.

2.39.6 Example

```
SQLNativeSql(dbc,
                 (SQLCHAR*)"INSERT INTO T1 VALUES( {d '1981-09-27'} )",
                  SQL_NTS,
                  outText,
                  100,
                  &outTextLen);
```

INSERT INTO T1 VALUES(TO_DATE('1981-09-27', 'YYYY-MM-DD')) is saved in outText.

2.40 SQLNumParams

SQLNumParams returns the number of parameters in an SQL statement.

2.40.1 Syntax

```
SQLRETURN  SQLNumParams (
    SQLHSTMT        stmt,
    SQLSMALLINT     *parameterCounterPtr );
```

2.40.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT *	parameter-CountPtr	Output	Pointer of the number of the parameters

2.40.3 Return Values

```
SQL_SUCCESS
```

2.40.4 Description

This function can be called only after SQLPrepare () is called.

If the stmt does not include parameters, SQLNumParams () will set 0 in *parameterCountPtr.

2.40.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.40.6 Related Functions

```
SQLBindParameter

SQLDescribeParam
```

2.40.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_info2.cpp

```
SQLPrepare(hstmt, Statement, SQL_NTS);
// Check to see if there are any parameters. If so, process them.
SQLNumParams(hstmt, &NumParams);
if (NumParams) {
 // Allocate memory for three arrays. The first holds pointers to buffers in 
which
 // each parameter value will be stored in character form. The second con-
tains
 the
 // length of each buffer. The third contains the length/indicator value for 
each
 // parameter.
 PtrArray = (SQLPOINTER *) malloc(NumParams * sizeof(SQLPOINTER));
 BufferLenArray = (SQLINTEGER *) malloc(NumParams * sizeof(SQLINTEGER));
 LenOrIndArray = (SQLINTEGER *) malloc(NumParams * sizeof(SQLINTEGER));
 for (i = 0; i < NumParams; i++) {
 // Describe the parameter.
 SQLDescribeParam(hstmt, i + 1, &DataType, &ParamSize, &DecimalDigits, 
&Nullable);
 // Call a helper function to allocate a buffer in which to store the parame-
ter
 // value in character form. The function determines the size of the buffer 
from
 // the SQL data type and parameter size returned by SQLDescribeParam and 
returns
 // a pointer to the buffer and the length of the buffer.
 PtrArray[i] = (char*)malloc(ParamSize);
 BufferLenArray[i] = SQL_NTS;
 // Bind the memory to the parameter. Assume that we only have input parame-
ters.
 SQLBindParameter(hstmt, i + 1, SQL_PARAM_INPUT, SQL_C_CHAR, DataType, Param-
Size,
 DecimalDigits, PtrArray[i], BufferLenArray[i],
 &LenOrIndArray[i]);
 // Prompt the user for the value of the parameter and store it in the memory
 // allocated earlier. For simplicity, this function does not check the value
 // against the information returned by SQLDescribeParam. Instead, the driver 
does
 // this when a statement is executed.
 strcpy((char*)PtrArray[i], "AAAAAAA");
 BufferLenArray[i] = 7;
 }
}
```

2.41 SQLNumResultCols

SQLNumResultCols returns the number of columns in a result set.

2.41.1 Syntax

```
SQLRETURN  SQLNumResultCols (
    SQLHSTMT        stmt,
    SQLSMALLINT     *col );
```

2.41.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT *	col	Output	Pointer to save the number of columns in a result set.

2.41.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.41.4 Description

If the commands related to stmt do not return columns, SQLNumResultCols () will set 0 in *col.

This function can be called only after SQLPrepare () or SQLExecDirect () is called. After this function is called, SQLDescribeCol (), SQLColAttribute (), SQLBindCol () or SQLGetData () can be called.

When the command is in ready, executed, or defined status, SQLNumResultCols () can be successfully called.

2.41.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	

If SQLNumResultCols () is called after SQLPrepare () and before SQLExecute (), no SQLSTATE returned by these two functions can be returned.

2.41.6 Related Functions

```
SQLBindCol

SQLColAttribute

SQLDescribeCol

SQLExecDirect

SQLExecute

SQLFetch

SQLGetData

SQLPrepare

SQLSetStmtAttr
```

2.41.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex1.cpp

```
sprintf(query,"SELECT * FROM DEMO_EX1");
if (SQLExecDirect(stmt,query, SQL_NTS) != SQL_SUCCESS)
{
 execute_err(dbc, stmt, query);
 SQLFreeStmt(stmt, SQL_DROP);
 return SQL_ERROR;
}
SQLNumResultCols(stmt, &columnCount);
```

2.42 SQLParamData

You can use this when inserting data while running command.

2.42.1 Syntax

```
SQLRETURN SQLParamData (
    SQLHSTMT       stmt,
    SQLPOINTER     *value);
```

2.42.2 Argument

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	In	Command Handle
SQLPOINTER *	value	Out	Pointer to save address specified in SQL-BindParameter

2.42.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_NEED_DATA

SQL_NO_DATA

SQL_INVALID_HANDLE

SQL_ERROR
```

2.42.4 Description

You can use this with SQLPutData when inserting data while running command.

2.42.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly

SQLSTATE	Description	Comments
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY010	Continuous Function Error	

2.42.6 Related Function

```
SQLBindParameter

SQLExecDirect

SQLExecute

SQLPutData
```

2.43 SQLPrepare

This prepares an SQL string for execution. SQLPrepareW() as a Unicode string supports same execution as SQLPrepare().

2.43.1 Syntax

```
SQLRETURN  SQLPrepare (
    SQLHSTMT       stmt,
    SQLCHAR        *sql,
    SQLSMALLINT    sqlLength );
```

2.43.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	sql	Input	SQL text string column
SQLSMALLINT	sqlLength	Input	The length, in bytes, of the contents of the *sql* argument

2.43.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.43.4 Description

An application calls SQLPrepare () to send an SQL statement to the database. The application may include more than one parameter markers (?) in the SQL statement. To include the parameter marker, an application inserts the parameter marker in the SQL character string at a proper position.

Once the statement is ready, an application calls the function. To refer to the statement, use the statement handle. The statements related to the statement handle can be executed again when an application calls SQLFreeStmt () using SQL_DROP option to release the statement or when the statement handle is used again to call SQLPrepare (), SQLExecDirect (), or catalog function such like SQLColumns (), SQLTables (). Once an application prepares the statement, an application an request information about the result set format. Calling SQLDescribeCol () after SQLPrepare () may not be as effective as calling it after SQLExecute () or SQLExecDirect ().

The ODBC data type will be inspected when the command is executed (in case not all parameters are bound.) For maximum inter-operation, an application must unbound all parameters applied to the previous SQL statement before the same command prepares a new SQL statement. This prevents errors that occur due to pervious parameters applied to new information.

2.43.5 Diagnosis

SQLSTATE	Description	Comments
08003		When stmt is not connected or the connection is released
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	
HY001	Memory allocation error	Cannot allocate the requested memory for the ODBC to execute the function and complete execution.
HY009	Use an invalid pointer (null pointer)	sql is a NULL pointer

2.43.6 Related Functions

```
SQLAllocHandle

SQLBindCol

SQLBindParameter

SQLEndTran

SQLExecDirect

SQLExecute

SQLRowCount
```

2.43.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex2.cpp

```
sprintf(query,"INSERT INTO DEMO_EX2 VALUES( ?, ?, ?, ?, ?, ? )");

/* Prepare for a statement and bind the variable. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
    SQLFreeStmt(stmt, SQL_DROP);
```

```
        return SQL_ERROR;
    }
```

2.44 SQLPrimaryKeys

SQLPrimaryKeys returns the column names that make up 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.

SQLPrimarykeysW() as a Unicode string supports same execution as SQLPrimarykey().

2.44.1 Syntax

```
SQLRETURN  SQLPrimaryKeys (
    SQLHSTMT       stmt,
    SQLCHAR        *cName,
    SQLSMALLINT    cNameLength,
    SQLCHAR        *sName,
    SQLSMALLINT    sNameLength,
    SQLCHAR        *tName,
    SQLSMALLINT    tNameLength );
```

2.44.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	cName	Input	Catalog Name
SQLSMALLINT	cNameLength	Input	The length, in bytes, of **cName*
SQLCHAR *	sName	Input	Schema Name
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	tName	Input	The table name. Cannot be a NULL pointer. tName cannot contain a string search pattern.
SQLSMALLINT	tNameLength	Input	The length, in bytes, of **tName*

2.44.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.44.4 Description

SQLPrimaryKeys () returns the results in the format of standard result set which is sorted in order byTABLE_CAT, TABLE_SCHEM, TABLE_NAME, and KEY_SEQ.

The search type cannot be used to define the schema-defining arguments or the table name.

In many cases, calling of SQLPrimaryKeys () is mapped on the search that is complex and cost a lot. Therefore, the stored result set must be used instead of repeating the calling.

2.44.4.1 Columns Returned by SQLPrimaryKeys ()

The following table lists the columns of the result set.

No.	Name	Data Type	Description
1	TABLE_CAT	VARCHAR	Null will be always returned.
2	TABLE_SCHEM	VARCHAR	Primary key table schema name
3	TABLE_NAME	VARCHAR (NOT NULL)	Primary key table name
4	COLUMN_NAME	VARCHAR (NOT NULL)	First primary key column name
5	KEY_SEQ	SMALLINT (NOT NULL)	Column orders of the first primary key starting with 1
6	PK_NAME	VARCHAR	First primary key name

2.44.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	
HY009	Invalid Arguments used (null pointer).	tNameTransfer NULL pointer

2.44.6 Related Functions

`SQLBindCol`

`SQLFetch`

`SQLStatistics`

2.44.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta3.cpp

```
if (SQLPrimaryKeys(stmt,
                   NULL, 0,
                   NULL, 0,
                   (char*)"DEMO_META3", SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPrimaryKeys");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
SQLBindCol(stmt, 1, SQL_C_CHAR, szCatalog, STR_LEN,&cbCatalog);
SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, STR_LEN,&cbTableName);
SQLBindCol(stmt, 4, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
SQLBindCol(stmt, 5, SQL_C_SSHORT, &KeySeq, 0, &cbKeySeq);
SQLBindCol(stmt, 6, SQL_C_CHAR, szPkName, STR_LEN, &cbPkName);
while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
{
    if ( rc == SQL_ERROR )
    {
       execute_err(dbc, stmt, "SQLPrimaryKeys:SQLFetch");
       break;
    }
    printf("%-20s%-20s%-3d%-20s\n", szTableName,
                  szColumnName, KeySeq, szPkName);
}
```

2.45 SQLProcedureColumns

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 statement.

SQLProcedureColumnsW() as a Unicode string supports same execution as SQLProcedureColumns().

2.45.1 Syntax

```
SQLRETURN  SQLProcedureColumns (
    SQLHSTMT        stmt,
    SQLCHAR         *cName,
    SQLSMALLINT     cNameLength,
    SQLCHAR         *sName,
    SQLSMALLINT     sNameLength,
    SQLCHAR         *pName,
    SQLSMALLINT     pNameLength,
    SQLCHAR         *colName,
    SQLSMALLINT     colNameLength );
```

2.45.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	cName	Input	Procedure Catalog Name
SQLSMALLINT	cNameLength	Input	The length, in bytes, of **cName*
SQLCHAR *	sName	Input	Procedure Schema Name
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	pName	Input	Procedure name. Cannot be a NULL pointer. *pName* must not include the character string search pattern.
SQLSMALLINT	pNameLength	Input	The length, in bytes, of **pName*
SQLCHAR *	colName	Input	Column Name
SQLSMALLINT	colName-Length	Input	The length, in bytes, of **colName*

2.45.3 Return Values

```
SQL_SUCCESS
```

```
SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.45.4 Description

This function is used before a statement is executed to retrieve the procedure parameters and the columns that form the result sets returned by the procedure.

SQLProcedureColumns () returns the results in the standard result set sorted in order by PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, and COLUMN_TYPE. The column names will be returned for each procedure in the following order: Name of returned value, each parameter names to call the procedures (or calling order), and column names of the result set returned by the procedure (or column order).

An application must bind the unique columns which contains the ODBC driver information to the last data of the result set.

2.45.4.1 Columns Returned by SQLProcedureColumns ()

Name	No.	Data Type	Description
PROCEDURE_CAT	1	VARCHAR	Returns NULL always
PROCEDURE _SCHEM	2	VARCHAR	Procedure schema name; NULL if not applicable to the database
PROCEDURE _NAME	3	VARCHAR (NOT NULL)	Procedure name
COLUMN_NAME	4	VARCHAR (NOT NULL)	Procedure column name. The driver returns an empty string for a procedure column that does not have a name.
COLUMN_TYPE	5	SMALLINT (NOT NULL)	Defines the procedure column as a parameter or a result set column: SQL_PARAM_INPUT: The procedure column is the input parameter. SQL_PARAM_INPUT_OUTPUT: The procedure column is the input/output parameters. SQL_PARAM_OUTPUT: The procedure column is the output parameter.
DATA_TYPE	6	SMALLINT (NOT NULL)	SQL data type
TYPE_NAME	7	VARCHAR (NOT NULL)	Name of the data type corresponding to the database

Name	No.	Data Type	Description
COLUMN_SIZE	8	INTEGER	Column Size. NULL will be returned when the column size is not proper.
BUFFER_LENGTH	9	INTEGER	The maximum byte storing the data
DECIMAL_DIGITS	10	SMALLINT	The NULL will return the data type that cannot apply the decimal points of the string and the decimal points.
NUM_PREC_RADIX	11	SMALLINT	In case of the numeric data type 10: For COLUMN_SIZE and DECIMAL_DIGIT, decimal digits allowable in this string is be given. For example, DECIMAL(12,5) string can return NUM_PREC_RADIX 10, COLUMN_SIZE 12, and DECIMAL_DIGITS 5.
NULLABLE	12	SMALLINT (NOT NULL)	SQL_NO_NullS when the procedure column does not allow NULL value, or SQL_Nullable when NULL is allowed.
REMARKS	13	VARCHAR	Description of the procedure column
COLUMN_DEF	14	VARCHAR	The default value of the column. If NULL was specified as the default value, this column is the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, this column contains TRUNCATED, with no enclosing single quotation marks. If no default value was specified, this column is NULL. The value of COLUMN_DEF can be used in generating a new column definition, except when it contains the value TRUNCATED.
SQL_DATA_TYPE	15	SMALLINT (NOT NULL)	SQL data type
SQL_DATETIME_SUB	16	SMALLINT	The subtype code for datetime and interval data types. For other data types, this column returns a NULL.
CHAR_OCTET_LENGTH	17	INTEGER	Maximum digits of the character string or binary data-type string.
ORDINAL_POSITION	18	INTEGER (NOT NULL)	Location of the order of the parameters in the procedure definition (starting with 1)
IS_NULLABLE	19	VARCHAR	NO : When the string does not contain the NULL YES : When the string contain the NULL

2.45.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	
HY009	Invalid Arguments used (null pointer).	CName is a NULL pointer

2.45.6 Related Functions

SQLBindCol

SQLFetch

SQLProcedures

2.45.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta6.cpp

```
if (SQLProcedureColumns(stmt,
                        NULL, 0,
                        NULL, 0,
                        "DEMO_META6_PROC", SQL_NTS,
                        NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLProcedureColumns");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
 SQLBindCol(stmt, 1, SQL_C_CHAR, szCatalog, STR_LEN,&cbCatalog);
 SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
 SQLBindCol(stmt, 3, SQL_C_CHAR, szProcName, STR_LEN,&cbProcName);
 SQLBindCol(stmt, 4, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
 SQLBindCol(stmt, 5, SQL_C_SSHORT, &ColumnType, 0, &cbColumnType);
 SQLBindCol(stmt, 7, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
 SQLBindCol(stmt, 8, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
 SQLBindCol(stmt, 10, SQL_C_SSHORT, &DecimalDigits, 0, &cbDecimalDigits);
 SQLBindCol(stmt, 11, SQL_C_SSHORT, &NumPrecRadix, 0, &cbNumPrecRadix);
 SQLBindCol(stmt, 18, SQL_C_SLONG, &OrdinalPosition, 0, &cbOrdinalPosition);
```

2.46 SQLProcedures

SQLProcedures returns the list of procedure names stored in a specific database. Procedure is a generic term used to describe an executable object, or a named entity that can be invoked using input and output parameters.

SQLProceduresW() as a Unicode string supports same execution as SQLProcedures().

2.46.1 Syntax

```
SQLRETURN  SQLProcedures (
    SQLHSTMT       stmt,
    SQLCHAR        *cName,
    SQLSMALLINT    cNameLength,
    SQLCHAR        *sName ,
    SQLSMALLINT    sNameLength,
    SQLCHAR        *pName,
    SQLSMALLINT    pNameLength );
```

2.46.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	cName	Input	Procedure Catalog Name
SQLSMALLINT	cNameLength	Input	The length, in bytes, of **cName*
SQLCHAR *	sName	Input	Procedure Schema Name
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	pName	Input	Procedure name. Cannot be a NULL pointer. pName must not contain the string search-patterns.
SQLSMALLINT	pNameLength	Input	The length, in bytes, of **pName*

2.46.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.46.4 Description

SQLProcedures () displays the list of all procedures in the required range. A user may have or have not a privilege to use and execute these procedures.

SQLProcedures () returns the results in the format of the standard result set format sorted in order by PROCEDURE_CAT, PROCEDURE_SCHEM, and PROCEDURE_NAME.

Note SQLProcedures () may not return all procedures. An application can use the valid procedures whether the procedure is returned by SQLProcedures () or not.

2.46.4.1 The Column Returned by SQLProcedures ()

Name	No.	Data Type	Description
PROCEDURE_CAT	1	VARCHAR	Procedure catalog identifier; NULL if not applicable to the database..
PROCEDURE _SCHEM	2	VARCHAR	Procedure schema identifier; NULL if not applicable to the database
PROCEDURE _NAME	3	VARCHAR (NOT NULL)	Procedure identifier
NUM_INPUT_PARAMS	4	N/A	Reserved for future use. An application must not apply any returned data to this result string.
NUM_OUTPUT_PARAMS	5	N/A	Reserved for future use. An application must not apply any returned data to this result string.
NUM_RESULT_SETS	6	N/A	Reserved for future use. An application must not apply any returned data to this string.
REMARKS	7	VARCHAR	Procedure description
PROCEDURE_TYPE	8	SMALLINT	Definition of the procedure type SQL_PT_UNKNOWN: It cannot be determined whether the procedure returns a value. SQL_PT_PROCEDURE: The returned object is a procedure; that is, it does not have a return value. SQL_PT_FUNCTION: The returned object is a function; that is, it has a return value.

