Developing an ODBC C++ Client with MySQL Database

Author: Rajinder Yadav
Date: Aug 21, 2007

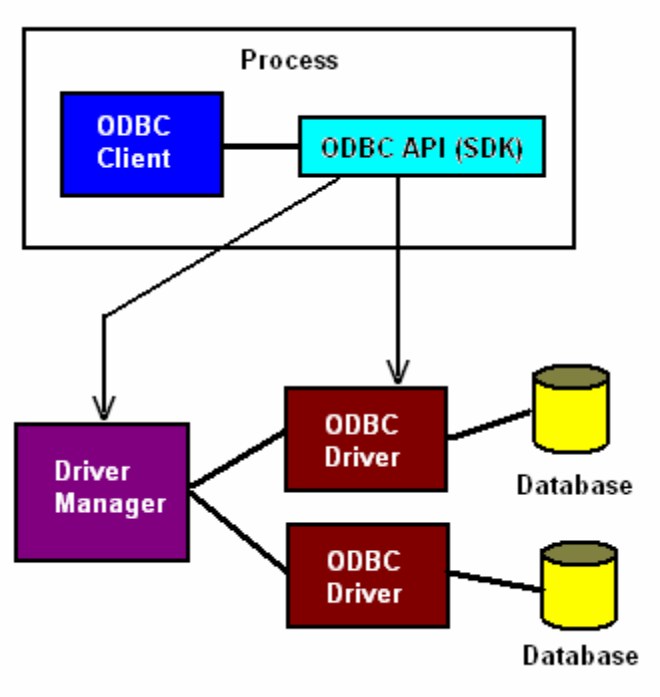
Web: http://devmentor.org
Email: rajinder@devmentor.org

Assumptions

I am going to assume you already know how to code in C++ and have worked with a database before and understand some SQL. Although we are not using any SQL in the development of our front end client, it will be helpful and required for anyone wanting to learn C++ ODBC database development. You will also need to know how to program in MFC, but I will start off introducing the basic concepts of using a simple console application before trying things up.

What is ODBC

Open Database Connectivity (ODBC) is a Microsoft born technology that is fast becoming an industry standard and is supported by almost every database vendor. There are two layers you must be aware of, the ODBC Driver and the ODBC API interface which is based on the Call Level Interface specification of the SQL Access Group. We will be working with the ODBC API to interact with a vendor specific ODBC driver what will be acting as a bridge to communicate with the vendor's database.



ODBC allows the developer to interact with various databases through a set of APIs making life easier and uses Structured Query Language (SQL) as its database access language. ODBC is designed for interoperability , allowing a client application to be developed once to access different Database Management Systems (DBMS) without code changes.

What you will need
You will need a SQL database to work with. If you don't have one installed on your PC there are many to choose from. The one I found that's free and easy to install, setup and get going with was **MySQL (www.mysql.org)**, you can download their community version. If you do, make sure to also get the ODBC driver, plus they also have separate downloads for their **GUI Tools**. I would recommend you get the at least the following two GUI Tools as it will make it easier to administrate and work with the database.