2.46.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	
HY009	Invalid Arguments used (null pointer).	CName is a NULL pointer sNme, pName are NULL pointer

2.46.6 Related Functions

SQLBindCol

SQLFetch

SQLProcedureColumns

2.46.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta5.cpp

```
if (SQLProcedures(stmt,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLProcedures");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

2.47 SQLPutData

You can use this when inserting data while running command.

2.47.1 Syntax

```
SQLRETURN SQLPutData (
    SQLHSTMT      stmt,
    SQLPOINTER    data,
    SQLLEN        strLength);
```

2.47.2 Argument

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	In	Command Handle
SQLPOINTER	data	In	Pointer of Data Buffer
SQLLEN	strLength	In	Data Size

2.47.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.47.4 Description

You can use this with SQLParamData when inserting data while running command.

2.47.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY010	Continuous Function Error	

2.47.6 Related Function

`SQLBindParameter`

`SQLExecDirect`

`SQLExecute`

`SQLParamData`

2.48 SQLRowCount

SQLRowCount returns the number of rows affected by an UPDATE, DELETE,or INSERT statement.

2.48.1 Syntax

```
SQLRETURN  SQLRowCount (
    SQLHSTMT       stmt,
    SQLINTEGER     *row );
```

2.48.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLINTEGER *	row	Output	Pointer to save the number of row

2.48.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.48.4 Description

If the most recent statement referred to by the input statement handle is not UPDATE,DELETE, or INSERT statement or if it is not successfully executed, this function returns *row as -1.

However, if no row was affected after UPDATE, DELETE, or INSERT statement or if SELECT statement is submitted,it returns *row as 0.

The rows in the tables affected by the SQL statement such as cascading delete operationare not included.

Before this function is called, SQLExecute () or SQLExecDirect () must be called.

2.48.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.48.6 Related Functions

SQLExecDirect

SQLExecute

2.48.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_exa5.cpp

```
if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, query);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
SQLRowCount(stmt, &row_count);
```

2.49 SQLSetConnectAttr

SQLSetConnectAttr() sets attributes that govern aspects of connections.

SQLSetConnectAttrW() as a Unicode string supports same execution as SQLSetConnectAttr().

2.49.1 Syntax

```
SQLRETURN  SQLSetConnectAttr (
    SQLHDBC       dbc,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    StringLength );
```

2.49.2 Arguments

Data Type	Argument	In/Out	Description
SQLHDBC	dbc	Input	Connection Handle
SQLINTEGER	Attribute	Input	Attribute to set
SQLPOINTER	ValuePtr	Input	Pointer of value contaied attribute. Depending on the Attribute value, the *ValuePtr* will be either a 32-bit unsigned integer or a pointer indicating the Null-terminator string. SQL_ATTR_AUTOCOMMIT SQL_ATTR_CONNECTION_TIMEOUT SQL_ATTR_PORT SQL_ATTR_TXN_ISOLATION ALTIBASE_APP_INFO ALTIBASE_DATE_FORMAT
SQLINTEGER	StringLength	Input	When *ValuePtr* is the character string, *StringLength* is the length of the character string or SQL_NTS. If *ValuePtr* is 32-bit integer, this argument is ignored.

2.49.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.49.4 Description

An application can call SQLSetConnectAttr () at any point whether the connection is allocated or released. All connections successfully set by an application and statement attribute are stored until SQLFreeHandle () is called in the current connection.

2.49.4.1 Connection Attribute

Attribute	Contents
SQL_ATTR_AUTOCOMMIT	32-bit value to set the commit mode. SQL_AUTOCOMMIT_ON: Each SQL statement is automatically committed SQL_AUTOCOMMIT_OFF: Not automatically committed.
SQL_ATTR_CONNECTION_TIMEOUT	Set timeout value to prevent blocking that may occur in select() or poll() in an unstable network
SQL_ATTR_PORT	Server port setting (32-bit Integer)
SQL_ATTR_TXN_ISOLATION	32-bit value that sets the transaction isolation level for the current connection referred to by dbc.
ALTIBASE_APP_INFO	Specify an identifier for an ODBC client to obtain more detailed information on the user's session.
ALTIBASE_DATE_FORMAT	Sets the date format. Supports YYYY/MM/DD HH:MI:SS as the default.

2.49.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	

In case of the statement attribute, SQLSetConnectAttr () can return any SQL state returned by SQLSetStmtAttr ().

2.49.6 Related Functions

```
SQLAllocHandle
```

2.49.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_tran1.cpp

```
if (SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
 (void*)SQL_AUTOCOMMIT_OFF, 0) != SQL_SUCCESS)
{
    execute_err(dbc, SQL_NULL_HSTMT, "Autocommit OFF");
    return SQL_ERROR;
}
```

2.50 SQLSetDescField

This specifies an attribute of descriptor. Unicode SQLSetDescFieldW() supports same execution as SQLSetDescField().

2.50.1 Syntax

```
SQLRETURN SQLSetDescField (
    SQLHDESC       desc,
    SQLSMALLINT    recNumber,
    SQLSMALLINT    fieldIdentifier,
    SQLPOINTER     value,
    SQLINTEGER     bufferLength);
```

2.50.2 Argument

Data Type	Argument	In/Out	Description
SQLHDESC	desc	In	Descriptor Handle
SQLSMALLINT	renNumber	In	This starts from 1 of column number.
SQLSMALLINT	fieldIdentifier	In	This specifies the attribute of column to retrieve.
SQLPOINTER	value	In	Buffer pointer specifying attributes
SQLINTEGER	bufferLength	In	Value size

2.50.3 Return Value

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.50.4 Description

This specifies an attribute of descriptor.

2.50.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General Error	No error occurs explicitly.
HY001	Memory Allocation Error	This denotes to fail to allocate memory for handle.
HY010	Order Error of Calling Function	
07009	Invalid Descriptor Index	The value of recNumber is incorrect.

2.50.6 Related Function

`SQLBindCol`

`SQLBindParameter`

`SQLGetDescField`

`SQLGetDescRec`

2.51 SQLSetEnvAttr

SQLSetEnvAttr() sets the environment attribute for the current environment.

2.51.1 Syntax

```
SQLRETURN  SQLSetEnvAttr (
    SQLHENV        env,
    SQLINTEGER     Attribute,
    SQLPOINTER     Value,
    SQLINTEGER     StringLength );
```

2.51.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV	env	Input	Environment Handle
SQLINTEGER	Attribute	Input	Environment attribute to set.
SQLPOINTER	Value	Input	Pointer of the value related to the *Attribute*. Depending on the Attribute, the *Value* will be either a 32-bit unsigned integer or a pointer indicating the Null-terminator string.
SQLINTEGER	StringLength	Input	In case the attribute is the character, it is the length of **Value*.

2.51.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.51.4 Description

An application can call SQLSetEnvAttr () only when no connection handle is allocated in the current environment. All environment attribute successfully set in an application are stored till SQLFreeHandle () is called in the current environment.

2.51.4.1 Environment Attributes

Attribute	Contents
SQL_ATTR_ODBC_VERSION	(Applied only in Win32.) SQL_OV_ODBC3 or SQL_OV_ODBC2

2.51.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.51.6 Related Functions

`SQLAllocHandle`

2.52 SQLSetStmtAttr

SQLSetStmtAttr() sets the attribute related to the statement handle.

SQLSetStmtAttrW() as a Unicode string supports same execution as SQLSetStmtAttr().

2.52.1 Syntax

```
SQLRETURN  SQLSetStmtAttr (
    SQLHSTMT      stmt,
    SQLINTEGER    Attribute,
    SQLPOINTER    param,
    SQLINTEGER    StringLength );
```

2.52.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV	stmt	Input	Statement handle

Data Type	Argument	In/Out	Description
SQLINTEGER	Attribute	Input	Attribute to set, Supported attribute value: SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_SCROLLABLE, SQL_ATTR_CURSOR_SENSITIVITY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_PARAM_BIND_TYPE, SQL_ATTR_PARAM_STATUS_PTR, SQL_ATTR_PARAMS_PROCESSED_PTR, SQL_ATTR_PARAMS_ROW_COUNTS_PTR, SQL_ATTR_PARAMS_SET_ROW_COUNTS, SQL_ATTR_PARAMSET_SIZE, SQL_ATTR_ROW_ARRAY_SIZE, SQL_ATTR_ROW_BIND_TYPE, SQL_ATTR_ROW_STATUS_PTR, SQL_ATTR_ROWS_FETCHED_PTR ALTIBASE_STMT_ATTR_ATOMIC_ARRAY Attribute value which is not currently being supported: SQL_ATTR_APP_PARAM_DESC, SQL_ATTR_APP_ROW_DESC, SQL_ATTR_ASYNC_ENABLE, SQL_ATTR_ENABLE_AUTO_IPD, SQL_ATTR_FETCH_BOOKMARK_PTR, SQL_ATTR_IMP_PARAM_DESC, SQL_ATTR_IMP_ROW_DESC, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_METADATA_ID, SQL_ATTR_NOSCAN, SQL_ATTR_PARAM_BIND_OFFSET_PTR, SQL_ATTR_PARAM_OPERATION_PTR, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_RETRIEVE_DATA, SQL_ATTR_ROW_BIND_OFFSET_PTR, SQL_ATTR_ROW_NUMBER, SQL_ATTR_ROW_OPERATION_PTR, SQL_ATTR_SIMULATE_CURSOR, SQL_ATTR_USE_BOOKMARKS
SQLPOINTER	param	Input	Pointer of the value related to the Attribute Depending on the pointer, Attribute, the *param* will be a 32-bit integer, the pointer of the Null-terminator, the character string's pointer, the binary pointer, or the value defined in the ODBC. If Attribute is the unique value of the ODBC, *param* is the integer number with a sign.

Data Type	Argument	In/Out	Description
SQLINTEGER	StringLength	Input	If *Attribute* has been defined in the ODBC and *param* indicates the character string or binary buffer, this argument must be the byte length of **param*. If *Attribute* has been defined in the ODBC and *param* is an integer, this argument is ignored.

2.52.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.52.4 Description

The command option for stmt is valid until the option is changed by calling of SQLSetStmtAttr () or till stmt is removed by calling of SQLFreeHandle (). Handle release method: Calling SQL_CLOSE, SQL_UNBIND, or SQL_RESET_PARAMS with SQLFreeStmt () does not reset the statement attribute.

To use a column-wise binding, set SQL_ATTR_PARAM_BIND_TYPE in Arguments Attribute of an application function SQLSetStmtAttr() and set SQL_PARAM_BIND_BY_COLUMN in param. ARRAY_SIZE changes only PARAM_SIZE at every execution moment. PARAM_STATUS_PTR is the vaule to return if each column will be executed, and specifies SQLSMALLINT array. If succeeds, SQL_SUCCESS, SQL_SUCCESS_WITH_INFO will be returned, and if fails, then SQL_PARAM_UNUSED will be returned respectively.

For Macro values: SQL_PARAM_SUCCESS is 0SQL_PARAM_ERROR is 5, SQL_PARAM_SUCCESS_WITH_INFO is 6SQL_PARAM_UNUSED is 7.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, SQL_PARAM_BIND_BY_COLUMN);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
```

Designates the pointer of the variables to store the number of columns processed by PARAMS_PROCESSED_PTR. The pointer type of SQLINTEGER. Then, execute SQLBindParameter () like before.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);
```

then, Executing

```
SQLBindParameter().
```

When using the row-wise binding, define the size of the structure in PARAM_Bind_Type, unlike in the column-wise binding.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, sizeof(struct…));
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, ARRAY_SIZE, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_STATUS_PTR, ParamStatusArray, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMS_PROCESSED_PTR, &ParamsProcessed, 0);
```

then, Execute

```
SQLBindParameter()
```

2.52.4.1 Statement Attributes

Attribute	Contents
SQL_ATTR_CONCURRENCY	SQLUINTEGER value specifying the temporary processing of the cursor: SQL_CONCUR_READ_ONLY: The cursor supports read-only. Updating is not allowed.
SQL_ATTR_CURSOR_SCROLLABLE	32-bit integer to designate whether the open cursor can be scrolled for this statement handle. (Supports only SQL_FETCH_NEXT.)
SQL_ATTR_CURSOR_SENSITIVITY	Not used
SQL_ATTR_CURSOR_TYPE	SQLUINTEGER indicating the cursor type. SQL_CURSOR_FORWARD_ONLY: The cursor only proceeds forward. SQL_CURSOR_STATIC: The data located in the result set are static.
SQL_ATTR_PARAM_BIND_TYPE	Setting value which is necessary for array binding To retrieve the data by the column-wise binding, SQL_PARAM_BIND_BY_COLUMN (default) is set in this field. To retrieve the data by the row-wise binding, the length of the structure or the length of the buffer to which the dynamic parameter will be bound will be set. This length must include the bound parameter space.

Attribute	Contents
SQL_ATTR_PARAM_STATUS_PTR	(Necessary for array binding) This field is requested only when PARAMSET_SIZE is higher than 1. Status values are as follows: • SQL_PARAM_SUCCESS: An SQL statement has been successfully executed. The macro value is 0. • SQL_PARAM_SUCCESS_WITH_INFO: An SQL statement has been successfully executed.: however, the warning data can be viewed from the diagnosis data structure. The macro value is 5. • SQL_PARAM_ERROR: Error occurs during the execution of parameters. additional error information can be viewed on the diagnosis data structure. The macro value is 6. • SQL_PARAM_UNUSED: This parameter set is not used partly because an error has occurred preventing the previous parameter from further proceeding. The macro value is 7.
SQL_ATTR_PARAMS_PROCESSED_PTR	(Necessary for array binding) Indicates the buffer that return the number of parameters including the errors. In case of the NULL pointer, no data will be returned. If SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is not returned upon calling of SQLExecDirect () or SQLExecute (), the contents of the buffer will not be defined.
SQL_ATTR_PARAMS_ROW_COUNTS_PTR	(Necessary for array binding) Sets the buffer that returns the following values as the result of SQLExecute to array binding. SQL_SUCCESS: When an SQL statement was successfully executed for all array elements SQL_ERROR When even one array element fails to execute SQL statement SQL_NO_DATA: When no array element was changed (inputted or deleted)
SQL_ATTR_PARAMS_SET_ROW_COUNTS	(Necessary for array binding) SQL_ROW_COUNTS_ON: Returns the number of records changed by each array element. In other words, if there are changed records, the number of records will be returned. If there is no record changed, 0 will be returned. In case an error occurs, SQL_USHRT_MAX (65534) will be returned. SQL_ROW_COUNTS_OFF: Existing behaviors of attribute SQL_ATTR_PARAM_STATUS_PTR will be maintained.

Attribute	Contents
SQL_ATTR_PARAMSET_SIZE	(Necessary for array binding) SQLUINTEGER value that indicates the number of values for each parameter.
SQL_ATTR_ROW_ARRAY_SIZE	(Necessary setting value at the time of the array fetch in the SELECT statement) SQLUINTEGER that indicates the number of rows returned by calling each SQLFetch ().
SQL_ATTR_ROW_BIND_TYPE	(Necessary setting value at the time of the array fetch in the SELECT statement) SQLUINTEGER that sets the direction of the binding to be used upon calling of SQLFetch (). The column-wise binding is searched by setting SQL_BIND_BY_COLUMN. The row-wise binding searches the data by setting the length of the structure or the length of the buffer to which the results columns will be bound. In case the length is specified, the space for the columns to be bound must be included.
SQL_ATTR_ROW_STATUS_PTR	(Necessary setting value at the time of the array fetch in the SELECT statement) The size of the array is same as the number of rows of the row set. In this attribute, the NULL pointer can be set. In this case, the ODBC does not return the column status value.
SQL_ATTR_ROWS_FETCHED_PTR	(Necessary setting value at the time of the array fetch in the SELECT statement) SQLUINTEGER* value indicating the buffer that returns the number of fetched rows after SQLFetch () is called

Attribute	Contents
ALTIBASE_STMT_ATTR_ATOMIC_ARRAY	This is ALTIBASE HDB only attribute indicating to execute Atomic Array Insert. Array Insert can execute the entire statements respectively, whereas Atomic Array Insert can execute several statements at one execution and this results in them executed as a single statement in other words. You may specify ALTIBASE_STMT_ATTR_ATOMIC_ARRAY in Attribute and SQL_TRUE/SQL_FALSE in the param argument. If you choose SQL_TRUE, you can execute Atomic Array Insert. However, you must specify Array Size in order that Atomic Array Insert has better performance than the existing Array Insert does.(SQL_ATTR_PARAMSET_SIZE) And you can save result values with using the following attributes. SQL_ATTR_PARAM_STATUS_PRT : Real result values are saved only in the first row and the rest is successfully processed. SQL_ATTR_PARAMS_PROCESSED_PTR : Real result values are saved only in the first row and the rest of them are ignored.

2.52.5 Diagnosis

SQLSTATE	Description	Comments
HY000	General error	

2.52.6 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_ex4.cpp

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE, (void*)sizeof(demo_ex4_data),
0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (void*)10, 0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMS_PROCESSED_PTR,(void*) &processed_ptr,
0);
SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_STATUS_PTR, (void*)status, 0);
```

You may use Automic Array Insert.

```
SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER) array_size, 0); //
Specify Array Size.
SQLSetStmtAttr(stmt, ALTIBASE_STMT_ATTR_ATOMIC_ARRAY, (SQLPOINTER) SQL_TRUE,
0); // Specify Atomic attribute.
```

2.53 SQLSpecialColumns

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.

SQLSpecialColumnsW() as a Unicode string supports same execution as SQLSpecialColumns().

2.53.1 Syntax

```
SQLRETURN  SQLSpecialColumns (
    SQLHSTMT       stmt,
    SQLSMALLINT    fColType,
    SQLCHAR        *szTableQual,
    SQLSMALLINT    cbTableQual,
    SQLCHAR        *szTableOwner,
    SQLSMALLINT    cbTableOwner,
    SQLCHAR        *szTableName,
    SQLSMALLINT    cbTableName,
    SQLSMALLINT    fScope,
    SQLSMALLINT    fNullable );
```

2.53.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLSMALLINT	fColType	Input	Type of the column to be returned SQL_BEST_ROWID: Returns the optimal column that uniquely identify the rows in the table by searching column value(s).
SQLCHAR *	szTableQual	Input	Null will be always returned.
SQLSMALLINT	cbTableQual	Input	The length, in bytes, of **szTableQual*
SQLCHAR *	szTableOwner	Input	Schema Name
SQLSMALLINT	cbTable-Owner	Input	The length, in bytes, of **szTableOwner*
SQLCHAR *	szTableName	Input	The table name. Cannot be a NULL pointer.
SQLSMALLINT	cbTableName	Input	The length, in bytes, of **szTableName*
SQLSMALLINT	fScope	Input	Not used

Data Type	Argument	In/Out	Description
SQLSMALLINT	fNullable	Input	Not used (Does not allow NULL data because the corresponding columns are returned to the primary keys.)

2.53.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.53.4 Description

When fColType is SQL_BEST_ROWID, SQLSpecialColumns () returns column(s) that uniquely identify each row in the table. These columns can be used in select-list or Where clause.

SQLSpecialColumns () is used to return these columns because SQLColumns () does not return the columns that are automatically updated when columns or rows are updated by the transaction.

If there are no columns that uniquely identifies each row in the table, SQLSpecialColumns () will return the row set without rows. To subsequently call SQLFetch () on the command syntax, return SQL_NO_DATA.

When the fact that the database does not support fColType, fScope, and fNullable arguments is stated, SQLSpecialColumns () will return the empty result set.

2.53.4.1 Columns Returned by SQLSpecialColumns ()

Name	No.	Data Type	Description
SCOPE	1	SMALLINT	SQL_SCOPE_SESSION value is fixed to 2.
COLUMN_NAME	2	VARCHAR (NOT NULL)	Column Name.As for the unnamed string, ODBC returns the empty character string.
DATA_TYPE	3	SMALLINT (NOT NULL)	SQL data type
TYPE_NAME	4	VARCHAR (NOT NULL)	Character string representing the name of the data type corresponding to DATA_TYPE.
COLUMN_SIZE	5	INTEGER	Column Size. NULL will be returned when the string size is not proper.
BUFFER_LENGTH	6	INTEGER	The maximum byte storing the data

Name	No.	Data Type	Description
DECIMAL_DIGITS	7	SMALLINT	The NULL will return the data type that cannot apply the decimal points of the column and the decimal points.
PSEUDO_COLUMN	8	SMALLINT	Maximum digits of the character of binary data-type string. For other data types, NULL will be returned.

2.53.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication line failure	Communication channel failure before the function processing is completed between the ODBC and the database
HY009	Use an invalid pointer (null pointer)	szTableName is a NULL pointer.

2.53.6 Related Functions

```
SQLBindCol

SQLColumns

SQLFetch

SQLPrimaryKeys
```

2.53.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta7.cpp .>

```
if (SQLSpecialColumns(stmt, 0,
                           NULL, 0,
                           NULL, 0,
                           "DEMO_META7", SQL_NTS,
                           NULL, 0) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLColumns");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
 SQLBindCol(stmt, 2, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
 SQLBindCol(stmt, 3, SQL_C_SSHORT, &DataType, 0, &cbDataType);
 SQLBindCol(stmt, 4, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
 SQLBindCol(stmt, 5, SQL_C_SLONG, &ColumnSize, 0, &cbColumnSize);
 SQLBindCol(stmt, 6, SQL_C_SLONG, &BufferLength, 0, &cbBufferLength);
 SQLBindCol(stmt, 7, SQL_C_SSHORT, &DecimalDigits, 0, &cbDecimalDigits);
```

2.54 SQLStatistics

SQLStatistics retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns the information as a result set.

SQLStatisticsW() as a Unicode string supports same execution as SQLStatistics().

2.54.1 Syntax

```
SQLRETURN  SQLStatistics (
    SQLHSTMT        stmt,
    SQLCHAR         *cName,
    SQLSMALLINT     cNameLength,
    SQLCHAR         *sName,
    SQLSMALLINT     sNameLength,
    SQLCHAR         *tName,
    SQLSMALLINT     tNameLength,
    SQLSMALLINT     unique,
    SQLSMALLINT     reserved );
```

2.54.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR *	cName	Input	Catalog Name
SQLSMALLINT	cNameLength	Input	The length, in bytes, of **cName*
SQLCHAR *	sName	Input	Schema Name
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	tName	Input	The table name. Cannot be a NULL pointer.
SQLSMALLINT	tNameLength	Input	The length, in bytes, of **tName*
SQLSMALLINT	unique	Input	index type: SQL_INDEX_UNIQUE or SQL_INDEX_ALL
SQLSMALLINT	reserved	Input	For future use

2.54.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO
```

```
SQL_INVALID_HANDLE

SQL_ERROR
```

2.54.4 Description

returns information about a single table as a standard result set, ordered by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, and ORDINAL_POSITION. The result set combines the statistical information of the table and the data for index.