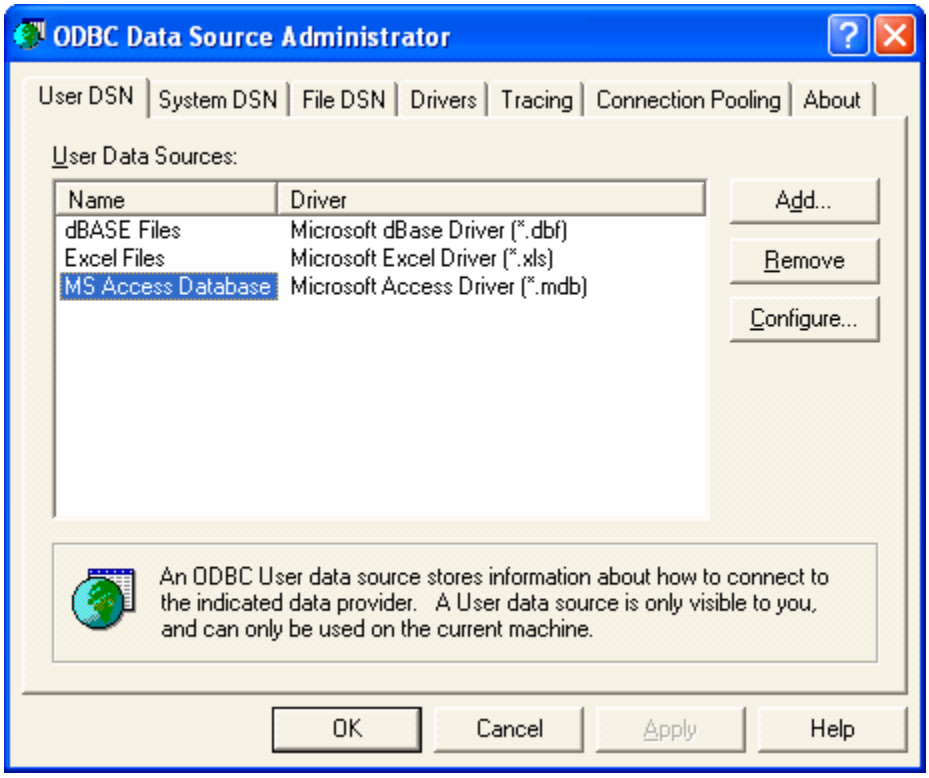
MySQL Administrator
MySQL Query Browser

As of this writing, the **ODBC driver download** can be found under their "Connectors" download area.

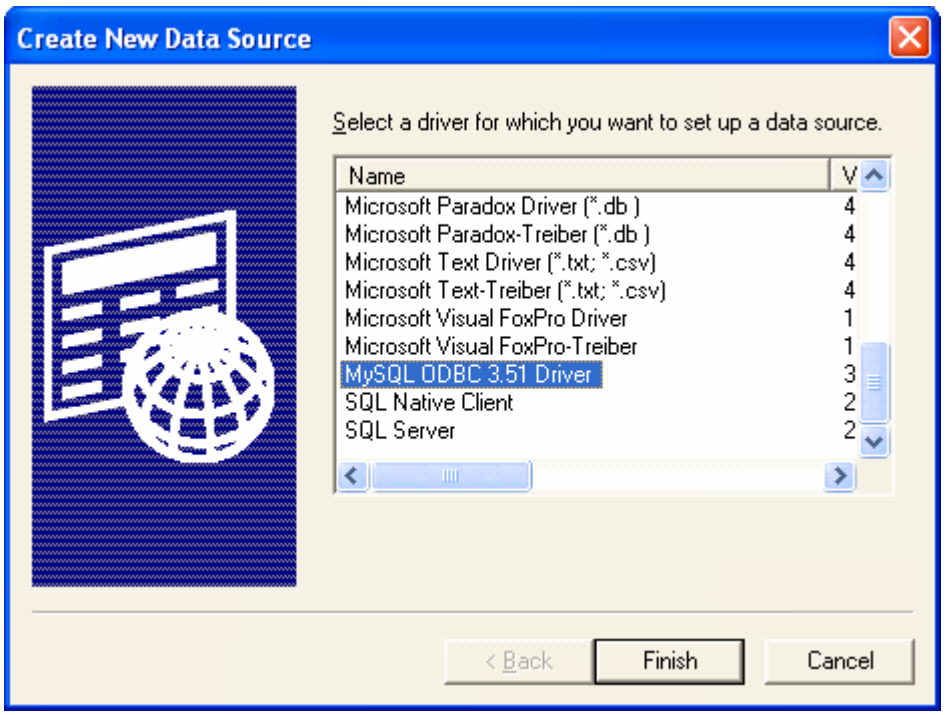
The other thing you will need to have is **Microsoft Platform SDK** installed.

Setting up the Data Source

Before you can connect to a database, a datasource need to be setup. To do this, go to the Administrative menu and select "Data Source (ODBC)". This will bring up the following dialog box.