2.54.4.1 Column returned by SQLStatistics ()

Column Name	No.	Data Type	Description
TABLE_CAT	1	VARCHAR	Null will be always returned.
TABLE_SCHEM	2	VARCHAR	Schema name of the table to which the statistic or index applies.
TABLE_NAME	3	VARCHAR (NOT NULL)	Table name of the table to which the statistic or index applies.
NON_UNIQUE	4	SMALLINT	Indicates whether the index prohibits duplicate values: SQL_TRUE if the index values can be nonunique. SQL_FALSE if the index values must be unique. NULL is returned if TYPE is SQL_TABLE_STAT.
INDEX_QUALIFIER	5	VARCHAR	An empty character string is returned.
INDEX_NAME	6	VARCHAR	Index name; NULL data will be returned when the TYPE is SQL_TABLE_STAT.
TYPE	7	SMALLINT (NOT NULL)	TYPE of the returned data SQL_TABLE_STAT: Indicates whether the statistical data for the table are displayed SQL_INDEX_BTREE: Indicates B-Tree index. SQL_INDEX_HASHED: Indicates the hashed index. SQL_INDEX_OTHER : Indicates other index types. (T-Tree)
ORDINAL_POSITION	8	SMALLINT	Position of the columns in order in the index. (Starting with 1) NULL data will be returned when the TYPE is SQL_TABLE_STAT

Column Name	No.	Data Type	Description
COLUMN_NAME	9	VARCHAR	Column name . If the column is expression (e.g. SALARY + BENEFITS), the expression will be returned. If the expression cannot be determined, the empty character string will be returned. Null data will be returned when the TYPE is SQL_TABLE_STAT.
ASC_OR_DESC	10	CHAR(1)	Column sorting order. A: In ascending orders, D In descending orders, If the database does not support the sorting orders and the TYPE is SQL_TABLE_STAT, NULL data will be returned.
CARDINALITY	11	INTEGER	Table or index order. When the TYPE is SQL_TABLE_STAT, the row number will be returned. If the TYPE is not SQL_TABLE_STAT, the unique number of the index will be returned. If the database does not support the TYPE, NULL data will be returned.
PAGES	12	INTEGER	Page number used to store data in the index or table. If the TYPE is SQL_TABLE_STAT, this column will include the number of pages used to define the table. If the TYPE is not SQL_TABLE_STAT, this column contains the number of pages used to store the index. If the database does not support the TYPE, NULL data returns.
FILTER_CONDITION	13	VARCHAR	In case of the filtered index, this column will have the filtering conditions. 30000; filter" conditions cannot be decided, this column is an empty character string. Null In case the filter has not been faltered. When it is not possible to decide whether the index has been filtered or the TYPE is SQL_TABLE_STAT.

2.54.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	

SQLSTATE	Description	Comments
HY009	Use an invalid pointer (null pointer)	tName is a NULL pointer cName is a NULL pointer sName is a NULL pointer

2.54.6 Related Functions

SQLBindCol

SQLFetch

SQLPrimaryKeys

2.54.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta4.cpp

```
if (SQLStatistics(stmt,NULL, 0,
                       NULL, 0,
                       "DEMO_META4", SQL_NTS,
                       SQL_INDEX_ALL, 0) != SQL_SUCCESS)
 {
    execute_err(dbc, stmt, "SQLStatistics");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
 }
 SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
 SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, STR_LEN,&cbTableName);
 SQLBindCol(stmt, 4, SQL_C_SSHORT, &NonUnique, 0, &cbNonUnique);
 SQLBindCol(stmt, 6, SQL_C_CHAR, szIndexName, STR_LEN, &cbIndexName);
 SQLBindCol(stmt, 8, SQL_C_SSHORT, &OrdinalPosition, 0, &cbOrdinalPosition);
 SQLBindCol(stmt, 9, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
 SQLBindCol(stmt, 10, SQL_C_CHAR, szAscDesc, 2, &cbAscDesc);
```

2.55 SQLTablePrivileges

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 statement.

SQLTablePrivilegesW() as a Unicode string supports same execution as SQLTablePrivileges().

2.55.1 Syntax

```
SQLRETURN  SQLTablePrivileges(
    SQLHSTMT        stmt,
    SQLCHAR         *cName,
    SQLSMALLINT     cNaneLength,
    SQLCHAR         *sName,
    SQLSMALLINT     sNameLength,
    SQLCHAR         *tName,
    SQLSMALLINT     tNameLength);
```

2.55.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle
SQLCHAR*	cName	Input	Catalog Name
SQLSMALLINT	cNameLength	Input	The length, in bytes, of **cName*
SQLCHAR *	sName	Input	Name of the schema to retrieve
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	tName	Input	Table name to retrieve
SQLSMALLINT	tNameLength	Input	The length, in bytes, of **tName*

2.55.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

2.55.4 Description

SQLTablePrivileges () returns the data in the standard result set format sorted by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, PRIVILEGE, and GRANTEE.

2.55.4.1 Columns Returned by SQLTablePrivileges ()

Column Name	No.	Data Type	Description
TABLE_CAT	1	VARCHAR	Always NULL Return
TABLE_SCHEM	2	VARCHAR	Schema name; NULL if not applicable to the database.
TABLE_NAME	3	VARCHAR (NOT NULL)	Table name
GRANTOR	4	VARCHAR	Name of the user who granted the privilege; NULL if not applicable to the database.
GRANTEE	5	VARCHAR(NOT NULL)	Name of the user to whom the privilege was granted.
PRIVILEGE	6	VARCHAR(NOT NULL)	Table privilege. One of the following privileges: Alter: The grantee can change the definition of the table. Delete: The grantee can dele the rows in the table. Index: The grantee can perform index operations (such as create or alter) for the table. INSERT: The grantee can insert new rows to the table. REFERENCES: The grantee can refer to the columns of the table with the limited conditions. SELECT: The grantee can search one or multiple columns in the table. UPDATE: The grantee can modify one or more data for the table.
IS_GRANTABLE	7	VARCHAR	Indicates whether the grantee can give privilege to other users. Yes. No. Or NULL if unknown or not applicable to the database.

2.55.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database

SQLSTATE	Description	Comments
HY009	Invalid Arguments used (null pointer).	Argument cName is a NULL pointer.
HY090	Invalid character string or buffer length	One of the name length Arguments is smaller than 0 or is not equal to SQL_NTS.

2.55.6 Related Functions

```
SQLBindCol

SQLCancel

SQLColumns

SQLFetch

SQLPrimaryKeys

SQLStatistics

SQLTables
```

2.55.7 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_meta10.cpp

```
if (SQLTablePrivileges(stmt,
                            NULL, 0,
                            "SYS", SQL_NTS,
                            "DEMO_META10", SQL_NTS) != SQL_SUCCESS)
 {
    execute_err(dbc, stmt, "SQLTablePrivileges");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
 }
 SQLBindCol(stmt, 2, SQL_C_CHAR, szSchema, NAME_LEN, &cbSchema);
 SQLBindCol(stmt, 3, SQL_C_CHAR, szTableName, NAME_LEN,&cbTableName);
 SQLBindCol(stmt, 4, SQL_C_CHAR, szGrantor, NAME_LEN, &cbGrantor);
 SQLBindCol(stmt, 5, SQL_C_CHAR, szGrantee, NAME_LEN, &cbGrantee);
 SQLBindCol(stmt, 6, SQL_C_CHAR, szPrivilege, NAME_LEN,&cbPrivilege);
 SQLBindCol(stmt, 7, SQL_C_CHAR, szGrantable, 5, &cbGrantable);
```

2.56 SQLTables

SQLTables returns the list of table, catalog, or schema names, and table types, stored in a specific data source. The driver returns the information as a result set.

SQLTablesW() as a Unicode string supports same execution as SQLTables().

2.56.1 Syntax

```
SQLRETURN  SQLTables (
    SQLHSTMT       stmt,
    SQLCHAR        *cName,
    SQLSMALLINT    cNameLength,
    SQLCHAR        *sName,
    SQLSMALLINT    sNameLength,
    SQLCHAR        *tName,
    SQLSMALLINT    tNameLength,
    SQLCHAR        *tableType,
    SQLSMALLINT    tableTypeLength);
```

2.56.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	Statement handle for the searched result
SQLCHAR *	cName	Input	Catalog Name
SQLSMALLINT	cNameLength	Input	The length, in bytes, of **cName*
SQLCHAR *	sName	Input	Schema Name
SQLSMALLINT	sNameLength	Input	The length, in bytes, of **sName*
SQLCHAR *	tName	Input	Table Name
SQLSMALLINT	tNameLength	Input	The length, in bytes, of **tName*
SQLCHAR *	tableType	Input	Table type list to compare
SQLSMALLINT	tableType-Length	Input	The length, in bytes, of **tableType*

2.56.3 Return Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE
```

```
SQL_ERROR
```

2.56.4 Description

SQLTables () displays all table information within the required range. Some users may or may not have the SELECT privilege to any of these tables.

The application must be able to handle a situation where the user selects a table for which SELECT privileges are not granted.

The following special syntaxes are defined for cName, sName, tName, and tableType of SQLTables () to support the catalogs, schema, and the list of table types:

If sName is SQL_ALL_SCHEMAS and cName and tName are empty character strings, the result set will include the schema list valid for the database. (All columns expect that TABLE_SCHEM columns will include NULL data.)

If tableType is SQL_ALL_TABLE_TYPES and cName, sName, and tName are empty character strings, the result set will include the list of table types valid for the database. (All columns except TABLE_TYPE column will include the NULL data.)

An application must specify the tableType in capital letters.

SQLTables () returns the data in the standard result set format sorted by TABLE_TYPE, TABLE_CAT, TABLE_SCHEM, and TABLE_NAME.

2.56.4.1 Columns Returned by SQLTables ()

Name	No.	Data Type	Description
TABLE_CAT	1	VARCHAR	Null will be always returned.
TABLE_SCHEM	2	VARCHAR	Schema name including TABLE_Name; NULL if not applicable to the database
TABLE_NAME	3	VARCHAR (NOT NULL)	Table Name
TABLE_TYPE	4	VARCHAR	Table type name (Only ‘Table’ exists in ALTIBASE HDB.)
REMARKS	5	VARCHAR	For future use.

2.56.5 Diagnosis

SQLSTATE	Description	Comments
08S01	Communication channel error	Communication channel failure before the function processing is completed between the ODBC and the database
HY000	General error	

2.56.6 Related Functions

SQLBindCol

SQLColumns

SQLFetch

SQLStatistics

2.56.7 Example

See: $$ALTIBASE_HOME/sample/SQLCLI/demo_meta1.cpp

```
if (SQLTables(stmt,
                   NULL, 0,
                   NULL, 0,
                   NULL, 0,
                   NULL, 0) != SQL_SUCCESS)
 {
    execute_err(dbc, stmt, "SQLTables");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
 }

 if (SQLBindCol(stmt, 2, SQL_C_CHAR,
                schem, sizeof(schem), &schem_ind) != SQL_SUCCESS)
 {
    execute_err(dbc, stmt, "SQLBindCol");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
 }

 if (SQLBindCol(stmt, 3, SQL_C_CHAR,
                name, sizeof(name), &name_ind) != SQL_SUCCESS)
 {
    execute_err(dbc, stmt, "SQLBindCol");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
 }

 if (SQLBindCol(stmt, 4, SQL_C_CHAR,
                type, sizeof(type), &type_ind) != SQL_SUCCESS)
 {
    execute_err(dbc, stmt, "SQLBindCol");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
 }
while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
 {
 if ( rc == SQL_ERROR )
 {
    execute_err(dbc, stmt, "SQLFetch");
    break;
 }
 printf("%-40s%-40s%s\n", schem, name, type);
 }
```

2.57 SQLTransact

SQLTransact requests a commit or rollback operation for all active operations on all statements associated with a connection.

Normally terminates or cancels all changed in the database made after the connection or before calling of SQLTransact ().

If the transaction is in use in the connection status, an application must call SQLTransact () before disconnecting the database.

SQLTransact () has been replaced by SQLEndTran ().

2.57.1 Syntax

```
SQLRETURN  SQLTransact (
    SQLHENV         env,
    SQLHDBC         dbc,
    SQLSMALLINT     type );
```

2.57.2 Arguments

Data Type	Argument	In/Out	Description
SQLHENV	Env	Input	Environment Handle
SQLHDBC	Dbc	Input	Connection Handle
SQLSMALLINT	type	Input	One of the following two values. SQL_COMMIT, SQL_ROLLBACK

2.57.3 Return Values

```
SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_ERROR
```

2.57.4 Description

Completing the transaction with SQL_COMMIT or SQL_ROLLBACK will have the following effects:

Even after SQLTransact () is called, the statement handle remains valid.

The bound parameter and the column binding remain alive longer than the transaction.

Discarding incompleted result sets..

Calling SQLTransact () when there is not transaction is currently used will return SQL_SUCCESS without any affecting the database server.

SQLTransact() may fail while Commit or Rollback is executed due to loss of connection. In this case, an application cannot judge whether commit or rollback was made.In this case the database administrator handles it manually.

2.57.5 Example

See: $ALTIBASE_HOME/sample/SQLCLI/demo_tran1.cpp

```
SQLTransact(SQL_NULL_HENV, dbc, SQL_COMMIT);
```

3 LOB Interface

This chapter describes functions and data types that can be used for handling LOB data.

3.1 LOB Data Types

The following table shows SQL data type identifiers that support LOB:

SQL TypeIdentifier	Data Type	Description
SQL_BLOB	BLOB	BLOB is a binary data type with a variable length.
SQL_CLOB	CLOB	CLOB is a character data type with a variable length.

The following table shows C data type identifiers that support LOB. It lists C data type of ODBC for each identifier and their definition.

C Type Identifier	ODBC C Type	C Type Definition
SQL_C_BLOB_LOCATOR	SQLUBIGINT	unsigned _int64
SQL_C_CLOB_LOCATOR	SQLUBIGINT	unsigned _int64

The name of a 64-bit integer type may vary depending on platform. The _int64 shown in the above table is the name of a 64-bit integer that is used in several platforms including Windows.

Use SQL_C_CHAR for CLOB data and SQL_C_BINARY for BLOB data to bind user variables.

To obtain a LOB locator, bind SQL_C_CLOB_LOCATOR or SQL_C_BLOB_LOCATOR appropriately based on the LOB column type. A LOB locator in this context, – a LOB location input scheme – is a handle that is used during LOB data operation like a file pointer in an operating system.

The LOB location input scheme for Read can be obtained after SELECT LOB column name FROM table where… and select are executed. The LOB location input scheme for Write can be obtained after SELECT LOB column name FROM table where… FOR UPDATE are executed.

Since a LOB location input scheme refers to LOB data at a certain point in relation to MVCC, it has the same life cycle with the transaction that has created itself. Therefore, to perform LOB operation with a LOB location input scheme, a connection should be always established in Non-Autocommit Mode.

Care must be taken as there is no LOB type of a user variable such as SQL_C_BLOB or SQL_C_CLOB.

3.2 Function Overview

The functions that are available for manipulating LOB data are as follows:

1. SQLBindFileToCol() (Non-standard)

   Full Retrieve

2. SQLBindFileToParam() (Non-standard)

   Full Insert, Full Update

3. SQLGetLobLength() (Non-standard)

   Get the length of LOB data.

4. SQLGetLob() (Non-standard)

   Partial Retrieve

5. SQLPutLob() (Non-standard)

   Partial Insert, Partial Update, Partial Delete

6. SQLFreeLob() (Non-standard)

   Release the LOB locator being used.

7. SQLGetData(), SQLPutData() (Standard)

   Full Retrieve/Update

8. Other ODBC standard functions

Among the above functions, the functions #1 through #6 are special functions that are provided by ALTIBASE HDB for LOB manipulation, and they are not ODBC standard functions. ODBC standard functions such as #7 and #8 can be used to access LOB data whether the database column type is LOB or not. However, when only a standard function is used, the functions for partial update and partial retrieve are not available. If a user wants to do programming with ODBC Driver Manager, he should add the following entry to the odbc.ini file:

```
LongDataCompat = yes
```

If the above entry is added, the types such as SQL_BLOB and SQL_CLOB are converted to SQL_LONGVARBINARY and SQL_LONGVARCHAR types respectively before they are passed to the user. Therefore, if ODBC Driver Manager is used, transparent manipulation of LOB data can be ensured.

3.3 SQLBindFileToCol

SQLBindFileToCol binds a file or files to the columns in the result set for BLOB or CLOB data type.

3.3.1 Syntax

```
SQLRETURN SQLBindFileToCol(
    SQLHSTMT        stmt,
    SQLSMALLINT     col,
    SQLCHAR         *fileName,
    SQLINTEGER      *fileNameLength,
    SQLUINTEGER     *fileOptions,
    SQLINTEGER      fileNameBufferSize,
    SQLLEN          *valueLength);
```

3.3.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	Stmt	Input	An instruction handle for the found results.
SQLSMALLINT	Col	Input	Begins from #1 in the order of columns in the result set to bind.
SQLCHAR *	filename	Input(Pending)	A pointer to the buffer that holds a filename or an array of filenames. It cannot be NULL.Upon SQLFetch(), there should be a filename stored in this buffer, and SQLFetch() returns data to the file(s).Either of an absolute path (recommended) and a relative path is allowed.
SQLINTEGER *	fileName-Length	Input(Pending)	A pointer to the buffer that holds a filename length or an array of filename lengths.Upon SQLFetch(), there should be a filename length stored in this buffer.If this argument is NULL, a filename is regarded as a null-terminated string. That is, it has the same effect as if SQL_NTS were stored in the memory pointed by this argument.The max. length of a filename is 255 characters.

Data Type	Argument	In/Out	Description
SQLINTEGER *	fileOptions	Input(Pending)	A pointer to the buffer that holds a file option or an array of file options. Upon SQLFetch(), there should a file option stored in this buffer. The following options are available: SQL_FILE_CREATE creates one if there is no file, and returns SQL_ERROR if there is a file. SQL_FILE_OVERWRITE creates one if there is no file, and overwrites it if there is a file. SQL_FILE_APPEND creates one if there is no file, and appends to it if there is a file.Only one of the above options can be selected and there is no default option. This argument cannot be NULL.
SQLINTEGER *	fileNameBufferSize	Input	Sets the length of the fileName buffer.
SQLLEN *	valueLength	Output(Pending)	A pointer to the buffer that holds an indicator variable or an array of indicator variables. It cannot be NULL. It is used to return the length of the data stored in a file or to indicate that LOB is NULL. SQLFetch() can return the following values to the buffer pointed by this pointer: 1. Data length, 2. SQL_NULL_DATA.

3.3.3 Result Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

3.3.4 Description

SQLBindFileToCol() binds LOB data in the result set to a file, and SQLBindCol() binds it to an application variable (memory buffer).

If SQLFetch() is called after SQLBindFileToCol() is called, LOB data from DBMS is stored in a file, and the length (byte) of the data stored in the file is stored in the buffer pointed by the valueLength pointer. If LOB is NULL, SQL_NULL_DATA is stored in the buffer pointed by the valueLength pointer. The values of fileName, fileNameLength and fileOptions arguments are referred to upon SQLFetch(), and any error in these arguments is also reported during fetching.

To transfer more than one LOB to a file at once upon fetching, all of the fileName, fileNameLength, fileOptions and valueLength arguments should be arrays.

3.3.5 Diagnosis

SQLSTATE	Description	Note
08S01	A communication line fault (Data transmission failure)	A communication line fails before function processing is complete between ODBC and DB.
HY000	A general error	

3.3.6 Related Functions

SQLBindCol

SQLBindFileToParam

SQLDescribeCol

SQLFetch

3.3.7 Examples

It is assumed that a table has been created with the following DDL.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 BLOB);
```

Write one LOB to a file.

```
SQLCHAR fileName[16];
SQLINTEGER fileNameLength = 15;
SQLUINTEGER fileOptions = SQL_FILE_CREATE;
SQLINTEGER valueLength;
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1");
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPrepare : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Specify a file to put the result values of Select. */
strcpy(fileName, "Terminator2.avi");
if (SQLBindFileToCol(stmt, 1, fileName, &fileNameLength, &fileOptions, 16,
&valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindFileToCol : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
if (SQLFetch(stmt) == SQL_SUCCESS)
{
    printf("SQLFetch success!!!\n");
}
else
{
    execute_err(dbc, stmt, "SQLFetch : ");
}
```

Write three LOB's to a file.

```
SQLCHAR fileName[3][10];
SQLINTEGER fileNameLength[3];
SQLUINTEGER fileOptions[3];
SQLINTEGER valueLength[3];
.
.
.
if (SQLSetStmtAttr(stmt, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER) 3, 0) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLSetStmtAttr : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLSetStmtAttr(stmt, SQL_ATTR_ROW_BIND_TYPE, SQL_BIND_BY_COLUMN, 0) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLSetStmtAttr : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(query, "SELECT i2 FROM T1 WHERE i1 >= 1 AND i1 <= 3");

if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Specify a file to put the result values of Select. */
strcpy(fileName[0], "Cube.avi");
strcpy(fileName[1], "Movie.avi");
strcpy(fileName[2], "Term.avi");

for (i = 0; i < 3; i++)
{
    fileNameLength[i] = strlen(fileName[i]);
    fileOptions[i] = SQL_FILE_CREATE;
}
```

```
if (SQLBindFileToCol(stmt, 1, (SQLCHAR *) fileName, fileNameLength, fileOp-
tions, 10, valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindFileToCol : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) == SQL_SUCCESS)
{
    printf("SQLFetch success!!!\n");
}
else
{
    execute_err(dbc, stmt, "SQLFetch : ");
}
```

Write n LOB's to a file.

```
SQLCHAR fileName[11];
SQLINTEGER fileNameLength = 10;
SQLUINTEGER fileOptions = SQL_FILE_OVERWRITE;
SQLINTEGER valueLength;
.
strcpy(query, "SELECT i2 FROM T1");

if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindFileToCol(stmt, 1, fileName, &fileNameLength, &fileOptions, 11,
&valueLength) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindFileToCol : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

for (i = 0; ; i++)
{
    sprintf(fileName, "Term%02d.avi", i + 1);
    rc = SQLFetch(stmt);
    if (rc == SQL_SUCCESS)
    {
       printf("SQLFetch of file[%02] success!!!\n", i + 1);
    }
    else if (rc == SQL_NO_DATA)
    {
       break;
    }
    else
    {
       execute_err(dbc, stmt, "SQLFetch : ");
       break;
    }
}
```

3.4 SQLBindFileToParam

SQLBindFileToParam binds the parameter market '?' used for LOB data type to a file or files. When SQLExecute() or SQLExecDirect() is called, data is transferred from the file(s) to the database management system.

3.4.1 Syntax

```
SQLRETURN SQLBindFileToParam(
    SQLHSTMT        stmt,
    SQLSMALLINT     par,
    SQLSMALLINT     sqlType,
    SQLCHAR         *fileName,
    SQLINTEGER      *fileNameLength,
    SQLUINTEGER     *fileOptions,
    SQLINTEGER      maxFileNameLength,
    SQLLEN          *ind);
```

3.4.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	A handle for the found results.
SQLSMALLINT	Par	Input	The order of parameters, which begins at 1.
SQLSMALLINT	sqlType	Input	The SQL data type of a parameter.The following options are available: SQL_BLOB SQL_CLOB
SQLCHAR *	fileName	Input(Pending)	A pointer to the buffer that holds a filename or an array of filenames. Upon SQLExecute() or SQLExecDirect(), there should be a filename stored in this buffer. Either of an absolute path (recommended) and a relative path is allowed. This argument cannot be NULL.

Data Type	Argument	In/Out	Description
SQLINTEGER *	fileName-Length	Input(Pending)	A pointer to the buffer that holds a file-name length or an array of filename lengths.Upon SQLExecute() or SQLExec-Direct(), there should be a filename length stored in this buffer.If this argu-ment is NULL, a filename is regarded as a null-terminated string. That is, it has the same effect as if SQL_NTS were stored in the memory pointed by this argu-ment.The max. length of a filename is 255 characters.
SQLINTEGER *	fileOptions	Input(Pending)	A pointer to the buffer that holds a file option or an array of file options. Upon SQLExecute() or SQLExecDirect(), there should a file option stored in this buf-fer.The following option is available: SQL_FILE_READ.
SQLINTEGER *	fileNameBuf-ferSize	Input	The length of the filename buffer.
SQLLEN *	Ind	Input(Pending)	A pointer to the buffer that holds an indi-cator variable or an array of indicator vari-ables. It cannot be NULL.It is used to determine if LOB is NULL.The following values can be set for the buffer pointed by this pointer: 0, SQL_NULL_DATA.

3.4.3 Result Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

3.4.4 Description

SQLBindFileToParam() binds a LOB parameter marker to a file. SQLBindParameter() can be used to bind a parameter marker to an application variable (memory buffer). For SQLBindFileToParam() and SQLBindParameter(), only the binding by the most recently called bind function is valid.

Because the values of fileName, fileNameLength, fileOptions and ind arguments are referred to upon SQLExecute() or SQLExecDirect(), they should be set before SQLExecute() or SQLExecDirect() is called. When SQLExecute() or SQLExecDirect() is called, data is transferred from the file being bound to DBMS.

If LOB is NULL, the buffer pointed by the ind pointer should be set to SQL_NULL_DATA, and then SQLExecute() or SQLExecDirect() should be called. If LOB is not NULL, the buffer pointed by the ind pointer should be set to 0. The ind argument cannot be a NULL pointer.

To bind an array of files to a parameter marker, all of the fileName, fileNameLength, fileOptions and ind arguments should be arrays.

3.4.5 Diagnosis

SQLSTATE	Description	Note
08S01	A communication link fault (Data transmission failure)	A communication link failed before function processing is complete between ODBC and DB.
HY000	A general error	

3.4.6 Related Functions

```
SQLBindCol

SQLBindFileToCol

SQLExecute

SQLExecDirect

SQLDescribeParam
```

3.4.7 Examples

It is assumed that a table has been created with the following DDL.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 BLOB);
```

Input one LOB to a table.

```
SQLCHAR fileName[16];
SQLINTEGER fileNameLength = 15;
SQLUINTEGER fileOptions = SQL_FILE_READ;
SQLINTEGER ind = 0;
.
strcpy(query, "INSERT INTO T1 VALUES (1, ?)");

/* Prepare a statement and bind a file. */
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPrepare : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(fileName, "Terminator2.avi");
if (SQLBindFileToParam(stmt, 1, SQL_BLOB, fileName, &fileNameLength, &fileOp-
tions, 16, &ind) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindFileToParam : “);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

Input three LOB’s to a table.

```
    SQLINTEGER i1[3];
    SQLCHAR fileName[3][10];
    SQLINTEGER fileNameLength[3];
    SQLUINTEGER fileOptions[3];
    SQLINTEGER ind[3];
    .
    if (SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER) 3, 0) !=
    SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLSetStmtAttr : ");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLSetStmtAttr(stmt, SQL_ATTR_PARAM_BIND_TYPE,
    SQL_PARAMETER_BIND_BY_COLUMN, 0) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLSetStmtAttr : ");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    strcpy(query, "INSERT INTO T1 VALUES (?, ?)");

    /* Prepare a statement and bind a file. */
    if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, “SQLPrepare : ”);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_INTEGER, SQL_INTEGER, 0,
    0, (SQLPOINTER) i1, 0, NULL) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, “SQLBindParameter : “);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
```

```
if (SQLBindFileToParam(stmt, 2, SQL_BLOB, (SQLCHAR *) fileName, fileName-
Length, fileOptions, 10, ind) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindFileToParam : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Specify a file to insert data. */
strcpy(fileName[0], "Cube.avi");
strcpy(fileName[1], "Movie.avi");
strcpy(fileName[2], "Term.avi");

for (i = 0; i < 3; i++)
{
    i1[i] = i + 1;
    fileNameLength[i] = strlen(fileName[i]);
    fileOptions[i] = SQL_FILE_READ;
    ind[i] = 0;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

Update one LOB in a table.

```
SQLCHAR fileName[16];
SQLINTEGER fileNameLength = 15;
SQLUINTEGER fileOptions = SQL_FILE_READ;
SQLINTEGER ind = 0;
.
strcpy(query, "UPDATE T1 SET i2=? WHERE i1=1");

/* Prepare a statement and bind a file.
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPrepare : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

strcpy(fileName, "Terminator2_fix.avi");
if (SQLBindFileToParam(stmt, 1, SQL_BLOB, fileName, &fileNameLength, &fileOp-
tions, 16, &ind) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindFileToParam : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

3.5 SQLGetLobLength

SQLGetLobLength gets the length of the LOB pointed by the LOB locator obtained during the current transaction.

3.5.1 Syntax

```
SQLRETURN SQLGetLobLength(
    SQLHSTMT       stmt,
    SQLUBIGINT     locator,
    SQLSMALLINT    locatorCType,
    SQLLEN *       valueLength);
```

3.5.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	Stmt	Input	A handle for the found results.
SQLUBIGINT	locator	Input	A LOB locator.
SQLSMALLINT	locatorCType	Input	The C data type of A LOB locator. It can have the following values: SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLLEN *	valueLength	Output	Used to store the LOB length or to indicate that LOB is NULL. The buffer pointed by the pointer returns the following values: Data length SQL_NULL_DATA

3.5.3 Result Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

3.5.4 Description

A function that is used to get the length of the LOB pointed by a LOB locator.

A LOB locator has the value that directly points LOB in a database (not offset in LOB). A LOB locator can be obtained in two ways:

It can be obtained from the LOB column in the result set of the SELECT SQL statement with SQLBind-Col() or SQLGetData() function.

In this case, the application buffer type bound by the user should be SQL_C_CLOB_LOCATOR or SQL_C_BLOB_LOCATOR.

It can be obtained from the output parameter of SQLBindParameter().

In this case, the application buffer type bound by the user should be SQL_C_CLOB_LOCATOR or SQL_C_BLOB_LOCATOR.

If a LOB locator has not been obtained during the current transaction, it cannot be used as an argument for this function. This is because a LOB locator becomes invalid if a transaction is terminated. If an invalid LOB locator is used as an argument, this function will return SQL_ERROR, and the buffer pointed by the valueLength argument will not be changed.

The length of LOB is returned via the valueLength argument. However, If a LOB locator points NULL LOB, SQL_NULL_DATA is returned to the buffer pointed by the valueLength argument. If a LOB locator points NULL LOB, this function will not return an error.

3.5.5 Diagnosis

SQLSTATE	Description	Note
08S01	A communication link fault (Data transmission failure)	A communication link failed before function processing is complete between ODBC and DB.
HY000	A general error	

3.5.6 Related Functions

```
SQLBindCol

SQLBindParameter

SQLFetch

SQLExecute

SQLGetLob

SQLPutLob
```

3.5.7 Example

It is assumed that a table has been created with the following DDL.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 BLOB);
```

Get the length of LOB data.

```
DQLINTEGER valueLength;
SQLUBIGINT lobLoc;
.
.
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecDirect : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_BLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindCol : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFetch : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLGetLobLength(stmt, lobLoc, SQL_C_BLOB_LOCATOR, &valueLength) != 
SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLGetLobLength : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

printf("SQLGetLobLength success!!!\n");

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFreeLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

3.6 SQLGetLob

SQLGetLob gets a part of data in the LOB pointed by the LOB locator obtained during the current transaction to an application data buffer.

3.6.1 Syntax

```
SQLRETURN SQLGetLob(
    SQLHSTMT        stmt,
    SQLSMALLINT     locatorCType,
    SQLUBIGINT      sourceLocator,
    SQLLEN          fromPosition,
    SQLLEN          forLength,
    SQLSMALLINT     targetCType,
    SQLPOINTER      value,
    SQLLEN          bufferSize,
    SQLLEN *        valueLength);
```

3.6.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	Stmt	Input	A handle for the found results.
SQLSMALLINT	locatorCType	Input	The C data type identifier of a LOB locator. It can have the following values: SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLUBIGINT	sourceLocator	Input	A source LOB locator.
SQLLEN	fromPosition	Input	The start point of data to transfer from LOB (byte). It begins at 1.
SQLLEN	forLength	Input	The length of data to transfer from LOB (byte).
SQLSMALLINT	targetCType	Input	The C data type identifier of the value buffer. It can have the following values: SQL_C_BINARY SQL_C_CHAR If the user reads BLOB data into the SQL_C_CHAR buffer, BINARY is converted to CHAR, and the result value is stored in an application buffer.
SQLPOINTER	Value	Output	A pointer to the buffer that holds data.
SQLLEN	bufferSize	Input	The size of the value buffer (byte).

Data Type	Argument	In/Out	Description
SQLLEN *	valueLength	Output	A pointer to the buffer to which the length of data stored in the value buffer is returned. This argument cannot be NULL.

3.6.3 Result Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

3.6.4 Description

Gets a part of data in the LOB pointed by the source locator to an application data buffer. It is used to get LOB data in parts. The total length of LOB can be obtained by calling SQLGetLobLength().

If a source locator is not the LOB locator obtained during the current transaction, it cannot be used as an argument for this function. This is because a LOB locator becomes invalid if a transaction is terminated. If a source LOB locator is not valid, the SQLGetLob() function will return SQL_ERROR, and the buffer pointed by the value and valueLength arguments will not be changed.

If a source locator points NULL LOB, the SQLGetLob() function works in the same way as when a LOB locator points LOB with length 0.

If the size of data that will be returned by calling SQLGetLob() exceeds the size of bufferSize, the SQLGetLob() will return SQL_SUCCESS_WITH_INFO (SQLSTATE=01004), and the data will be truncated to fit the buffer size before it is returned to the buffer.

3.6.5 Diagnosis

SQLSTATE	Description	Remark
08S01	A communication link fault (Data transmission failure)	A communication link failed before function processing is complete between ODBC and DB.
HY000	A general error	

3.6.6 Related Functions

```
SQLGetLobLength
```

```
SQLPutLob
```

3.6.7 Example

It is assumed that a table has been created with the following DDL.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 CLOB);
```

Get LOB data to an application buffer by using the SQLGetLob() function.

```
SQLCHAR buf[1024];
SQLINTEGER valueLength, accumLength, forLength, procLength;
SQLUBIGINT lobLoc;
.
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_CLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindCol : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFetch : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLGetLobLength(stmt, lobLoc, SQL_C_CLOB_LOCATOR, &valueLength) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLGetLobLength : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

for (accumLength = 0; accumLength < valueLength; accumLength += procLength)
{
    if (valueLength - accumLength > 256)
    {
       forLength = 256;
    }
    else
    {
       forLength = valueLength - accumLength;
    }
    if (SQLGetLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, accumLength, forLength,
SQL_C_CHAR, buf, 256, &procLength) != SQL_SUCCESS)
    {
       execute_err(dbc, stmt, "SQLGetLob : ");
       SQLFreeStmt(stmt, SQL_DROP);
```

```
            return SQL_ERROR;
        }
        printf("%s", buf);
    }

    if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLFreeLob : ");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
```

3.7 SQLPutLob

SQLPutLob insert, update or delete data in an application data buffer to the current LOB pointed by a LOB locator. Each operation is handled as a special case of the update operation. Inserting is to update LOB data with length 0 with new data. Deleting is to update the LOB data for length n with 0.

3.7.1 Syntax

```
SQLRETURN SQLPutLob(
    SQLHSTMT        stmt,
    SQLSMALLINT     locatorCType,
    SQLUBIGINT      targetLocator,
    SQLLEN          fromPosition,
    SQLLEN          forLength,
    SQLSMALLINT     sourceCType,
    SQLPOINTER      value,
    SQLLEN          valueLength);
```

3.7.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	A handle for the found results.
SQLSMALLINT	locatorCType	Input	The C data type of a target LOB locator. SQL_C_BLOB_LOCATOR SQL_C_CLOB_LOCATOR
SQLUBIGINT	targetLocator	Input	A target LOB locator.
SQLLEN	fromPosition	Input	The start point to update data in LOB (byte). It begins at 1.
SQLLEN	forLength	Input	The length of a segment in LOB to be updated (byte).
SQLSMALLINT	sourceCType	Input	The C data type identifier of the value buffer. SQL_C_BINARY (for BLOB), SQL_C_CHAR (for CLOB)
SQLPOINTER	value	Input	A pointer to the buffer that holds data.
SQLLEN	valueLength	Input	The length of a buffer that holds data (byte). It should be set with 0 or a larger value. It cannot be set with SQL_NULL_DATA. If the value is 0, the data for forLength from fromPosition in a target LOB is deleted (0 does not mean SQL_NTS.)

3.7.3 Result Values

```
SQL_SUCCESS

SQL_SUCCESS_WITH_INFO

SQL_INVALID_HANDLE

SQL_ERROR
```

3.7.4 Description

Inserts or updates data stored in an application data buffer to the LOB pointed by a target LOB locator. It can also delete a part or all of the data in the LOB pointed a target LOB locator.

It replaces the data for forLength from fromPosition with the data for valueLength in the value buffer. If forLength > valueLength, the length of target LOB decreases, and if forLength < valueLength, the length increases.

If forLength = 0 and valueLength > 0, it inserts data in the value buffer to the fromPosition position of target LOB. When this happens, the data after the fromPosition position in the original target LOB is shifted backward for valueLength. If forLength > 0 and valueLength = 0, it deletes the data for forLength from the fromPosition position of target LOB. When this happens, the data after the fromPosition + forLength position in the original target LOB is shifted forward for forLength.

If a target locator is not the LOB locator opened during the current transaction, it cannot be used as an argument for this function. This is because the LOB locator becomes invalid if a transaction is terminated. If a target LOB locator is not valid, the SQLPutLob() function will return SQL_ERROR.

If a target locator points NULL LOB, the SQLPutLob() function works in the same way as when a LOB locator points LOB with length 0.

The fromPosition argument should not exceed target LOB at the time of calling. If it is larger than target LOB, the SQLPutLob() function will return SQL_ERROR.

3.7.5 Diagnosis

SQLSTATE	Description	Note
08S01	A communication link fault (Data transmission failure)	A communication link failed before function processing is complete between ODBC and DB.
HY000	A general error	

3.7.6 Related Functions

```
SQLGetLobLength
```

SQLGetLob

3.7.7 Examples

It is assumed that a table has been created with the following DDL.

```
CREATE TABLE T1 (i1 INTEGER PRIMARY KEY, i2 CLOB);
```

After inserting a record with the CLOB column value being 'Ver.Beta', replace 'Beta' with 'Gamma'.

```
SQLCHAR buf[5];
SQLUBIGINT lobLoc;
.
strcpy(query, "INSERT INTO T1 VALUES (1, 'Ver.Beta')");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1 FOR UPDATE");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecDirect : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_CLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindCol : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFetch : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, "Gamma", 5);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 4, 4, SQL_C_CHAR, buf, 5) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPutLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFreeLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

After inserting a record with the CLOB column value being ‘ss’, insert ‘mile’ between two s’s.

```
SQLCHAR buf[4];
SQLUBIGINT lobLoc;
.
strcpy(query, "INSERT INTO T1 VALUES (2, 'ss')");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecDirect : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
.
strcpy(query, "SELECT i2 FROM T1 WHERE i1=1 FOR UPDATE");
if (SQLExecDirect(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLExecDirect : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindCol(stmt, 1, SQL_C_CLOB_LOCATOR, &lobLoc, 0, NULL) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLBindCol : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFetch(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLFetch : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, "mile", 4);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 1, 0, SQL_C_CHAR, buf, 4) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPutLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Delete ‘mil’ from 'smiles’. */
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 1, 3, SQL_C_CHAR, NULL, 0) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPutLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* Append ‘4’ to ‘ses’. */
memcpy(buf, "4", 1);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 3, 0, SQL_C_CHAR, buf, 1) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, “SQLPutLob : ”);
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
```

```
{
    execute_err(dbc, stmt, "SQLFreeLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

Insert a record with the CLOB column value being 'Ver.0.9a'.