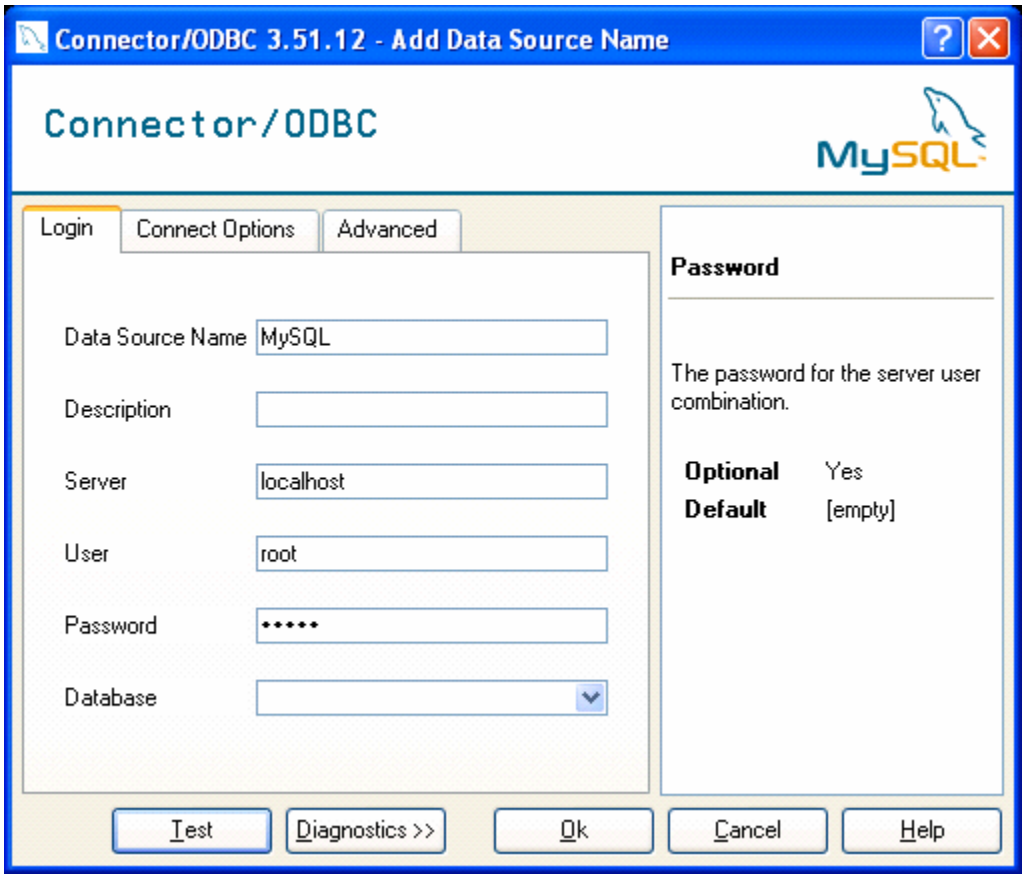
Click on the "Add" button and scroll down till you see the driver for your database, in our case it's "MySQL ODBC Driver".



Click "Finish" and fill in the required field in the "Connector/ODBC" dialog box, such as the user id and password. The user id will be 'root' by default and whatever admin password you might have set during the database install & setup process. The "Data Source Name" field is the connection string you will pass when connecting to the ODBC datasource in your code, this field is user defined and called anything.

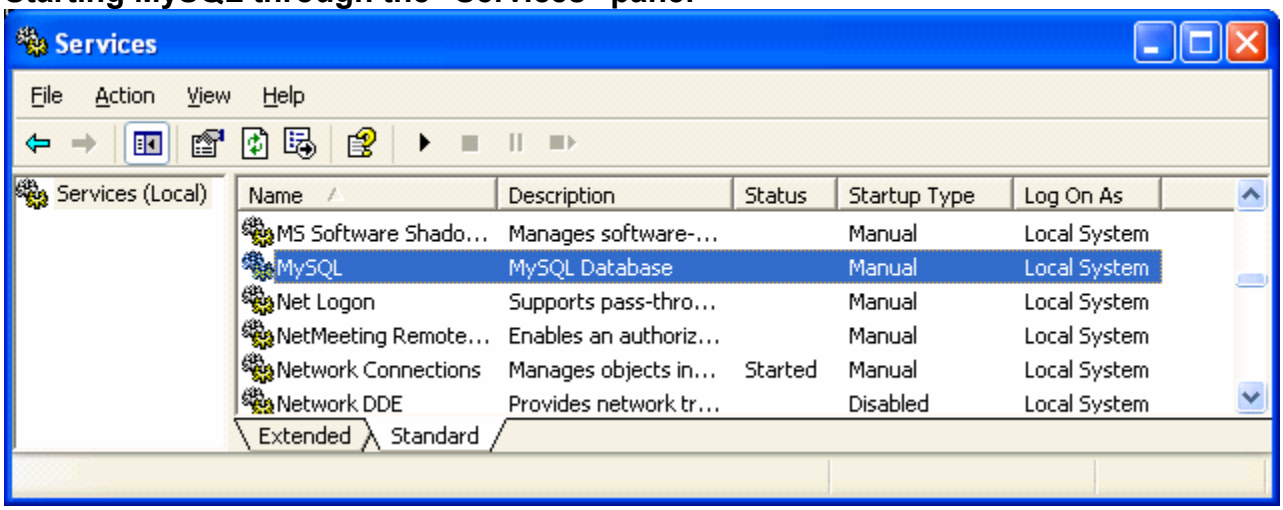
The other filed here are optional, but the best was to get a feel for what is required is to try to connect to a database using the "MySQL Query Browser" GUI Tool, and use the same connection parameters when setting up the datasource.

Note: there is a "Test" button which can be used to confirm you've entered the correct field values.



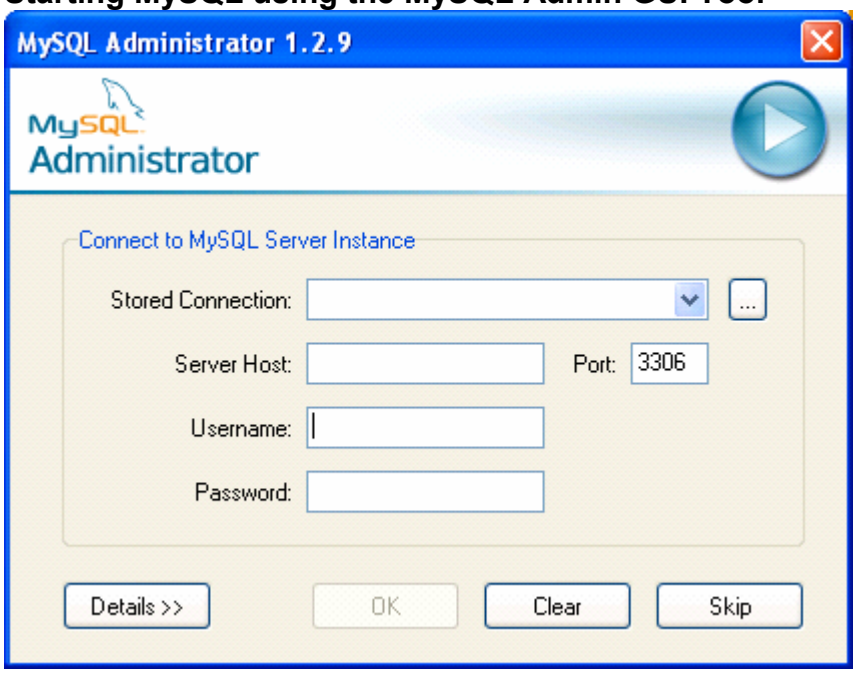
Also make sure the database has been started and is running, you can check this either through the System Service control panel, or with MySQL Admin GUI Tool.

Starting MySQL through the "Services" panel



From the service panel, read the "Status" panel to see if MySQL database server was started. To start the service, right click on "MySQL" and select "Start" from the context menu, or hit the "Run" button.

Starting MySQL using the MySQL Admin GUI Tool



When you see this login Dialog, click on the "CTRL" key, this will change the "Cancel" button to "Skip", you can't login right now since the database may not be running.

Click on the "Skip" button to open the MySQL Admin dialog box Start/Stop Service panel, click on the "Start Service" to make the database server operational. Look in the "Log Messages" status window to see that the server was started.

Compiling
The header files you will need to include are **<sql.h>**, **<sqlext.h>**, and **<sqltypes.h>** also make sure you are linking to the following libraries if you're getting ODBC linker errors

odbc32.lib
odbccp32.lib

sql.h header file contains function prototypes for the core ODBC interface conformance level

sqlext.h header files contains function prototypes for level 1 & 2 API

sqltypes.h header files contains SQL data type definitions

Handles
You will be working with handles that are 32-bits values, there are 4 handle types you need to know, there are:

Table 1 - ODBC Handles

HANDLE	HANDLE TYPE	DESCRIPTION
SQLHENV	Environment	Global context in which to access data
SQLHDBC	Connection	Identifies a connection (driver, datasource), state and settings
SQLHSTMT	Statement	A SQL statement and any associated result set
SQLHDESC	Descriptor	Metadata collection used to describe a SQL statement, columns, result set

These handles are allocated using the **SQLAllocHandle( )** API and released using the **SQLFreeHandle( )** API.