```
SQLCHAR buf[8];
SQLINTEGER lobInd;
SQLUBIGINT lobLoc;
.
.
.
strcpy(query, "INSERT INTO T1 VALUES (5, ?)");
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPrepare : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_CLOB_LOCATOR,
SQL_CLOB_LOCATOR, 0, 0, &lobLoc, 0, &lobInd) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindParameter : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, "Ver.0.9a", 8);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 0, 0, SQL_C_CHAR, buf, 7) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPutLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}



/* In 'Ver.0.9a', replace '0.9' with '1'. */
memcpy(buf, "1", 1);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 4, 3, SQL_C_CHAR, buf, 1) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPutLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFreeLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
```

```
        return SQL_ERROR;
    }
```

Change the CLOB of multiple records to 'Retail' at once.

```
SQLCHAR buf[6];
SQLINTEGER lobInd;
SQLUBIGINT lobLoc;
.
.
.
strcpy(query, "UPDATE T1 SET i2=? WHERE i1>=1 AND i1<=100");
if (SQLPrepare(stmt, query, SQL_NTS) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPrepare : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

/* If an UPDATE query is executed after a LOB locator parameter is being out-
bound, LOB columns to be updated will be truncated to null automatically. */

if (SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_CLOB_LOCATOR,
SQL_CLOB_LOCATOR, 0, 0, &lobLoc, 0, &lobInd) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLBindParameter : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLExecute(stmt) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLExecute : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

memcpy(buf, "Retail", 6);
if (SQLPutLob(stmt, SQL_C_CLOB_LOCATOR, lobLoc, 0, 0, SQL_C_CHAR, buf, 6) !=
SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLPutLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}

if (SQLFreeLob(stmt, lobLoc) != SQL_SUCCESS)
{
    execute_err(dbc, stmt, "SQLFreeLob : ");
    SQLFreeStmt(stmt, SQL_DROP);
    return SQL_ERROR;
}
```

3.8 SQLFreeLob

SQLFreeLob releases resources that are related to a LOB locator opened during the current transaction.

3.8.1 Syntax

```
SQLRETURN SQLFreeLob (
 SQLHSTMT stmt,
 SQLUBIGINT locator);
```

3.8.2 Arguments

Data Type	Argument	In/Out	Description
SQLHSTMT	stmt	Input	A handle for the found results.
SQLUBIGINT	locator	Input	A LOB locator.

3.8.3 Result Values

```
SQL_SUCCESS

SQL_INVALID_HANDLE

SQL_ERROR
```

3.8.4 Description

Reports that operation of LOB pointed by a LOB locator is complete. This will release the LOB locator assigned by a server and other related resources in the server.

This function does not commit or rollback changes to LOB pointed by a LOB locator.

If a transaction is terminated with SQLEndTran(), a LOB locator is automatically released and this function does have to be called.

3.8.5 Diagnosis

SQLSTATE	Description	Note
08S01	A communication link fault (Data transmission failure)	A communication link failed before function processing is complete between ODBC and DB.
HY000	A general error	

3.8.6 Related Functions

```
SQLGetLobLength

SQLGetLob

SQLPutLob
```

3.8.7 Example

Please see the examples of SQLGetLobLength(), SQLGetLob() and SQLPutLob().

Appendix A. Sample Codes

The appendix contains the entire sample codes which used this document

Programing Considerations

The following describes notes on programming the client using the ODBC and frequent errors:

Multithreading

On multi-threaded operating systems, multiple environment handle (SQLHENV) can be allocated by one application. In case of a multi-threaded application, one environment handle and one connection handle must be allocated to each thread. In the multi-threaded program, the same connection handle can be used by multiple threads at the same time.

When develop multi-threaded applications, ideAllocErrorSpace () function that allocates space for each thread to store the database-related errors must be called during creation of the thread.

Statement Handles

The statement handles (SQLHSTMT) can be allocated by one connection handle (SQLHDBC.) up to 1024

Binding Parameters

Note on using the last argument, valueLength (indicator), upon calling SQLBindCol () and SQLBindParameter ().

In case the value fetched by the output argument in SQLBindCol () is NULL, SQL_NULL_DATA will be returned to the indicator argument.

The valueLength of SQLBindParameter () is used to indicate the length of the buffer when the data type of the argument bound to the input buffer is the variable type. (When the parameter inout type is SQL_PARAM_INOUT)

For example,

Indicator value is SQL_NTS: A string in which the buffer ends with NULL ('')

SQL_NULL_DATA Binding the NULL value

If inout type is SQL_PARAM_OUTPUT in SQLBindParameter (), the indicator functions same as SQLBindCol () indicator in SELECT statement.

Transaction Commit Mode

In case the program is terminated without being committed in the autocommit off session, all execution statement not committed will be all rolled back. However, if the program is terminated after SQLDisconnect () is called, the program will be committed.

Using SQLFreeStmt() function

If the second argument is SQL_DROP in SQLFreeStmt (), the status of the handle will be changed into the previous status before the handle was allocated. however, when SQL_CLOSE argument is used, the handle status after SQLAllocStmt () is executed will be selected and can be used for other queries. In case the command executes SQLPrepare (), the status will be changed into the preparation status after SQLFreeStmt () is called by SQL_CLOSE.

In case SELECT statement is executed by SQLPrepare (). Execution result of SELECT statement upon changing from SQLExecute () to SQLFetch (). When SQLExecute () is called again by binding other hosts variables without fetching untill the last record of the result set, "invalid cursor state" may occur. To prevent this, the user must call SQLFreeStmt (.. , SQL_CLOSE) and SQLExecute ().However, if SQLFetch () was executed until the last record in the execution result of SELECT statement, SQLFreeStmt () of SQL_CLOSE does not need to be called for normal operation.

Sample of Simple Basic Program

```
/***********************************************
** File name = demo_ex1.cpp
***********************************************/
#include <sqlcli.h>
#include <stdio.h>
#include <stdlib.h>
#include <a.h>


#define SQL_LEN 1000
#define MSG_LEN 1024

SQLHENV  env;  // Environment Handle
SQLHDBC  dbc;  // Connection Handle
int      conn_flag;

SQLRETURN alloc_handle();
SQLRETURN db_connect();
void free_handle();

SQLRETURN execute_select();
void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q);

int main()
{
    SQLRETURN    rc;

    env = SQL_NULL_HENV;
    dbc = SQL_NULL_HDBC;
    conn_flag = 0;

    /* allocate handle */
    rc = alloc_handle();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* Connect to Altibase Server */
    rc = db_connect();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    rc = execute_select();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    free_handle();
}


static void print_diagnostic(SQLSMALLINT aHandleType, SQLHANDLE aHandle)
{
    SQLRETURN   rc;
    SQLSMALLINT sRecordNo;
    SQLCHAR     sSQLSTATE[6];
    SQLCHAR     sMessage[2048];
    SQLSMALLINT sMessageLength;
    SQLINTEGER  sNativeError;

    sRecordNo = 1;

    while ((rc = SQLGetDiagRec(aHandleType,
                               aHandle,
                               sRecordNo,
                               sSQLSTATE,
                               &sNativeError,
                               sMessage,
                               sizeof(sMessage),
                               &sMessageLength)) != SQL_NO_DATA)
    {
        printf("Diagnostic Record %d\n", sRecordNo);
        printf("     SQLSTATE     : %s\n", sSQLSTATE);
        printf("     Message text : %s\n", sMessage);
        printf("     Message len  : %d\n", sMessageLength);
        printf("     Native error : 0x%X\n", sNativeError);

        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        {
            break;
        }

        sRecordNo++;
    }
}

void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q)
{
    printf("Error : %s\n",q);

    if (aStmt == SQL_NULL_HSTMT)
```

```
        {
            if (aCon != SQL_NULL_HDBC)
            {
                print_diagnostic(SQL_HANDLE_DBC, aCon);
            }
        }
        else
        {
            print_diagnostic(SQL_HANDLE_STMT, aStmt);
        }
    }

    SQLRETURN alloc_handle()
    {
        /* allocate Environment handle */
        if (SQLAllocEnv(&env) != SQL_SUCCESS)
        {
            printf("SQLAllocEnv error!!\n");
            return SQL_ERROR;
        }

        /* allocate Connection handle */
        if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
        {
            printf("SQLAllocConnect error!!\n");
            return SQL_ERROR;
        }
        return SQL_SUCCESS;
    }

    void free_handle()
    {
        if ( conn_flag == 1 )
        {
            /* close connection */
            SQLDisconnect( dbc );
        }
        /* free connection handle */
        if ( dbc != NULL )
        {
            SQLFreeConnect( dbc );
        }
        if ( env != NULL )
        {
            SQLFreeEnv( env );
        }
    }

    SQLRETURN db_connect()
    {
        char    *USERNAME = "SYS";        // user name
        char    *PASSWD   = "MANAGER";    // user password
        char    *NLS      = "US7ASCII";   // NLS_USE ( KO16KSC5601, US7ASCII )
        char     connStr[1024];

        sprintf(connStr,
                "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s", /*
    ;PORT_NO=20300", */
                USERNAME, PASSWD, 1, NLS);

        /* establish connection */
        if (SQLDriverConnect( dbc, NULL, (SQLCHAR *)connStr, SQL_NTS,
                              NULL, 0, NULL,
                              SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
        {
```

```
        execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
        return SQL_ERROR;
    }

    conn_flag = 1;

    return SQL_SUCCESS;
}

SQLRETURN execute_select()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    SQLRETURN    rc;
    int          i;
    char         query[SQL_LEN];

    SQLSMALLINT  columnCount;
    char         columnName[50];
    SQLSMALLINT  columnNameLength;
    SQLSMALLINT  dataType;
    SQLSMALLINT  scale;
    SQLSMALLINT  nullable;
    SQLULEN      columnSize;

    void       **columnPtr;
    SQLLEN      *columnInd;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
    {
        printf("SQLAllocStmt error!!\n");
        return SQL_ERROR;
    }

    sprintf(query,"SELECT * FROM DEMO_EX1");
    if (SQLExecDirect(stmt, (SQLCHAR *)query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    SQLNumResultCols(stmt, &columnCount);
    columnPtr = (void**) malloc( sizeof(void*) * columnCount );
    columnInd = (SQLLEN*) malloc( sizeof(SQLLEN) * columnCount );
    if ( columnPtr == NULL )
    {
        return SQL_ERROR;
    }

    for ( i=0; i<columnCount; i++ )
    {
        SQLDescribeCol(stmt, i+1,
                      (SQLCHAR *)columnName, sizeof(columnName), &columnName-
Length,
                       &dataType, &columnSize, &scale, &nullable);
        printf("columnName = %s, nullable = %d\n", columnName, nullable);
        switch (dataType)
        {
        case SQL_CHAR:
            printf("%s : CHAR(%d)\n", columnName, columnSize);
            columnPtr[i] = (char*) malloc( columnSize + 1 );
            SQLBindCol(stmt, i+1, SQL_C_CHAR, columnPtr[i], columnSize+1, 
&columnInd[i]);
            break;
        case SQL_VARCHAR:
            printf("%s : VARCHAR(%d)\n", columnName, columnSize);
            columnPtr[i] = (char*) malloc( columnSize + 1 );
            SQLBindCol(stmt, i+1, SQL_C_CHAR, columnPtr[i], columnSize+1,
&columnInd[i]);
            break;
        case SQL_INTEGER:
            printf("%s : INTEGER\n", columnName);
            columnPtr[i] = (int*) malloc( sizeof(int) );
            SQLBindCol(stmt, i+1, SQL_C_SLONG, columnPtr[i], 0, &column-
Ind[i]);
            break;
        case SQL_SMALLINT:
            printf("%s : SMALLINT\n", columnName);
            columnPtr[i] = (short*) malloc( sizeof(short) );
            SQLBindCol(stmt, i+1, SQL_C_SSHORT, columnPtr[i], 0, &column-
Ind[i]);
            break;
        case SQL_NUMERIC:
            printf("%s : NUMERIC(%d,%d)\n", columnName, columnSize, scale);
            columnPtr[i] = (double*) malloc( sizeof(double) );
            SQLBindCol(stmt, i+1, SQL_C_DOUBLE, columnPtr[i], 0, &column-
Ind[i]);
            break;
        case SQL_TYPE_TIMESTAMP:
            printf("%s : DATE\n", columnName);
            columnPtr[i] = (SQL_TIMESTAMP_STRUCT*) malloc(
sizeof(SQL_TIMESTAMP_STRUCT) );
            SQLBindCol(stmt, i+1, SQL_C_TYPE_TIMESTAMP, columnPtr[i], 0,
&columnInd[i]);
            break;
        }
    }

    /* fetches next rowset of data from the result set and print to stdout */


printf("============================================================
======\n");
    while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
    {
        if ( rc != SQL_SUCCESS )
        {
            execute_err(dbc, stmt, query);
            break;
        }
        for ( i=0; i<columnCount; i++ )
        {
            SQLDescribeCol(stmt, i+1,
                           NULL, 0, NULL,
                           &dataType, NULL, NULL, NULL);
            if ( columnInd[i] == SQL_NULL_DATA )
            {
                printf("NULL\t");
                continue;
            }
            switch (dataType)
            {
            case SQL_CHAR:
            case SQL_VARCHAR:
                printf("%s\t", columnPtr[i]);
                break;
            case SQL_INTEGER:
                printf("%d\t", *(int*)columnPtr[i]);
                break;
            case SQL_SMALLINT:
                printf("%d\t", *(short*)columnPtr[i]);
                break;
            case SQL_NUMERIC:
                printf("%10.3f\t", *(double*)columnPtr[i]);
                break;
            case SQL_TYPE_TIMESTAMP:
                printf("%4d/%02d/%02d %02d:%02d:%02d\t",
                        ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->year,
                        ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->month,
                        ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->day,
                        ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->hour,
                        ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->minute,
                        ((SQL_TIMESTAMP_STRUCT*)columnPtr[i])->second);
                break;
            }
        }
        printf("\n");
    }

    SQLFreeStmt(stmt, SQL_DROP);

    for ( i=0; i<columnCount; i++ )
    {
        free( columnPtr[i] );
    }
    free( columnPtr );
    free( columnInd );

    return SQL_SUCCESS;
}
```

Sample of Using Metadata

```
/***********************************************
** File name = demo_meta1.cpp
** Meta data search program example
************************************************/
#include <sqlcli.h>
#include <stdio.h>
#include <stdlib.h>

#define SQL_LEN 1000
#define MSG_LEN 1024

SQLHENV  env;  // Environment Handle
SQLHDBC  dbc;  // Connection Handle
int      conn_flag;

SQLRETURN alloc_handle();
SQLRETURN db_connect();
void free_handle();

SQLRETURN get_tables();
void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q);

int main()
{
    SQLRETURN    rc;

    env = SQL_NULL_HENV;
    dbc = SQL_NULL_HDBC;
    conn_flag = 0;

    /* allocate handle */
    rc = alloc_handle();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* Connect to Altibase Server */
    rc = db_connect();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    rc = get_tables();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    free_handle();
}


static void print_diagnostic(SQLSMALLINT aHandleType, SQLHANDLE aHandle)
{
    SQLRETURN   rc;
    SQLSMALLINT sRecordNo;
    SQLCHAR     sSQLSTATE[6];
    SQLCHAR     sMessage[2048];
    SQLSMALLINT sMessageLength;
    SQLINTEGER  sNativeError;

    sRecordNo = 1;

    while ((rc = SQLGetDiagRec(aHandleType,
                               aHandle,
                               sRecordNo,
                               sSQLSTATE,
                               &sNativeError,
                               sMessage,
                               sizeof(sMessage),
                               &sMessageLength)) != SQL_NO_DATA)
    {
        printf("Diagnostic Record %d\n", sRecordNo);
        printf("     SQLSTATE     : %s\n", sSQLSTATE);
        printf("     Message text : %s\n", sMessage);
        printf("     Message len  : %d\n", sMessageLength);
        printf("     Native error : 0x%X\n", sNativeError);

        if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
        {
            break;
        }

        sRecordNo++;
    }
}
```

```
void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q)
{
    printf("Error : %s\n",q);

    if (aStmt == SQL_NULL_HSTMT)
    {
        if (aCon != SQL_NULL_HDBC)
        {
            print_diagnostic(SQL_HANDLE_DBC, aCon);
        }
    }
    else
    {
        print_diagnostic(SQL_HANDLE_STMT, aStmt);
    }
}


SQLRETURN alloc_handle()
{
    /* allocate Environment handle */
    if (SQLAllocEnv(&env) != SQL_SUCCESS)
    {
        printf("SQLAllocEnv error!!\n");
        return SQL_ERROR;
    }

    /* allocate Connection handle */
    if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
    {
        printf("SQLAllocConnect error!!\n");
        return SQL_ERROR;
    }
    return SQL_SUCCESS;
}

void free_handle()
{
    if ( conn_flag == 1 )
    {
        /* close connection */
        SQLDisconnect( dbc );
    }
    /* free connection handle */
    if ( dbc != NULL )
    {
        SQLFreeConnect( dbc );
    }
    if ( env != NULL )
    {
        SQLFreeEnv( env );
    }
}

SQLRETURN db_connect()
{
    char    *USERNAME = "SYS";        // user name
    char    *PASSWD   = "MANAGER";    // user password
    char    *NLS      = "US7ASCII";   // NLS_USE ( KO16KSC5601, US7ASCII )
    char     connStr[1024];

    sprintf(connStr,
            "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s", /*
;PORT_NO=20300", */
            USERNAME, PASSWD, 1, NLS);
    /* establish connection */
    if (SQLDriverConnect( dbc, NULL, (SQLCHAR *) connStr, SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
    {
        execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
        return SQL_ERROR;
    }

    conn_flag = 1;

    return SQL_SUCCESS;
}

SQLRETURN get_tables()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    SQLRETURN    rc;

    char         schem[50+1] = {0};
    char         name[50+1] = {0};
    char         type[50+1] = {0};
    SQLLEN       schem_ind;
    SQLLEN       name_ind;
    SQLLEN       type_ind;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt))
    {
        printf("SQLAllocHandle error!!\n");
        return SQL_ERROR;
    }

    if (SQLTables(stmt,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0,
                  NULL, 0) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLTables");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 2, SQL_C_CHAR,
                   schem, sizeof(schem), &schem_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 3, SQL_C_CHAR,
                   name, sizeof(name), &name_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindCol(stmt, 4, SQL_C_CHAR,
                   type, sizeof(type), &type_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, "SQLBindCol");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    /* fetches the next rowset of data from the result set and print to stdout
*/
    printf("TABLE_SCHEM\t\tTABLE_NAME\t\tTABLE_TYPE\n");

printf("==============================================================\n");
    while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA)
    {
        if ( rc == SQL_ERROR )
        {
            execute_err(dbc, stmt, "SQLFetch");
            break;
        }
        printf("%-40s%-40s%s\n", schem, name, type);
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);/* == SQLFreeStmt(stmt, SQL_DROP);
*/

    return SQL_SUCCESS;
}
```

Example of Procedure test Program

```
/***********************************************
** File name = demo_ex6.cpp
** Example of procedure test program
***********************************************/
#include <sqlcli.h>
#include <stdio.h>
#include <stdlib.h>


#define SQL_LEN 1000
#define MSG_LEN 1024

SQLHENV  env;  // Environment Handle
SQLHDBC  dbc;  // Connection Handle
int      conn_flag;

SQLRETURN alloc_handle();
SQLRETURN db_connect();
void free_handle();

SQLRETURN execute_proc();
SQLRETURN execute_select();

void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q);

int main()
{
    SQLRETURN    rc;

    env = SQL_NULL_HENV;
    dbc = SQL_NULL_HDBC;
    conn_flag = 0;

    /* allocate handle */
    rc = alloc_handle();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* Connect to Altibase Server */
    rc = db_connect();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* select data */
    rc = execute_select();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* procedure ?? */
    rc = execute_proc();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* select data */
    rc = execute_select();
    if ( rc != SQL_SUCCESS )
    {
        free_handle();
        exit(1);
    }