SQLAllocHandle( )

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

Arugments

```
[input] HandleTypes - defined types are:

        SQL_HANDLE_ENV
        SQL_HANDLE_DBC
        SQL_HANDLE_DESC
        SQL_HANDLE_STMT

[input] InputHandle - context handle that the allocated handle will belong to. For
instance a database handle belongs to a environment handle. Also the order of the handle
allocation is important, the Environment handle is allocated first, with a input context
handle of SQL_NULL_HANDLE.

[output] OutputHandlePtr - Allocated SQL handle
```

Given an environment handle, we can go ahead and allocate the database handle, the environment handle will be input handle as the InputHandle parameter to **SQLAllocHandle( )**.

Once a database handle is allocated, we allocate the SQL statement handle using the database handle as the InputHandle, since all SQL statement handles will belong to a database they are executed on.

Likewise a descriptor handle allocation will be based on a SQL statement handle, since it provides the description of the result set elements.

This will make more sense when you see the code.

Error Handling

The ODBC SQL APIs return an error code of type SQLRETURN which is an integer value. The returned error will be specific to the ODBC API but the general one will be **SQL_SUCCESS**, **SQL_SUCCESS_WITH_INFO** or **SQL_ERROR**.

You can use the **SQL_SUCCESS(err)** macro to test if the API call succeeded.

For a partial success when the SQL return code is **SQL_SUCCESS_WITH_INFO** or an error, a call made to **SQLGetDiagRec( )** will allow you to obtain the associated **SQLSTATE** value. Refer to the DOC API reference for more information.

Connecting to a Database
All ODBC application will have to properly set up the ODBC environment before connecting to a database. The code below allocates an environment handle and then set up the environment to use ODBC version 3.

```
SQLHENV   sql_hEnv  = 0;  // environment handle

SQLAllocHandle( SQL_HANDLE_ENV,         // type to allocate
                SQL_NULL_HANDLE,        // context handle
                &sql_hEnv );            // handle to be allocated

SQLSetEnvAttr( sql_hEnv,
               SQL_ATTR_ODBC_VERSION,
               (void*) SQL_OV_ODBC3,
               0 );
```

The next step is to allocate the database connection handle based on the environment handle that will be used to make a connection to the datasource.

```
SQLHDBC sql_hDBC  = 0;

SQLAllocHandle( SQL_HANDLE_DBC,    // type to allocate
                sql_hEnv,          // in context of environ handle
                &sql_hDBC );       // handle to be allocated
```

Once we have allocated a database handle we can make a connection to our datasource using **SQLConnect( )** or **SQLDriverConnect( )**.

The simplest way to connect is to use **SQLConnect( )**, it's good for applications that hard-code a datasource name. The following code would connect to a datasource named "MySQL" with the user "root" having a password of "admin".

```
SQLConnect( sql_hDBC,
            (SQLCHAR*) "MySQL",       // datasource name
            SQL_NTS,
            (SQLCHAR*) "root",        // user id
            SQL_NTS,
            (SQLCHAR*) "admin",       // password
            SQL_NTS );
```

To connect using **SQLDriverConnect( )** the call would be made as shown.

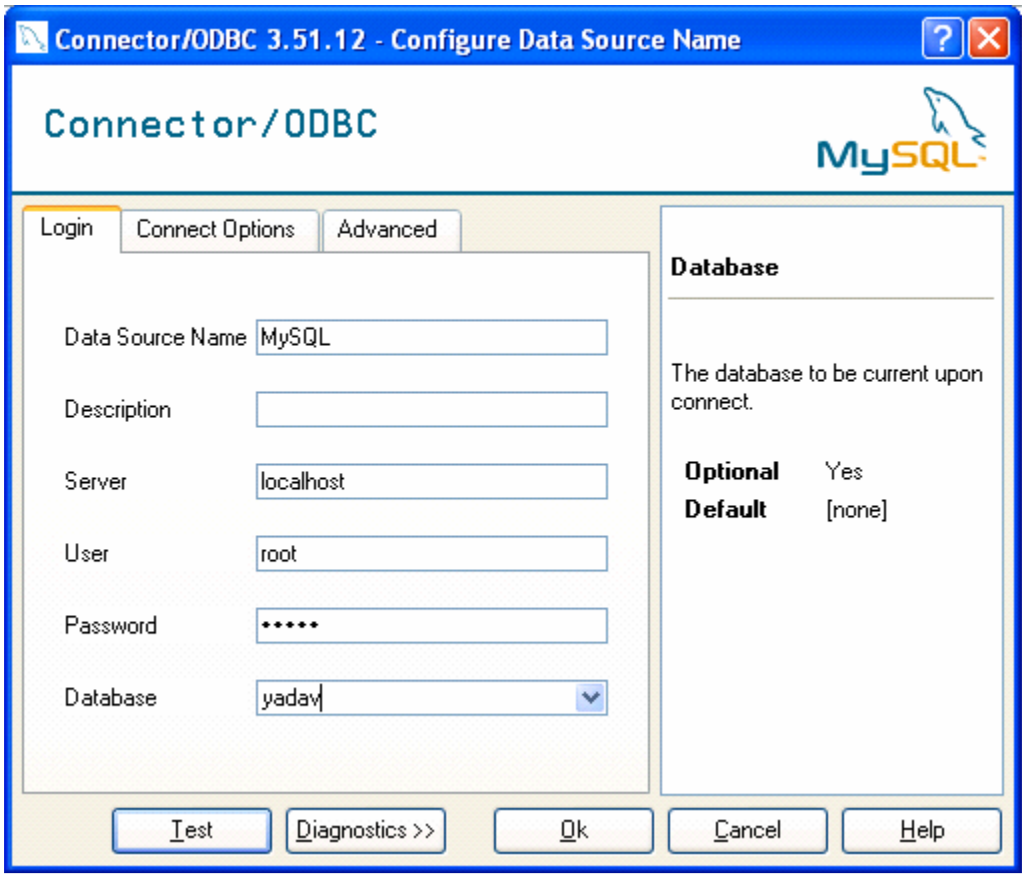
```
SQLDriverConnect( sql_hDBC,
                  0,
                  (SQLCHAR*)"DSN=MySQL;UID=root;PWD=admin;",
                  SQL_NTS,
                  szDNS,
                  1024,
                  &nSize,
                  SQL_DRIVER_COMPLETE );
```

Note: the construction of the connection string

```
"DSN=MySQL;UID=root;PWD=admin;"
```

The UID (User ID) and PWD (password) are optional if they were already provided when setting up the ODBC datasource, otherwise they are required or the connection will fail.

The '**database**' field in another optional field that can be defined in the connection string if the database connecting to was defined in the ODBC datasource, otherwise it will be required.



Below is the complete connection string that is return in the out parameter szDNS after the call to **SQLDriverConnect**. This string can be reused to reestablish a connection in the future if the User was prompted with a dialog box to enter additional information.

```
"DATABASE=yadav;DSN=MySQL;OPTION=0;PWD=admin;PORT=0;SERVER=localhost;UID=root"
```

All connections must be closed with a call to SQLDisconnect( )

The general API for SQLConnect is shown below. Note: for null terminated strings we can pass SQL_NTS as the parameter value for string length.

SQLConnect( )

```
SQLRETURN SQLConnect( SQLHDBC     ConnectionHandle,
                      SQLCHAR *   ServerName,
                      SQLSMALLINT NameLength1,
                      SQLCHAR *   UserName,
                      SQLSMALLINT NameLength2,
                      SQLCHAR *   Authentication,
                      SQLSMALLINT NameLength3 );
```

Arguments

[Input] **ConnectionHandle** - Connection handle.

[Input] **ServerName** - Data source name. The data might be located on the same computer as the program, or on another computer somewhere on a network. For information about how an application chooses a data source, see Choosing a Data Source or Driver.

[Input] **NameLength1** - Length of *ServerName.

[Input] **UserName** - User identifier.

[Input] **NameLength2** - Length of *UserName.

[Input] **Authentication** - Authentication string (typically the password).

[Input] **NameLength3** - Length of *Authentication.

SQLDriverConnect( )

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

Arguments (From MSDN)

[Input] ConnectionHandle - Connection handle.

[Input] WindowHandle - Window handle. The application can pass the handle of the parent window, if applicable, or a null pointer if either the window handle is not applicable or SQLDriverConnect will not present any dialog boxes.

[Input] InConnectionString - A full connection string (see the syntax in "Comments"), a partial connection string, or an empty string.

[Input] StringLength1 - Length of *InConnectionString, in characters.

[Output] OutConnectionString - Pointer to a buffer for the completed connection string. Upon successful connection to the target data source, this buffer contains the completed connection string. Applications should allocate at least 1,024 bytes for this buffer.