    /* disconnect, free handles */
    free_handle();
}

void execute_err(SQLHDBC aCon, SQLHSTMT aStmt, char* q)
{
    SQLINTEGER errNo;
    SQLSMALLINT msgLength;
    SQLCHAR errMsg[MSG_LEN];

    printf("Error : %s\n",q);
    if (SQLError ( SQL_NULL_HENV, aCon, aStmt,
                   NULL, &errNo,
                   errMsg, MSG_LEN, &msgLength ) == SQL_SUCCESS)
    {
        printf(" Error : # %ld, %s\n", errNo, errMsg);
    }
}

SQLRETURN alloc_handle()
{
    /* allocate Environment handle */
    if (SQLAllocEnv(&env) != SQL_SUCCESS)
    {
        printf("SQLAllocEnv error!!\n");
        return SQL_ERROR;
    }

    /* allocate Connection handle */
    if (SQLAllocConnect(env, &dbc) != SQL_SUCCESS)
    {
        printf("SQLAllocConnect error!!\n");

        return SQL_ERROR;
    }
    return SQL_SUCCESS;
}

void free_handle()
{
    if ( conn_flag == 1 )
    {
        /* close connection */
        SQLDisconnect( dbc );
    }
    /* free connection handle */
    if ( dbc != NULL )
    {
        SQLFreeConnect( dbc );
    }
    if ( env != NULL )
    {
        SQLFreeEnv( env );
    }
}

SQLRETURN db_connect()
{
    char    *USERNAME = "SYS";        // user name
    char    *PASSWD   = "MANAGER";    // user password
    char    *NLS      = "US7ASCII";   // NLS_USE ( KO16KSC5601, US7ASCII )
    char     connStr[1024];

    sprintf(connStr,
            "DSN=127.0.0.1;UID=%s;PWD=%s;CONNTYPE=%d;NLS_USE=%s", /*
;PORT_NO=20300", */
            USERNAME, PASSWD, 1, NLS);

    /* establish connection */
    if (SQLDriverConnect( dbc, NULL, (SQLCHAR *) connStr, SQL_NTS,
                          NULL, 0, NULL,
                          SQL_DRIVER_NOPROMPT ) != SQL_SUCCESS)
    {
        execute_err(dbc, SQL_NULL_HSTMT, "SQLDriverConnect");
        return SQL_ERROR;
    }

    conn_flag = 1;

    return SQL_SUCCESS;
}

SQLRETURN execute_select()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    SQLRETURN    rc;
    char         query[SQL_LEN];

    SQLINTEGER           id;
    char                 name[20+1];
    SQL_TIMESTAMP_STRUCT birth;
    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
    {
        printf("SQLAllocStmt error!!\n");
        return SQL_ERROR;
    }

    sprintf(query,"SELECT * FROM DEMO_EX6");
    if (SQLPrepare(stmt, (SQLCHAR *)query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    /* binds application data buffers to columns in the result set */
    if (SQLBindCol(stmt, 1, SQL_C_SLONG,
                   &id, 0, NULL) != SQL_SUCCESS)
    {
        printf("SQLBindCol error!!!\n");
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindCol(stmt, 2, SQL_C_CHAR,
                   name, sizeof(name), NULL) != SQL_SUCCESS)
    {
        printf("SQLBindCol error!!!\n");
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindCol(stmt, 3, SQL_C_TYPE_TIMESTAMP,
                   &birth, 0, NULL) != SQL_SUCCESS)
    {
        printf("SQLBindCol error!!!\n");
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

   /* fetches the next rowset of data from the result set and print to stdout
*/
    printf("id\t        Name\tbirth\n");

printf("=================================================================
=\n");
    if ( SQLExecute(stmt) != SQL_SUCCESS )
    {
        execute_err(dbc, stmt, "SQLExecute : ");
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    while ( (rc = SQLFetch(stmt)) != SQL_NO_DATA )
    {
        if ( rc != SQL_SUCCESS )
        {
            execute_err(dbc, stmt, query);
            break;
        }
        printf("%d%20s\t%4d/%02d/%02d %02d:%02d:%02d\n",
                id, name, birth.year, birth.month, birth.day,
                birth.hour, birth.minute, birth.second);
    }

    SQLFreeStmt(stmt, SQL_DROP);

    return SQL_SUCCESS;
}


SQLRETURN execute_proc()
{
    SQLHSTMT     stmt = SQL_NULL_HSTMT;
    char         query[SQL_LEN];

    SQLINTEGER           id;
    char                 name[20+1];
    SQL_TIMESTAMP_STRUCT birth;
    SQLINTEGER           ret = 0;

    SQLLEN               name_ind = SQL_NTS;

    /* allocate Statement handle */
    if (SQL_ERROR == SQLAllocStmt(dbc, &stmt))
    {
        printf("SQLAllocStmt error!!\n");
        return SQL_ERROR;
    }

    sprintf(query,"EXEC DEMO_PROC6( ?, ?, ?, ? )");

    /* prepares an SQL string for execution */
    if (SQLPrepare(stmt, (SQLCHAR *) query, SQL_NTS) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }

    if (SQLBindParameter(stmt,
                         1, /* Parameter number, starting at 1 */
                         SQL_PARAM_INPUT, /* in, out, inout */
                         SQL_C_SLONG, /* C data type of the parameter */
                         SQL_INTEGER, /* SQL data type of the parameter :
char(8)*/
                         0,          /* size of the column or expression,
precision */
                         0,          /* The decimal digits, scale */
                         &id,        /* A pointer to a buffer for the parame-
ter's data */
                       0,          /* Length of the ParameterValuePtr buffer
in bytes */
                         NULL        /* indicator */
                         ) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
        return SQL_ERROR;
    }
    if (SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
                         SQL_C_CHAR, SQL_VARCHAR,
                         20,  /* varchar(20) */
                         0,
                         name, sizeof(name), &name_ind) != SQL_SUCCESS)
    {
        execute_err(dbc, stmt, query);
        SQLFreeStmt(stmt, SQL_DROP);
```

```
            return SQL_ERROR;
        }
        if (SQLBindParameter(stmt, 3, SQL_PARAM_INPUT,
                             SQL_C_TYPE_TIMESTAMP, SQL_DATE,
                             0, 0, &birth, 0, NULL) != SQL_SUCCESS)
        {
            execute_err(dbc, stmt, query);
            SQLFreeStmt(stmt, SQL_DROP);
            return SQL_ERROR;
        }
        if (SQLBindParameter(stmt, 4, SQL_PARAM_OUTPUT,
                             SQL_C_SLONG, SQL_INTEGER,
                             0, 0, &ret,
                           0,/* For all fixed size C data type, this argument is
    ignored */
                             NULL) != SQL_SUCCESS)
        {
            execute_err(dbc, stmt, query);
            SQLFreeStmt(stmt, SQL_DROP);
            return SQL_ERROR;
        }

        /* executes a prepared statement */

        id = 5;
        sprintf(name, "name5");
        birth.year=2004;birth.month=5;birth.day=14;
        birth.hour=15;birth.minute=17;birth.second=20;
        birth.fraction=0;
        name_ind = 5;            /* name => length=5 */
        if (SQLExecute(stmt) != SQL_SUCCESS)
        {
            execute_err(dbc, stmt, query);
            SQLFreeStmt(stmt, SQL_DROP);
            return SQL_ERROR;
        }
        else
        {
            printf("\n======= Result of exec procedure ======\n");
            printf("ret => %d\n\n", ret);
        }

        SQLFreeStmt(stmt, SQL_DROP);

        return SQL_SUCCESS;
    }
```

Appendix B. Data Types

This appendix describes the data types of ALTIBASE HDB SQL data types, C data types, and the data conversion.

SQL Data Types

To find which data types the ALTIBASE HDB ODBC supports, call SQLGetTypeInfo (). The following table shows the list of the ALTIBASE HDB SQL data types and the standard SQL type identifier.

SQL type identifier	Data Types of ALTIBASE HDB	Comments
SQL_CHAR	CHAR(n)	Character string of fixed string length n.
SQL_VARCHAR	VARCHAR(n)	Variable-length character string with a maximum string length n.
SQL_WCHAR	NCHAR(n)	Unicode character data type with fixed length(n)N means the number of characters.
SQL_WVARCHAR	NVARCHAR(n)	Unicode character data type with variable length: If declared as fixed, SQL_WVARCHAR has a fixed length. If declared as variable, SQL_WVARCHAR has a variable length.N means the number of characters.
SQL_DECIMAL	DECIMAL(p, s)	See: NUMERIC(p, s)
SQL_NUMERIC	NUMERIC(p, s)	Signed, exact, numeric value with a precision p and scale s (1<=p<=38, -84<=s<=126)
SQL_SMALLINT	SMALLINT	2-byte integer data type(-2^15+1 ~ 2^15-1)
SQL_INTEGER	INTEGER	4-byte integer data type(-2^31+1 ~ 2^31-1)
SQL_BIGINT	BIGINT	8-byte integer data type(-2^63+1 ~2^63-1)
SQL_REAL	REAL	The same data type as Float of C
SQL_FLOAT	FLOAT(p)	Fixed decimal numeric type data from -1E+120 to 1E+120 (1<=p<=38)
SQL_DOUBLE	DOUBLE	The same data type with DOUBLE of C
SQL_BINARY	BLOB(n)	Binary data type size of n
SQL_TYPE_DATE	DATE	Year, month, and day fields, conforming to the rules of the Gregorian calendar.

SQL type identifier	Data Types of ALTIBASE HDB	Comments
SQL_TYPE_TIME	DATE	Hour, minute, and second fields, with valid values for hours of 00 to 23, valid values for minutes of 00 to 59, and valid values for seconds of 00 to 61.
SQL_TYPE_TIMESTAMP	DATE	Year, month, day, hour, minute, and second fields, with valid values as defined for the DATE and TIME data types.
SQL_INTERVAL	-	The result type of the DATE – DATE
SQL_BYTES	BYTE(n)	Binary data type with the fixed length as long as the specified size (1 byte<=n<=32000 bytes)
SQL_NIBBLE	NIBBLE(n)	Binary data type with the fixed length as long as the changeable size (n) (1 byte<=n<=255 bytes)
SQL_GEOMETRY	GEOMETRY	See: ALTIBASE HDB Spatial Temporary SQL document

C Data Types

C data types refer to the data type of C buffer used to store the data in an application.

C data type is specified with the type argument in SQLBindCol () and SQLGetData (), and in SQLBind-Parameter () with cType.

The following table is the list of valid type identifiers for C data type. Also, the table lists the definitions of C data type of the ODBC and the data types corresponding to each identifier.

C Type Identifier	ODBC C typedef	C type
SQL_C_CHAR	SQLCHAR *	unsigned char *
SQL_C_BIT	SQLCHAR	unsigned char
SQL_C_WCHAR	SQLWCHAR *	short *
SQL_C_STINYINT	SQLSCHAR	signed char
SQL_C_UTINYINT	SQLCHAR	unsigned char
SQL_C_SBIGINT	SQLBIGINT	_int64
SQL_C_UBIGINT	SQLUBIGINT	unsigned _int64
SQL_C_SSHORT	SQLSMALLINT	short int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_SLONG	SQLINTEGER	int
SQL_C_ULONG	SQLUINTEGER	unsigned int

C Type Identifier	ODBC C typedef	C type
SQL_C_FLOAT	SQLREAL	float
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_BINARY	SQLCHAR *	unsigned char *

C Type Identifier	ODBC C typedef	C type

SQL_C_TYPE_DATE	SQL_DATE_STRUCT	struct
		tagDATE_STRUCT {
		SQLSMALLINT year;
		SQLSMALLINT month;
		SQLSMALLINT day;
		} DATE_STRUCT

SQL_C_TYPE_TIME	SQL_TIME_STRUCT	struct
		tagTIME_STRUCT {
		SQLSMALLINT hour;
		SQLSMALLINT minute;
		SQLSMALLINT second;
		} TIME_STRUCT

SQL_C_TYPE_TIMESTAMP	SQL_TIMESTAMP_STRUCT	struct
		tagTIMESTAMP_STRUCT {
		SQLSMALLINT year;
		SQLSMALLINT month;
		SQLSMALLINT day;S
		QLSMALLINT hour;
		SQLSMALLINT minute;
		SQLSMALLINT second;
		SQLINTEGER fraction;
		} TIMESTAMP_STRUCT;

SQL_C_BYTES	SQLCHAR *	unsigned char *

SQL_C_NIBBLE	SQL_NIBBLE_STRUCT	struct
		tagNIBBLE_STRUCT {
		SQLCHAR length;
		SQLCHAR value[1];
		} NIBBLE_STRUCT

Converting SQL Data into C Data Types

	Converting Data from SQL to C Data types																	
	SQL_C_CHAR	SQL_C_BIT	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_SSHORT	SQL_C_USHORT	SQL_C_SLONG	SQL_C_ULONG	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_BINARY	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_BYTES	SQL_C_NIBBLE
SQL_CHAR	#	○	○	○									○					
SQL_VARCHAR	#	○	○	○									○					
SQL_DECIMAL	#	○	○	○	○	○	○	○	○	○	#	○	○					
SQL_NUMERIC	#	○	○	○	○	○	○	○	○	○	#	○	○					
SQL_SMALLINT(signed)	○	○	○	○	○	○	#	○	○	○	○	○	○					
SQL_INTEGER(signed)	○	○	○	○	○	○	○	○	#	○	○	○	○					
SQL_BIGINT(signed)	○	○	○	○	#	○	○	○	○	○	○	○	○					
SQL_REAL	○	○	○	○	○	○	○	○	○	○	#	○	○					
SQL_FLOAT	#	○	○	○	○	○	○	○	○	○	#	○	○					
SQL_DOUBLE	○	○	○	○	○	○	○	○	○	○	○	#	○					
SQL_BINARY	○												#					
SQL_TYPE_DATE	○												○	#		○		
SQL_TYPE_TIME	○												○		#	○		
SQL_TYPE_TIMESTAMP	○												○	○	○	#		
SQL_INTERVAL	○										○	#	○					
SQL_BYTES	○												○				#	
SQL_NIBBLE	○												○					#
SQL_GEOMETRY													#					

# : Default conversion

○ : Supported conversion

Converting C Data into SQL Data types

	Converting Data from C to SQL Data types															
	SQL_CHAR	SQL_VARCHAR	SQL_DECIMAL	SQL_NUMERIC	SQL_SMALLINT(signed)	SQL_INTEGER(signed)	SQL_BIGINT(signed)	SQL_REAL	SQL_FLOAT	SQL_DOUBLE	SQL_BINARY	SQL_DATE	SQL_INTERVAL	SQL_BYTES	SQL_NIBBLE	SQL_GEOMETRY
SQL_C_CHAR	#	#	#	#	○	○	○	○	○	○	○	○		○	○	
SQL_C_BIT	○	○	○	○	○	○	○	○	○	○						
SQL_C_STINYINT	○	○	○	○	○	○	○	○	○	○						
SQL_C_UTINYINT	○	○	○	○	○	○	○	○	○	○						
SQL_C_SBIGINT	○	○	○	○	○	○	#	○	○	○						
SQL_C_UBIGINT	○	○	○	○	○	○	○	○	○	○						
SQL_C_SSHORT	○	○	○	○	#	○	○	○	○	○						
SQL_C_USHORT	○	○	○	○	○	○	○	○	○	○						
SQL_C_SLONG	○	○	○	○	○	#	○	○	○	○						
SQL_C_ULONG	○	○	○	○	○	○	○	○	○	○						
SQL_C_FLOAT	○	○	○	○	○	○	○	#	○	○						
SQL_C_DOUBLE	○	○	○	○	○	○	○	○	#	#						
SQL_C_BINARY	○	○									#					○
SQL_C_TYPE_DATE																
SQL_C_TYPE_TIME												○				
SQL_C_TYPE_TIMESTAMP												○				
SQL_C_BYTES												○		#		
SQL_C_NIBBLE															#	

# : Default conversion

○ : Supported conversion

Appendix C. ODBC Error Codes

SQLERROR returns SQLSTATE as defined in X/Open and SQL Access Group SQL CAE specification (1992). SQLSTATE is a five-digit character string. This Chapter contains the ALTIBASE HDB ODBC Drive Error refrence.

ODBC Error Codes

SQLSTATE	Error	Can be returned from

01004	String data, right-truncated	`SQLDescribeCol`
		`SQLFetch`
		`SQLGetData`

07006	Restricted data type attribute violation	`SQLBindParameter`
		`SQLExecute`
		`SQLFetch`

07009	Invalid descriptor index	`SQLBindCol`
		`SQLBindParameter`
		`SQLColAttribute`
		`SQLDescribeCol`
		`SQLDescribeParam`
		`SQLGetData`

* For SQLSTATE 08001, 08002, 08003, and 08S01, see the next table.

SQLSTATE	Error	Can be returned from

HY000	General error	`SQLAllocStmt`
		`SQLAllocConnect`
		`SQLBindCol`
		`SQLBindParameter`
		`SQLColAttribute`
		`SQLColumns`
		`SQLConnect`
		`SQLDescribeCol`
		`SQLDisconnect`
		`SQLDriverConnect`
		`SQLEndTran`
		`SQLExecDirect`
		`SQLExecute`
		`SQLFetch`
		`SQLFreeHandle`
		`SQLFreeStmt`
		`SQLGetData`
		`SQLNumParams`
		`SQLNumResultCols`
		`SQLPrepare`
		`SQLPrimaryKeys`
		`SQLProcedureColumns`
		`SQLProcedures`
		`SQLRowCount`
		`SQLSetAttribute`
		`SQLSetConnectAttr`
		`SQLSetEnvAttr`
		`SQLStatistics`
		`SQLTables`

HY001	Memory allocation error (Cannot allocate the requested memory for the ODBC to execute and complete the function.)	`SQLAllocConnect`
		`SQLAllocStmt`
		`SQLBindCol`
		`SQLBindParameter`
		`SQLConnect`
		`SQLDriverConnect`
		`SQLExecDirect`
		`SQLGetTypeInfo`
		`SQLPrepare`

HY003	An application buffer type is not valid. (cType Argument value is not a valid C data type.)	`SQLBindCol` `SQLBindParameter`

HY009	Used an invalid pointer (null pointer)	`SQLAllocConnect`
		`SQLAllocStmt`
		`SQLBindParameter`
		`SQLExecDirect`
		`SQLForeignKeys`
		`SQLPrimaryKeys`
		`SQLProcedureColumns`
		`SQLProcedures`
		`SQLSpecialColumns`
		`SQLStatistics`
		`SQLTablePrivileges`

HY010	Function sequence error	`SQLAllocStmt`
		`SQLDescribeParam`
		`SQLGetData`

SQLSTATE	Error	Can be returned from

HY090	Invalid character string or buffer	`SQLBindParameter`
		`SQLDescribeCol`
		`SQLExecute`
		`SQLForeignKeys`
		`SQLGetData`
		`SQLGetStmtAttr`
		`SQLTablePrivileges`

HY092	Invalid attribute or option	`SQLGetStmtAttr`
HY105	Invalid parameter type	`SQLBindParameter`
HYC00	Used an attribute not supported.	`SQLGetConnectAttr` `SQLGetStmtAttr`

Database Connection-related Error Codes

SQLSTATE	Code	Error	Can be returned from
HY000	0x51001	The character set does not exist.	`SQLConnect` `SQLDriverConnect`
	0x5003b	The communication buffer is insufficient. (It exceeded the length of the communication buffer.)	`SQLExecute`
HY001	0x5104A	Memory allocation error (Cannot allocate the memory requested for the SQLCLI to execute the function and complete execution)	`SQLConnect` `SQLDriverConnect`
08001	0x50032	The ODBC cannot set the connection to a database.	`SQLConnect` `SQLDriverConnect`

Network-related Error Codes

SQLSTATE	Code	Error	Can be returned from
08002	0x51035	The corresponding dbc is already connected to the database.	`SQLConnect` `SQLDriverConnect`

08003	0x51036	Connection does not exist.	`SQLExecDirect`
			`SQLExecute`
			`SQLPrepare`

SQLSTATE	Code	Error	Can be returned from
08S01	0x51043	Communication channel error (Communication channel failure before the function is processed between the SQLCLI and the data-base.)	`SQLColumns` `SQLConnect` `SQLDriverConnect` `SQLExecDirect` `SQLExecute` `SQLFetch` `SQLForeignKeys` `SQLGetConnectAttr` `SQLPrepare` `SQLPrimaryKeys` `SQLProcedureCol-umns` `SQLProcedures` `SQLSetConnectAttr` `SQLSpecialColumns` `SQLStatistics` `SQLTablePrivileges` `SQLTables`

Statement State Transition-related Errors

The following table shows how each status is converted when the ODBC function that uses the corresponding handle type (environment, connection, or statement) is called in the statement status.

Summited statements have the following status:

State	Description
S0	Unallocated statement. (The connection state must be connected connection.)
S1	Allocated statement.
S2	Prepared statement. (A (possibly empty) result set will be created.)
S6	Cursor positioned with SQLFetch.

The entry values in the conversion table are as follows:

When the status is not converted after the function is executed

Sn : When the command status is converted into a specified status

(IH) : Invalid Handle

(HY010) : Function sequence error

(24000) : Invalid Cursor State

Note

S : Success. In this case, the function returns one of the following values: SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.

E : Error. In this case, the function returns SQL_ERROR:

R : Results. There is result set when Result command is executed. (There is possibility that the result set is empty set.)

NR : No Results. No result set when the command is executed.

NF : No Data Found. The function returns SQL_NO_DATA.

RD : Receive Done

P : Prepared. A statement was prepared.

NP : Not Prepared. A statement was not prepared.

The following example shows how to view a statement state transition table for SQLPrepare function:

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	S => S1	S => S2	(24000)
		E => S1	

If SQLPrepare function is called when the handle type is SQL_HANDLE_STMT and the command status is S0, the ODBC administrator will return SQL_INVALID_Handle (IH). If SQLPrepare function is called and successfully executed when the handle type is SQL_HANDLE_STMT and the status is S1, S1 status will be kept. If SQLPrepare function is called and successfully executed when the handle type is SQL_HANDLE_STMT and the status is converted to S2, the status of the command will be converted into S2. Otherwise, the command status will remain as S1 as it was. If the function is called while the handle type is SQL_HANDLE_STMT and the status is S6, the ODBC administrator always returns SQL_ERROR and SQLSTATE 24000 (Invalid Cursor State).

SQLAllocHandle

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
--	--	--	--
S1*			

* When *HandleType* is SQL_HANDLE_STMT

SQLBindCol

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	--	--	--

SQLBindParameter

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	--	--	--

SQLColumns, SQLGetTypeInfo, SQLPrimaryKeys, SQLProcedureColumns, SQLProcedures, SQLStatistics, SQLTables

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	S => S6	E => S1	(24000)
		S => S6	

SQLConnect

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(Error)	(Error)	(Error)	(Error)

SQLDisconnect

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
--	* => S0	* => S0	* => S0

When the *function SQLDisconnect is called, all commands related to the connection handle will be terminated. This function converts the connection status into allocated connection status.; Before the command status becomes S0, the connection status will become C4 (connected connection).

See SQLDriverConnect: SQLConnect

SQLExecDirect

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	S, NR => --	S, NR => S1	(24000)
	S, R => S6	S, R => S6	
		E => S1	

SQLExecute

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	(HY010)	S, NR => --	(24000)
		S, R => S6	
		E => --	

SQLFetch

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	(HY010)	(HY010)	S => --
			RD || NF || E => (if NP => S1, if P => S2)

SQLFreeHandle

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
--	(HY010)	(HY010)	(HY010)
(IH)	S0	S0	S0

(1) When the handle type of the first row is SQL_HANDLE_ENV or SQL_HANDLE_DBC:

(2) When the handle type of the second row is SQL_HANDLE_STMT:

SQLFreeStmt

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	--	--	NP = > S1
			P => S2
(IH)	S0	S0	S0

(1) When fOption of the first row is SQL_CLOSE:

(2) When fOption of the second row is SQL_DROP:

SQLGetData

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	(HY010)	(HY010)	S || NF => --

SQLGetTypeInfo: See SQLColumns.

SQLNumResultCols

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	(HY010)	S => --	S => --

SQLPrepare

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	S => --	S => --	(24000)
		E => S1	

SQLPrimaryKeys: See SQLColumns.

SQLProcedureColumns: See SQLColumns.

SQLProcedures: See SQLColumns.

SQLSetConnectAttr

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
--*	--	--	(24000)

When the *set attribute is the connection attribute: When the set attribute has the statement attribute, SQLSetStmtAttr will be referred to.

SQLSetEnvAttr

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(Error)	(Error)	(Error)	(Error)

SQLSetStmtAttr

S0Unallocated	S1Allocated	S2Prepared	S6Infetch
(IH)	--	(1) => --	(1) => --
		(2) => (Error)	(2) => (24000)

(1) When *Attribute argument is neither*

SQL_ATTR_CONCURRENCY,
SQL_ATTR_CURSOR_TYPE,
SQL_ATTR_SIMULATE_CURSOR,
SQL_ATTR_USE_BOOKMARKS,
SQL_ATTR_CURSOR_SCROLLABLE, nor
SQL_ATTR_CURSOR_SENSITIVITY

(2) When *Attribute argument is either*

SQL_ATTR_CONCURRENCY,
SQL_ATTR_CURSOR_TYPE,
SQL_ATTR_SIMULATE_CURSOR,
SQL_ATTR_USE_BOOKMARKS,
SQL_ATTR_CURSOR_SCROLLABLE, or
SQL_ATTR_CURSOR_SENSITIVITY

SQLStatistics: See SQLColumns.

SQLTables: See SQLColumns.

State Transition Tables

The followings summarize the major functions that affect the state transition:

Current State / Request	S0 UNALLOCATED	S1 ALLOCATED	S2 PREPARED	S6 INFETCH
Prepare	(IH)	S => S1	S => S2	(24000)
			E => S1	
ExecDirect	(IH)	S,NR => S1	S,NR => S1	(24000)
		S,R => S6	S,R => S6	
			E => S1	
Execute	(IH)	(HY010)	S,NR => S2	(24000)
			S,R => S6	
			E => S2	
Fetch	(IH)	(HY010)	(HY010)	S => S6
				RD || NF || E => (if NP => S1, if P => S2)
FreeStmt(CLOSE)	(IH)	S1	S2	NP => S1
				P => S2

Current State / Request	S0 UNALLOCATED	S1 ALLOCATED	S2 PREPARED	S6 INFETCH
FreeStmt(DROP)	(IH)	S0	S0	S0

Cf)

- (IH) : Invalid Handle (HY010) : Function Sequence Error

- (24000) : Invalid Cursor State

- S : Success	E : Error except Network Error		
- R : Results	NR : No Results	NF : No data Found	RD: Receive Done
- P : Prepared	NP : Not Prepared		

Appendix D. ODBC Conformance Levels

This appendix describes the conformance level of ALTIBASE HDB ODBC Driver.

Interface Conformance Levels

The purpose of classifying the conformance levels is to have the information about the features that the ODBC driver supports. There are total three conformance levels. To meet the specific conformance level, all the corresponding requirements for such level should be met.

The following table shows the conformance level in compliance with ODBC 3.x, which is different from ODBC 2.x conformance level. Conformance level 1 in compliance with ODBC2.x can be considered as the core conformance.

ALTIBASE HDB ODBC is implemented in compliance with ODBC 2.x so that it can be considered as the core in the following table. However, it may not support functions of ODBC 3.0 specification.

Function Conformance Level

Function Name	Level	Support status	To be supported	Remarks
SQLAllocHandle	Core	O		
SQLBindCol	Core	O		
SQLBindParameter	Core	O		
SQLBrowseConnect	Level1	X	X	
SQLBulkOperations	Level1	X	X	
SQLCancel	Core	X	O	
SQLCloseCursor	Core	O		
SQLColAttribute	Core	O		
SQLColumnPrivileges	Level2	X	X	Privileges about the column are not supported by ALTIBASE HDB.
SQLColumns	Core	O		
SQLConnect	Core	O		

Function Name	Level	Support status	To be supported	Remarks
QLCopyDesc	Core	X	O	
SQLDescribeCol	Core	O		
SQLDescribeParam	Level2	O		Not fully supported.
SQLDisconnect	Core	O		
SQLDriverConnect	Core	O		
SQLEndTran	Core	O		
SQLExecDirect	Core	O		
SQLExecute	Core	O		
SQLFetch	Core	O		
SQLFetchScroll	Core	O		
SQLForeignKeys	Level2	O		
SQLFreeHandle	Core	O		
SQLFreeStmt	Core	O		
SQLGetConnectAttr	Core	O		
SQLGetCursorName	Core	O		
SQLGetData	Core	O		
SQLGetDescField	Core	O		ODBC 3.0
SQLGetDescRec	Core	O		ODBC 3.0
SQLGetDiagField	Core	O		ODBC 3.0
SQLGetDiagRec	Core	O		ODBC 3.0
SQLGetEnvAttr	Core	O		
SQLGetFunctions	Core	O		
SQLGetStmtAttr	Core	O		
SQLGetTypeInfo	Core	O		
SQLMoreResults	Level1	O		
SQLNativeSql	Core	O		
SQLNumParams	Core	O		
SQLNumResultCols	Core	O		
SQLParamData	Core	O		
SQLPrepare	Core	O		

Function Name	Level	Support status	To be supported	Remarks
SQLPrimaryKeys	Level1	O		
SQLProcedureColumns	Level1	O		
SQLProcedures	Level1	O		
SQLPutData	Core	O		
SQLRowCount	Core	O		
SQLSetConnectAttr	Core	O		
SQLSetCursorName	Core	O		
SQLSetDescField	Core	O		ODBC 3.0
SQLSetDescRec	Core	O		ODBC 3.0
SQLSetEnvAttr	Core	O		
SQLSetPos	Level1	X	O	
SQLSetStmtAttr	Core	O		
SQLSpecialColumns	Core	O		
SQLStatistics	Core	O		
SQLTablePrivileges	Level2	O		
SQLTables	Core	O		

Appendix E. Upgrade

This appendix describes the requirements to make ODBC applications for ALTIBASE HDB 4 available for ALTIBASE HDB 5 as follows. The level of CLI interface has been improved since the upgrade to ALTIBASE HDB 5. Especially it has high compatibility with user applications and general purpose applications to follow the standard of X/Open CLI or ODBC specification to the highest degree.

This appendix explains data types newly added or defined, and other changes.

- Data Types
- Other Changes

Data Type

This section describes data types newly added to ALTIBASE HDB 5. You can resolve the problems derived from compiling the existing applications to firmly keep the standard compared to previous version.

SQLCHAR, SQLSCHAR

The previous CLI applications have used SQLCHAR and char together. However, standard-oriented SQLCHAR is defined newly as follows.