[Input] BufferLength - Length of the *OutConnectionString buffer. If the *OutConnectionString value is a Unicode string (when calling SQLDriverConnectW), the BufferLength argument must be an even number.

[Output] StringLength2Ptr - Pointer to a buffer in which to return the total number of characters (excluding the null-termination character) available to return in *OutConnectionString. If the number of characters available to return is greater than or equal to BufferLength, the completed connection string in *OutConnectionString is

```
truncated to BufferLength minus the length of a null-termination character.

[Input] DriverCompletion - Flag that indicates whether the Driver Manager or driver must
prompt for more connection information:

SQL_DRIVER_PROMPT
SQL_DRIVER_COMPLETE
SQL_DRIVER_COMPLETE_REQUIRED
SQL_DRIVER_NOPROMPT
```

Once a database connection has been established, to run a SQL statement use either **SQLPrepare** / **SQLExecute** or **SQLExecDirect.**

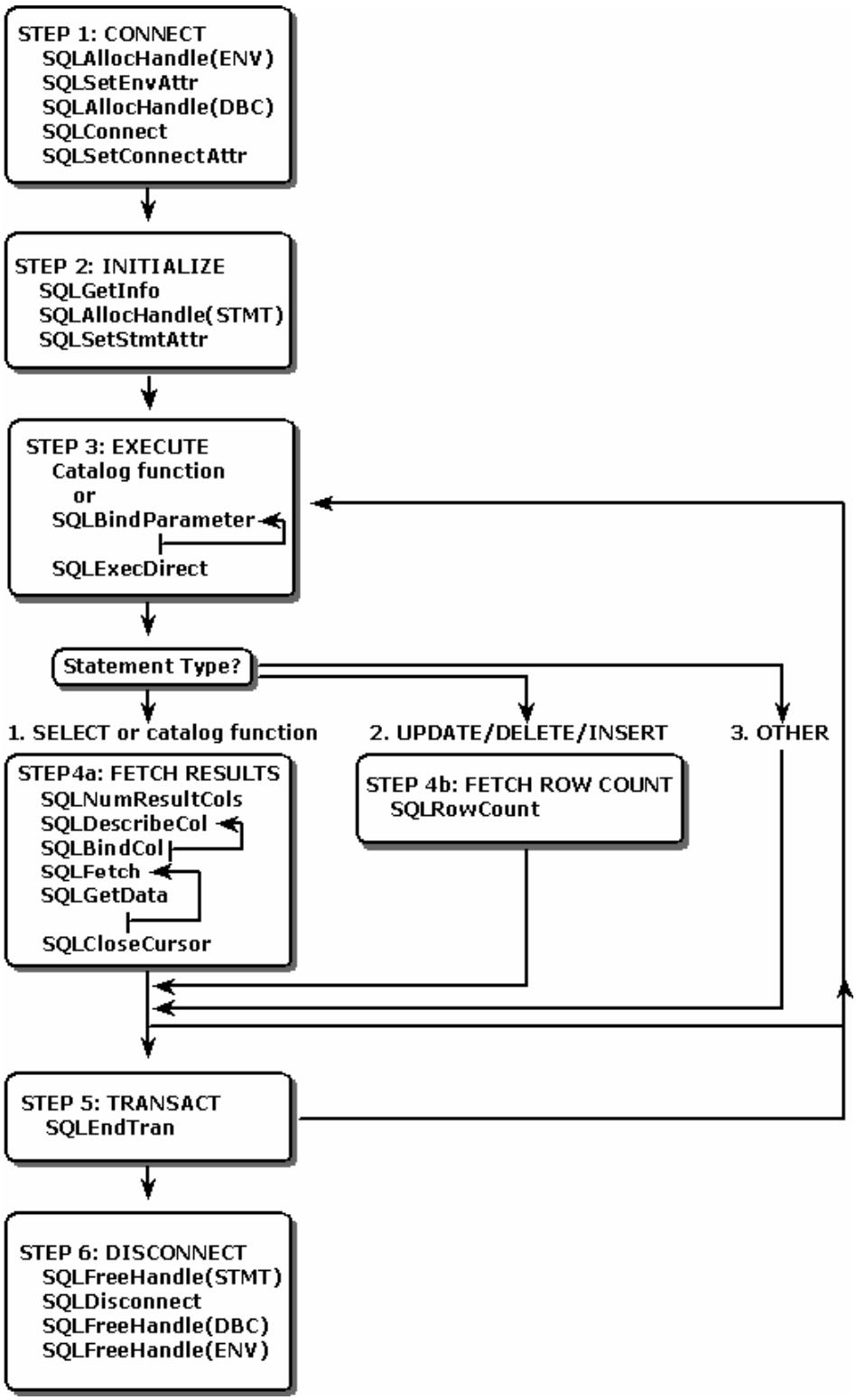
If you want to reissues the same SQL statement with differing values, you will use **SQLPrepare** and **SQLExecute** APIs. For static SQL statements you should use the **SQLExecDirect** API.

The following outlines the sequence of steps required to use a database.

1. Allocate an ODBC environment handles
2. Set the ODBC environment to use the an ODBC driver
3. Allocate a database handle
4. Allocate a SQL statement handle
5. Prepare the SQL statement
6. Execute the SQL statement
7. Read the data from the result set

The following page shows a state chart of the steps required by an application to work with an ODBC database.

Basic ODBC Application Steps



Flowchart obtained from MSDN

```
// MySQLSelect.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <sql.h>
#include <sqlext.h>
#include <sqltypes.h>
#include <iostream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{

   SQLHENV   sql_hEnv  = 0;
   SQLHDBC   sql_hDBC  = 0;
   SQLHSTMT  sql_hStmt = 0;

   SQLCHAR szDNS[1024] ={0};
   SQLSMALLINT nSize = 0;

   SQLRETURN sqlRet =
   SQLAllocHandle( SQL_HANDLE_ENV,
                   SQL_NULL_HANDLE,
                   &sql_hEnv );

   sqlRet =
   SQLSetEnvAttr( sql_hEnv,
                  SQL_ATTR_ODBC_VERSION,
                  (void*) SQL_OV_ODBC3,
                  0 );

   sqlRet =
   SQLAllocHandle( SQL_HANDLE_DBC,
                   sql_hEnv,
                   &sql_hDBC );


// if 0 - use SQLDriverConnect
// if 1 - use SQLConnect
//
#if 0
   sqlRet =
   SQLConnect( sql_hDBC,
               (SQLCHAR*)"MySQL",
               SQL_NTS,
               (SQLCHAR*)"root",
               SQL_NTS,
               (SQLCHAR*)"admin",
               SQL_NTS );

#else
   sqlRet =
   SQLDriverConnect( sql_hDBC,
                     0,
                     (SQLCHAR*)"DSN=MySQL;UID=root;PWD=admin;",
                     SQL_NTS,
                     szDNS,
                     1024,
                     &nSize,
                     SQL_DRIVER_COMPLETE );
#endif

   if( SQL_SUCCEEDED( sqlRet ) )
   {
      cout << "Connected to database " << endl
           << "Connection Info: " << endl
           << szDNS << endl;
```

```
      sqlRet =
      SQLAllocHandle( SQL_HANDLE_STMT,
                      sql_hDBC,
                      &sql_hStmt );

      sqlRet =
      SQLExecDirect( sql_hStmt,
                     (SQLCHAR*)"SELECT * FROM yadav.friends;",
                     SQL_NTS );

      SQLSMALLINT nCols = 0;
      SQLINTEGER nRows = 0;
      SQLINTEGER nIdicator = 0;
      SQLCHAR buf[1024] = {0};

      SQLNumResultCols( sql_hStmt, &nCols );
      SQLRowCount( sql_hStmt, &nRows );

      while( SQL_SUCCEEDED( sqlRet = SQLFetch( sql_hStmt ) ) )
      {
         cout << "Row " << endl;
         for( int i=1; i <= nCols; ++i )
         {
            sqlRet = SQLGetData( sql_hStmt,
                                 i,
                                 SQL_C_CHAR,
                                 buf,
                                 1024,
                                 &nIdicator );

            if( SQL_SUCCEEDED( sqlRet ) )
            {
               cout << "Column " << buf << endl;
            }
         }
      } // while

      SQLFreeHandle( SQL_HANDLE_STMT, sql_hStmt );
      SQLDisconnect( sql_hDBC );
   }
   else
   {
      cout << "Failed to connect to the database" << endl;
   }

   SQLFreeHandle( SQL_HANDLE_DBC, sql_hDBC );
   SQLFreeHandle( SQL_HANDLE_ENV, sql_hEnv );

   return 0;
}
```