```
typedef unsigned char SQLCHAR;
typedef signed char SQLSCHAR;
```

Therefore, errors occur when you compile the existing applications with the following statements.

```
char *query = “....”;
SQLPrepare(stmt, query, SQL_NTS);
```

You have only to modify type casting as follows to solve this problem.

```
char *query = “....”;
SQLPrepare(stmt, (SQLCHAR *)query, SQL_NTS);
```

SQL_BIT, SQL_VARBIT

Subsequent releases, starting with version 5, support BIT type as standard SQL92 and VARBIT type for your convenience. Refer to *SQL Reference* for details about them.

BIT to C type

The following indicates conversion table related to BIT.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
`SQL_C_CHAR`	BufferLength > 1	Data ('0' or '1')	1	n/a
	BufferLength <= 1	Undefined	Undefined	22003

`SQL_C_STINYINT`
`SQL_C_UTINYINT`
`SQL_C_SBIGINT`
`SQL_C_UBIGINT`
`SQL_C_SSHORT`
`SQL_C_USHORT`
`SQL_C_SLONG`
`SQL_C_ULONG`
`SQL_C_FLOAT`
`SQL_C_DOUBLE`
`SQL_C_NUMERIC`
	None(The values of BufferLength have the fixed type like this, and they are ignored in case of coversion.)	Data (0 or 1)	Size of C type	n/a

`SQL_C_BIT`	None	Data (0 or 1)	1	n/a
`SQL_C_BINARY`	None	Data (See below for formats)	Data length to be written in the memory the user binds	n/a

VARBIT to C type

The following indicates conversion table related to VARBIT.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
`SQL_C_CHAR`	BufferLength > 1	Data	Precision of varbit	n/a
	BufferLength <= 1	Undefined	Undefined	22003
`SQL_C_BIT`	None	Data (0 or 1)	1	n/a
`SQL_C_BINARY`		Data (Its format is same as that of BIT)	Data length to be written in the memory the user binds	n/a

C type to BIT/VARBIT

No type is converted to BIT currently.

Binary Format

0 7	8 15	16 23	24 31	32 39	...	8n 8n+7
Precision				Data		

```
where n= (Precision+7)/8 + 3

Precision : Length of BIT data
Data : BIT Data
```

Data Type Conversion Example

BIT/VARBIT SQL_C_BINARY

Data themselves are sent from server to user when you bind and fetch BIT to SQL_C_BINARY. Data formats sotred in user buffer are as mentioned above.

You can access to the server conveniently if specifying struct bit_t as the following examples and using it.

```
CREATE TABLE T1(I1 BIT(17), I2 VARBIT(37));
INSERT INTO T1 VALUES(BIT'11111011010011011',
VARBIT'0010010010101110001010100010010011011');
INSERT INTO T1 VALUES(BIT'110011011',
VARBIT'00111000101010001001001l011');
----------------------
void dump(unsigned char *Buffer, SQLINTEGER Length)
{
for (SQLINTEGER i = 0; i < Length; i++)
{
printf("%02X ", *(Buffer + i));
}
}

typedef struct bit_t
{
SQLUINTEGER mPrecision;
unsigned char mData[1];
} bit_t;

bit_t *Bit;
bit_t *Varbit;
SQLLEN Length;
SQLRETURN rc;

Bit = (bit_t *)malloc(BUFFER_SIZE);
Varbit = (bit_t *)malloc(BUFFER_SIZE);

SQLBindCol( stmt, 1,
SQL_C_BINARY,
(SQLPOINTER)Bit,
BUFFER_SIZE,
&LengthBit);

SQLBindCol( stmt, 2,
SQL_C_BINARY,
(SQLPOINTER)Varbit,
BUFFER_SIZE,
&LengthVarbit);
do
{
memset(Buffer, 0, BUFFER_SIZE);
rc = SQLFetch(stmt);

printf("-----\n");
printf(">> Bit\n");
printf("Length : %d\n", LengthBit);
printf("Precision : %d\n", Bit->mPrecision);
dump(Bit->mData, LengthBit - sizeof(SQLUINTEGER));
printf(">> Varbit\n");
printf("Length : %d\n", LengthVarbit);
printf("Precision : %d\n", Varbit->mPrecision);
dump(Varbit->mData, LengthVarbit - sizeof(SQLUINTEGER));
} while (rc != SQL_NO_DATA);
```

When you execute the program above, the results are as follows.

```
------
>> Bit
Length : 7
Precision : 17
FB 4D 80 (1111 1011 0100 1101 1)
>> Varbit
Length : 9
Precision : 37
24 AE 2A 24 D8 (0010 0100 1010 1110 0010 1010 0010 0100 1101 1)
------
>> Bit
Length : 7
Precision : 17 -> Precision indicates 17 because "0" bit is appended.
CD 80 00 (1100 1101 1000 0000)
>> Varbit
Length : 8
Precision : 27 -> Precision indicates 27 because VARBIT doesn't perform pad-
ding.
38 A8 93 60 (0011 1000 1010 1000 1001 0011 011)
```

If BUFFER_SIZE is less than required, SQLFetch() returns SQL_SUCCESS_WITH_INFO, and wirtes its data in the memory bound as BUFFER_SIZE.

BIT/VARBIT to SQL_C_BIT

SQL_C_BIT of ODBC requires special care because it is the unsigned 8bit integer whose value is 0 or 1. In other words, bound variables don't have 0x64 but 0x01 even though BIT '011001' is stored on the table of server when you bind them with SQL_C_BIT and fetch them.

BIT to SQL_C_CHAR

If you bind BIT column with SQL_C_CHAR when fetching it, the result always has 0 or 1 following ODBC type conversion rules.

```
CREATE TABLE T1 (I1 BIT(12));
INSERT INTO T1 VALUES(BIT'110011000010');
INSERT INTO T1 VALUES(BIT'010011000010');

SQLCHAR sData[128];
SQLLEN sLength;

sQuery = (SQLCHAR *)"SELECT I1 FROM T1";

SQLBindCol(stmt, 1, SQL_C_CHAR, sData, sizeof(sData), sLength);

SQLExecDirect(stmt, sQuery, SQL_NTS);

while (SQLFetch(stmt) != SQL_NO_DATA)
{
```

```
printf("bit value = %s, ", sData);
printf("sLength = %d\n", sLength);
}
```

If you execute program above, the following is displayed on the screen.

```
1, sLength = 1
0, sLength = 1
```

VARBIT to SQL_C_CHAR

All data in the column are fetched when you fetch VARBIT columns because conversion tool is made itself without VARBIT types in ODBC standard.

```
CREATE TABLE T1 (I1 VARBIT(12));
INSERT INTO T1 VALUES(VARBIT'110011000010');
INSERT INTO T1 VALUES(VARBIT'01011010');

SQLCHAR sData[128];
SQLLEN sLength;
sQuery = (SQLCHAR *)"SELECT I1 FROM T1";
SQLBindCol(stmt, 1, SQL_C_CHAR, sData, sizeof(sData), &sLength);
SQLExecDirect(stmt, sQuery, SQL_NTS);
while (SQLFetch(stmt) != SQL_NO_DATA)
{
printf("bit value = %s, ", sData);
printf("sLength = %d\n", sLength);
}
```

If you execute program above, the following is displayed on the screen.

```
110011000010, sLength = 12
01011010, sLength = 8
```

SQL_NIBBLE

SQL_C_NIBBLE supported in ALTIBASE HDB 4 is not available to ALTIBASE HDB 5. However, you can fetch data with SQL_C_BINARY.

NIBBLE to C type

Conversion is available only to SQL_C_CHAR and SQL_C_BINARY.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
SQL_C_CHAR	BufferLength > 1	Data ('0' or '1')	Data length to be written in the memory the user binds (Except null and termination character)	n/a
	BufferLength <= 1	Undefined	Undefined	22003

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
SQL_C_BINARY	None	Data (See below for formats)	Data length to be written in the memory the user binds	n/a

NIBBLE is fetched in binary format in the same way as BIT, but binary format of NIBBLE is different from that of BIT because its precision field has 1 byte integer.

Binary Format

0 7	8 15	...	8n 8n+7
Precision		Data	

```
Where n = (Precision+1)/2

Precision : Length of NIBBLE data
Data : Nibble Data
```

Data Type Conversion Example

NIBBLE to SQL_C_BINARY

Data themselves are sent from server to user when you bind and fetch NIBBLE to SQL_C_BINARY. Data types stored in user buffer are as metioned above.

You can access to the server conveniently if specifying nibble_t as the results are as follows.

```
CREATE TABLE T1(I1 NIBBLE, I2 NIBBLE(10), I3 NIBBLE(21) NOT NULL);
INSERT INTO T1 VALUES(NIBBLE'A', NIBBLE'0123456789', NIB-
BLE'0123456789ABCDEF00121');
INSERT INTO T1 VALUES(NIBBLE'B', NIBBLE'789', NIBBLE'ABCD1234');

-------------------

void dump(unsigned char *Buffer, int Length)
{
for (int i = 0; i < Length; i++) printf("%02X ", *(Buffer + i));
}
typedef struct nibble_t
{
unsigned char mPrecision;
unsigned char mData[1];
} nibble_t;
nibble_t *Buffer;

SQLLEN Length;
SQLRETURN rc;

Buffer = (nibble_t *)malloc(BUFFER_SIZE);

SQLBindCol(stmt, 2, SQL_C_BINARY, (SQLPOINTER)Buffer, BUFFER_SIZE, &Length);
do
{
```

```
memset(Buffer, 0, BUFFER_SIZE);
rc = SQLFetch(stmt);

printf("----\n");
printf("Length : %d\n", Length);
printf("Precision : %d\n", Buffer->mPrecision);
dump(Buffer->mData, Length - sizeof(SQLUINTEGER));
} while (rc != SQL_NO_DATA);
```

When you execute the program above, the results are as follows.

```
Length : 6
Precision : 10
01 23 45 67 89
----
Length : 3
Precision : 3
78 90
```

NIBBLE to SQL_C_CHAR

Examples and results are omitted cause of no unusual events in this case.

SQL_BYTE

This is bound to SQL_C_BINARY instead of SQL_C_BYTE in ALTIBASE HDB 4 and then is executed in the same way as this is in ALTIBASE HDB 4.

BYTE to C types

Conversion to other types is not available except SQL_C_CHAR and SQL_C_BINARY. However, original data requires special care that its 1 byte is expressed as ASCII 2 characters when you convert binary data to SQL_C_CHAR.

C type id	Test	*TargetValuePtr	*StrLen_or_IndPtr	SQLSTATE
`SQL_C_CHAR`	(Byte length of data) * 2 < BufferLength	Data	Length of data in bytes	n/a
	(Byte length of data) * 2 >= BufferLength	Truncated data	Length of data in bytes	01004
`SQL_C_BINARY`	Byte length of data <= BufferLength	Data	Length of data in bytes	n/a
	Byte length of data > BufferLength	Truncated data	Length of data in bytes	01004

Each byte of binary data is always converted to a pair of hex characters when ODBC CLI of ALTIBASE HDB converts them to SQL_C_CHAR. Therefore, if buffer size of bound SQL_C_CHAR indicates the even, NULL termination character is printed not in the last byte of it but in ahead of that.

Source binary data(hex characters	Size of bound buffer(bytes)	Contents of the buffer bound as SQL_C_CHAR	*StrLen_or_IndPtr	SQLSTATE
AA BB 11 12	8	hex : 41 41 42 42 31 31 00 ?? string : “AABB11”	8	01004
	9	hex : 41 41 42 42 31 31 31 32 00 string : “AABB1112”	8	n/a

Binary Format

Byte type doesn’t have special format like NIBBLE or BIT, but is the conjunction of binary data.

Data Type Conversion Example

BYTE to SQL_C_CHAR

```
CREATE TABLE T1(I1 BYTE(30));
INSERT INTO T1 VALUES(BYTE’56789ABC’);

-------------------
SQLLEN Length;
SQLRETURN rc;

// execution query : SELECT * FROM T1;
Buffer = (nibble_t *)malloc(BUFFER_SIZE);

SQLBindCol(stmt, 1, SQL_C_CHAR, (SQLPOINTER)Buffer, BUFFER_SIZE, &Length);
do
{
memset(Buffer, 0, BUFFER_SIZE);
rc = SQLFetch(stmt);
printf(“Length : %d\n”, Length);
printf(“Data : %s\n”, Length);
} while (rc != SQL_NO_DATA);
```

When you execute the program above, the results are as follows.

```
Execution Query 1 : BUFFER_SIZE >= 9
Length : 8
Data : 56789ABC

Execution Query 2 : BUFFER_SIZE == 8
Length : 8
Data : 56789A 8 This is bound in byte buffer, and then only 6 characters are
expressed.

Execution Query 3 : BUFFER_SIZE == 7
Length : 8
Data : 56789A same as BUFFER_SIZE == 8

Execution Query 4 : BUFFER_SIZE == 6
Length : 8
Data : 5678
```

SQLFetch() retunrs SQL_SUCCESS_WITH_INFO for execution results except 1. SQLSTATE indicates 01004.

BYTE to SQL_C_BINARY

No unusual events for binding to binary in this case.

DATE : SQL_TYPE_TIMESTAMP

SQL_TYPE_TIMESTAMP is retuned when ALTIBASE HDB 5 inserts data as SQL Type into DATE column with using SQLDescribeCol() or SQLColAttribute(). SQL_TYPE_TIMESTAMP is similar to ALTIBASE HDB DATE type of ODBC standard, and consists of year, month, day, hour and second.

However, if you call SQLColAttribute() or SQLDescribeCol(), SQL_DATE is returned as SQL type because DATE type in ALTIBASE HDB 4 consists of basic elements such as day and hour, and much data such as special characters separating basic elements. Therefore, when you use ODBC of ALTIBASE HDB 5, SQL_TYPE_DATE, SQL_TYPE_TIME and SQL_TYPE_TIMESTAMP as constant numbers for ODBC 3.0 are recommended.

LOB

Data Type

In ALTIBASE HDB 4, length of LOB type is limited to page size. However, it consists of BLOB and CLOB supporting maximum 2GB.

ALTIBASE HDB 4 DDL

```
CREATE TABLE T1 (I1 BLOB(3));
```

ALTIBASE HDB 5 DDL

```
CREATE TABLE T1 (I1 BLOB); ---> This precision in brackets disappears.
```

LOB Function Use

CLI application supports private functions to use LOB. Refer to LOB Interface of ODBC Reference for details about special functions for LOB.

You can use functions available to general binary and character type except functions for LOB. You can store and search LOB data with standard ODBC in ODBC application cause of these features.

You can't update and retrieve data partially in ODBC application, whereas you can in CLI application with SQLGetLob and SQLPutLob.

```
CREATE TABLE T1 (I1 BLOB, I2 CLOB); // CONNECTION의 AUTOCOMMIT This makes off
mode.

SQLCHAR sBlobData[128];
SQLCHAR sClobData[128];
SQLLEN sBlobLength;
SQLLEN sClobLength;
```

```
SQLCHAR *sQuery = (SQLCHAR *)"INSERT INTO T1 VALUES(?, ?)";

SQLPrepare(stmt, sQuery, SQL_NTS);
SQLBindParameter(stmt, 1, SQL_C_BINARY, SQL_BLOB,0, 0, sBlobData,
                 sizeof(sBlobData), &sBlobLength);
SQLBindParameter(stmt, 2, SQL_C_CHAR, SQL_CLOB,0, 0, sClobData, sizeof
                (sClobData), &sClobLength);
sBlobLength = create_blob_data(sBlobData);
sprintf((char *)sClobData, "this is clob data");
sClobLength = SQL_NTS;

SQLExecute(stmt);
```

Using LOB in ODBC application

If you want to fetch LOB column in ODBC application and store data in LOB column, call SQLDescribeCol, SQLColAttribute or SQLDescribeParam.

If you execute these functions in LOB column, they are returned as data types of SQL_BLOB and SQL_CLOB. However, ODBC application doesn't recognize data types such as SQL_BLOB or SQL_CLOB.

Therefore, you may return them as data type which ODBC application recognizes.You can solve this problem by setting LongDataCompat = on in odbc.ini. If you call SQLColAttribute() in LOB column for this option, ODBC returns SQL_LONGVINARYto SQL_BLOB and SQL_LONGVARCHAR to SQL_CLOB relatively.

LOB Use Examples in PHP Program

The following is the examples using LOB in PHP application. You may check 2 properties as follows in php.ini before executing programs.

```
odbc.defaultlrl = 4096 (This value must be specified as greater than 1)
odbc.defaultbinmode = 0 (You must specify this as 0 for using LOB because
this can be executed without allocating additional memory.)
```

~/.odbc.ini is as follows.

```
[Altibase]
Driver = AltibaseODBCDriver
Description = Altibase DSN
ServerType = Altibase
UserName = SYS
Password = MANAGER
Server = 127.0.0.1
Port = 20073
LongDataCompat = on
NLS_USE = US7ASCII
```

php program

```
<?
/*
* ==================================================
* Connection Trial
* ==================================================
*/
$Connection = @odbc_connect("Altibase", "SYS", "MANAGER");
```

```
if (!$Connection)
{
echo "ConnectFail!!!\n";
exit;
}

/*
* ==================================================
* Table Creation
* ==================================================
*/

@odbc_exec($Connection, "DROP TABLE T2 ");
if (!@odbc_exec($Connection,
"CREATE TABLE T2 (I1 INTEGER, B2 BLOB, C3 CLOB) TABLESPACE SYS_TBS_DATA"))
{
echo "create test table Fail!!!\n";
exit;
}

/*
* ==================================================
* autocommit off for using LOB
* ==================================================
*/

odbc_autocommit($Connection, FALSE);

/*
* ==================================================
* Data Insertion
* ==================================================
*/

$query = "INSERT INTO T2 VALUES (?, ?, ?)";
$Result1 = @odbc_prepare($Connection, $query);
if (!$Result1)
{
$msg = odbc_errormsg($Connection);
echo "prepare insert: $msg\n";
exit;
}

for ($i = 0; $i < 10; $i++)
{

/*
* ----------------------
* Reading in File
* ----------------------
*/

$fileno2 = $i + 1;
$filename2 = "a$fileno2.txt";
print("filename = $filename2\n");
$fp = fopen($filename2, "r");
$blob = fread($fp, 1000000);
fclose($fp);

$fileno3 = 10 - $i;
$filename3 = "a$fileno3.txt";
print("filename = $filename3\n");
$fp = fopen($filename3, "r");
$clob = fread($fp, 1000000);
fclose($fp);
```

```
/*
* ----------------------
* INSERT
* ----------------------
*/

$Result2 = @odbc_execute($Result1, array($i, $blob, $clob));

print("inserting $i ,$filename2 and $filename3 into T2 ......... ");

if (!$Result2)
{
print("FAIL\n");
$msg = odbc_errormsg($Connection);
echo "execute insert: $msg \n";
exit;
}

print("OK\n");
}

/*
* ==================================================
* COMMIT
* ==================================================
*/

odbc_commit($Connection);

/*
* ==================================================
* Check inserted data
* ==================================================
*/
print "\n\n";
print "==========================================\n";
print "Selecting from table\n";
print "==========================================\n";

$query = "select * from t2";
$Result1 = @odbc_exec($Connection, $query);
if (!$Result1)
{
$msg = odbc_errormsg($Connection);
echo "ERROR select: $msg\n";
exit;
}

$rownumber = 0;
while (odbc_fetch_row($Result1))
{
$data1 = odbc_result($Result1, 1);
$data2 = odbc_result($Result1, 2);
$data3 = odbc_result($Result1, 3);
$len2 = strlen($data2);
$len3 = strlen($data3);

print "\n==========================================\n";
print "Row $rownumber....\n";
$rownumber++;
print "data1 = ".$data1."\n";
print "-------\n";
print "data2 = \n";
// print $data2; // Output is omitted because this is binary data.
```

```
print "\n";
print "dataLen2 = [$len2]\n";
print "-------\n";
print "data3 = \n";
print $data3;
print "\n";
print "dataLen3 = [$len3]\n";
}

odbc_commit($Connection);

@odbc_close($Connection);
?>
```

Other Changes

This section describes changes except data types.

SQLCloseCursor

You can call functions in the following order because there is not ODBC state machine in CLI library of ALTIBASE HDB 4.

```
SQLHSTMT stmt;
SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
SQLPrepare(stmt, (SQLCHAR *)”SELECT I1 FROM T1”, SQL_NTS);

SQLExecute(stmt);
SQLFetch(stmt);
SQLExecute(stmt);
```

However, if you execute codes above with ODBC-CLI in ALTIBASE HDB 5, function sequence error occurs in SQLExecute(stmt). Because stmt performing SQLExecute() first indicates to generate result set. Therefore, ODBC cursor becomes open and state of stmt indicates S5. (Refer to MSDN ODBC specification.). However, error occurs in this state cause of no performing SQLExecute().

If you want to perform SQLExecute() again, call SQLCloseCursor() clearly as follows and then make stmt have S1 or S3 state.

```
SQLExecute(stmt);
SQLFetch(stmt);
SQLCloseCursor(stmt);
SQLExecute(stmt);
```

SQLBindParameter - ColumnSize Argument

ColumnSize of SQLBindParameter() as the 6th parameter in ALTIBASE HDB 5 is different from that of previous one. If you insert 0 into this argument for previous version, no problem. However, if you insert maximum length of data transmitted to server in ALTIBASE HDB 5, its performance has problem because it checks their information whenever executed.

SQLBindParameter – StrLen_or_IndPtr Argument

CLI library in ALTIBASE HDB 4 references data only if they, which StrLen_or_IndPtr argument indicates, have variable length. However, ALTIBASE HDB 5 references the values in the memory StrLen_or_IndPtr argument indicates whenever performing SQLExecute() or SQLExecDirect() because ALTIBASE HDB 5 can implement SQLPutData() and SQLParamData().

Therefore, you need special care in perfectly initializing memory the pointer indicates if sending StrLen_or_Ind as the last argument of SQLBindParameter() to not Null pointer but valid pointer variables.

If SQL_DATA_AT_EXEC is -2 as constant number or SQL_LEN_DATA_AT_EXEC() is less than -100 without initializing memory completely, CLI library judges user intends to send the argument with SQLPutData(). And CLI library returns SQL_NEED_DATA when you call SQLExecute().

If SQLExecDirect() returns the unintended value(SQL_NEED_DATA) cause of no initialized value above, this influences on functions called next. So you need special care that function sequence errors in all functions called next cause to return SQL_ERROR.

SQLPutData(), SQLParamData()

ODBC-CLI in ALTIBASE HDB 5 supports SQLPutData() and SQLParamData() provided not in previous version. Refer to MSDN for details about each function. The following is the example program with using functions and StrLen_or_IndPtr mentioned above.

```
Table Schema :
CREATE TABLE T2_CHAR (I1 INTEGER, I2 CHAR(50));

void putdata_test(void)
{
SQLRETURN sRetCode;
SQLHANDLE sStmt;

SQLINTEGER i1;
SQLLEN i1ind;

SQLCHAR *i2[10] =
{
(unsigned char *)"0000000000000.",
(unsigned char *)"1111111111111. test has been done.",
(unsigned char *)"2222222222222. Abra ca dabra",
(unsigned char *)"3333333333333. Short accounts make long friends.",
(unsigned char *)"4444444444444. Whar the hell are you doing man!",
(unsigned char *)"5555555555555. Oops! I missed this row. What an idiot!",
(unsigned char *)"6666666666666. SQLPutData test is well under way.",
(unsigned char *)"7777777777777. The length of this line is well over 50
                                 characters.",
(unsigned char *)"8888888888888. Hehehe",
(unsigned char *)"9999999999999. Can you see this?",
};

SQLLEN i2ind;

SQLINTEGER i;

SQLINTEGER sMarker = 0;

i1ind = SQL_DATA_AT_EXEC;
i2ind = SQL_DATA_AT_EXEC;
```

```
sRetCode = SQLAllocHandle(SQL_HANDLE_STMT, gHdbc, &sStmt);
check_error(SQL_HANDLE_DBC, gHdbc, "STMT handle allocation", sRetCode);

sRetCode = SQLBindParameter(sStmt, 1, SQL_PARAM_INPUT,
SQL_C_SLONG, SQL_INTEGER,
0, 0, (SQLPOINTER *)1, 0, &i1ind);

sRetCode = SQLBindParameter(sStmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_CHAR,
60, 0, (SQLPOINTER *)2, 0, &i2ind);

sRetCode = SQLPrepare(sStmt,
(SQLCHAR *)"insert into t2_char values (?, ?)", SQL_NTS);

for(i = 0; i < 10; i++)
{
i1 = i + 1000;

printf("\n");
printf(">>>>>>>> row %d : inserting %d, \"%s\"\n", i, i1, i2[i]);
sRetCode = SQLExecute(sStmt);

if(sRetCode == SQL_NEED_DATA)
{
sRetCode = SQLParamData(sStmt, (void **)&sMarker);

while(sRetCode == SQL_NEED_DATA)
{
printf("SQLParamData gave : %d\n", sMarker);

if(sMarker == 1)
{
sRetCode = SQLPutData(sStmt, &i1, 0);
}
else if(sMarker == 2)
{
int unitsize = 20;
int size;
int pos;
int len;

len = strlen((char *)(i2[i]));
for(pos = 0; pos < len;)
{
size = len - pos;
if(unitsize < size)
{
size = unitsize;
}
sRetCode = SQLPutData(sStmt, i2[i] + pos, size);|

pos += size;
}
}
else
{
printf("bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb!!! unknown marker value\n");
exit(1);
}

sRetCode = SQLParamData(sStmt, (void **)&sMarker);
}
}
}
```

```
sRetCode = SQLFreeHandle(SQL_HANDLE_STMT, sStmt);
check_error(SQL_HANDLE_DBC, gHdbc, "STMT free", sRetCode);
}
```

Limitation on ALTER SESSION statements

ALTIBASE HDB 4 specifies AUTOCOMMIT MODE and DEFAULT_DATE_FORMAT as session properties as follows.

```
SQLExecDirect(stmt,
“ALTER SESSION SET AUTOCOMMIT=FALSE”,
SQL_NTS);

SQLExecDirect(stmt,
“ALTER SESSION SET DEFAULT_DATE_FORMAT='YYYY/MM/DD'",
SQL_NTS);
```

2 session properties above must have information on ODBC-CLI because they definitely affect conversion of ODBC-CLI and operation of functions related to transactions. However, ODBC-CLI can’t know property changes if transmitting SQL syntaxes to server directly with using SQLExecDirect(). ODBC-CLI can get information from server to know the values of property, but this causes to have worse performance. In ALTIBASE HDB 5 the property is modified with SQLSetConnectAttr() to solve this problem. ALTIBASE HDB 5 always makes the property in ODBC-CLI same as that in server. ALTIBASE HDB executes as follows when using ODBC-CLI.

```
SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_OFF,
0);
SQLSetConnectAttr(conn,
ALTIBASE_DATE_FORMAT,
(SQLPOINTER)"YYYY/MM/DD",
SQL_NTS);
```

SQLRowCount(), SQLMoreResults() functions

There are 2 results of ODBC.

- Number of affected rows
- Result set

ODBC-CLI considers multiple results in ALTIBASE HDB 5. In other words, you can get them by one execution. Therefore, returned results of SQLRoewCount() are different from those of ALTIBASE HDB 4 when you use array binding.

- SQLRowCount() : gets affected row count in “current” result.
- SQLMoreResults() : moves to “next” result and returns SQL_NO_DATA if current result is last.

Example

```
CREATE TABLE T1 (I1 INTEGER);
INSERT INTO T1 VALUES(1);
INSERT INTO T1 VALUES(2); ........ repeat 1000 times
```

```
SELECT * FROM T1;
T1
-----
1
2
3
.
.
.
1000
-----

SQLINTEGER p1[3];
SQLINTEGER p2[3];
SQLLEN rowcount = 0L;
SQLLEN totalRowcount = 0L;

p1[0] = 10; p2[0] = 20;
p1[1] = 100; p2[1] = 200;
p1[2] = 11; p2[2] = 14;

SQLSetStmtAttr(stmt, SQL_ATTR_PARAMSET_SIZE, 3); // <--- array binding
SQLBindParameter(stmt, 1, p1 ..);
SQLBindParameter(stmt, 2, p2 ..);

SQLExecDirect(stmt,
(SQLCHAR *)"DELETE FROM T1 WHERE I1>? AND I1<?”,
SQL_NTS);

do {
SQLRowCount(stmt, &rowcount);
printf("%d\n", rowcount);
totalRowcount += rowcount;
rc = SQLMoreResults(stmt);
} while (rc != SQL_NO_DATA);

printf(“totalRowcount = %d\n”, totalRowCount);
```

Execution Results

```
9 => This is affected row count of DELETE FROM T1 WHERE I1>10 AND I1<20
199 => This is affected row count of DELETE FROM T1 WHERE I1>100 AND I1<200
0 => This is affected row count of DELETE FROM T1 WHERE I1>11 AND I1<14
     (No record exists because it is deleted by the first execution.)
208 => This is the total of affected row counts
```

Each execution result of syntax the argument indicates is created, and then sent to ODBC-CLI. When multiple results are created like this, each data can move for next result with SQLMoreResults() and have its result with SQLRowCount(). If you sum up 3results above in ALTIBASE HDB 4, SQLRowCount() returns 208. If you want same results in ALTIBASE HDB 5 as in ALTIBASE HDB 4, you may execute SQLMoreResults() repeatedly until it returns SQL_NO_DATA, and then add this result to result of SQLRowCount().

Unlimited Array Execute, Array Fetch

ALTIBASE HDB does not have restrictions on Array Execute and Array Fetch as buffer size. Therefore, you can bind array in the allocated memory and can use CM_BUFF_SIZE no more.

Unavailable Properties

Batch Processing Mode

You can’t use batch keyword of connection string and SQL_ATTR_BATCH.

SQL_ATTR_MAX_ROWS

This indicates to specify the number of prefetched row for better performance in ALTIBASE HDB 4.

However, this property is similar to LIMIT of SELECT statement following ODBC. This option is not available for ODBC-CLI of ALTIBASE HDB. Therefore, if you specify property above as SQLSetStmtAttr(), this asks your attention because error occurs like ‘Optional feature not implemented’.

Index