RECURSION SOFTWARE, Inc.

Database Toolkit: Portable and Cost Effective Software

By Katherine Ye

Recursion Software, Inc.

RECURSION SOFTWARE, Inc.

TABLE OF CONTENTS

Abstract 2
Why using ODBC 2
Disadvantage of ODBC 3
Programming with Database Toolkit 4
Benefits of using Database Toolkit 5
Conclusion 12

Database Toolkit: Portable and Cost Effective Software

By Katherine Ye

Abstract

Database programming is no easy task especially when it involves multiple DBMS spanning different platforms. Expectations are constantly running high. People are looking for ways to handle database processing that is portable, easy to use, yet not sacrificing powerful features of the underlying database. It also must have the flexibility to be customized for the special needs of the applications. This article provides an introduction to the Database Toolkit. It explores the benefit of using the Database Toolkit in a client server environment. It also covers examples of basic database operations using the Database Toolkit.

We are living in an information age. Our daily life involves absorbing useful information and filtering out garbage. Information (Data) plays an important role in our daily life. People, especially businesses, need to organize large amounts of disparate information. The information needs to be organized in such a way that you can easily access desired pieces of data quickly. One way to handle the information is to create a database to hold related data. But that is just the first step. It is typically more important to be able to get data out of the database whenever the needs arise. That is where ODBC comes into play.

Why using ODBC

ODBC stands for Open Database Connectivity. It is a widely used application programming interface for database access. ODBC is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs. It uses Structured Query Language (SQL) as its database access language.

ODBC is especially suitable for applications using several different database management systems (DBMS). ODBC is popular because it is independent of any one database management system or operating system. It provides an interface that is portable to multiple platforms and relational database management systems (RDBMS). A single application can be connected to different databases with no changes, re-compilation or re-linking required. At the same time, ODBC provides a powerful set of facilities for controlling and utilizing databases.

Despite all the advantages of ODBC, using ODBC can still be a non-trivial task.

Disadvantages of ODBC

1. Multiple installation of Driver Manager and Driver

In a client server environment, usually a Driver Manager and a series of ODBC drivers are installed on a server. A Driver Manager and an ODBC driver need to be installed on each client as well. This allows each client to access any data maintained on the server, but leads to increased cost for hardware and maintenance. Fig.1 illustrates the situation.

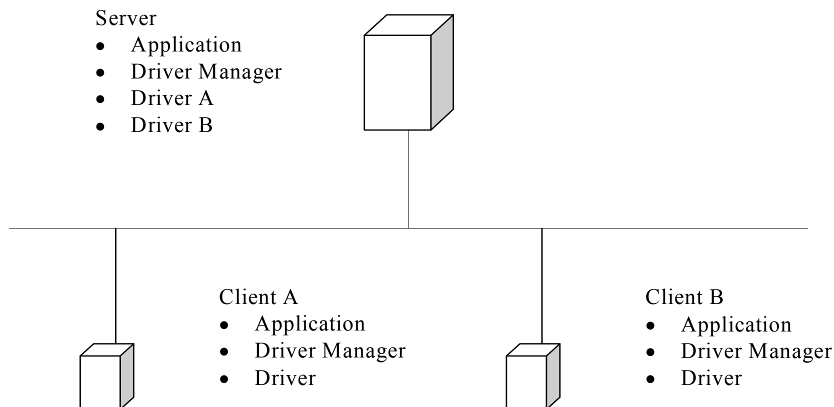

Fig. 1: Client access data from Server (ODBC)

2. Overhead of setup and tear down of database

There are several basic steps for using ODBC:

- Connection to Data Source.
    - Load the Driver Manger and allocate the environment handle
    - Register ODBC version
    - Allocate DBC handle. Also sets any application connection attributes.
- Initialize the Application
    - Find out the capability of driver through SQLGetInfo
- Prepare, Build and Execute query
    - Allocate statement handle and set statement attribute
    - Build and execute SQL statement
- Process the result
    - Usually involves binding an application variable to a column in the result set and retrieves the result

- Disconnection from the Data Source
    - Free statement handle
    - SQLDisconnection
    - Free DBC handle
    - Free environment handle

Here we illustrate the typical steps of programming with ODBC. In order for the application developers to work on ODBC, they need to do the setup such as allocating handles (environment, dbc, statement), setting desired attribute, performing the actual query and cleaning up which involves freeing the handles, disconnect, etc.

Programming with Recursion Software's Database Toolkit™

Recursion Software's Database Toolkit offers a nice solution for the above problems. First, it does not require an ODBC environment on the client machine. Second, it is portable and it simplifies the process of database programming. We will elaborate more details later. Now let us take a look of what is included in the Database Toolkit and what it can offer to application developers.

The Database Toolkit consists of two components, client and server. The server makes use of the native operating system's ODBC environment to respond to the requests of various database clients. The client exposes APIs to database applications. Fig. 2 demonstrates the architecture of the Database Toolkit.

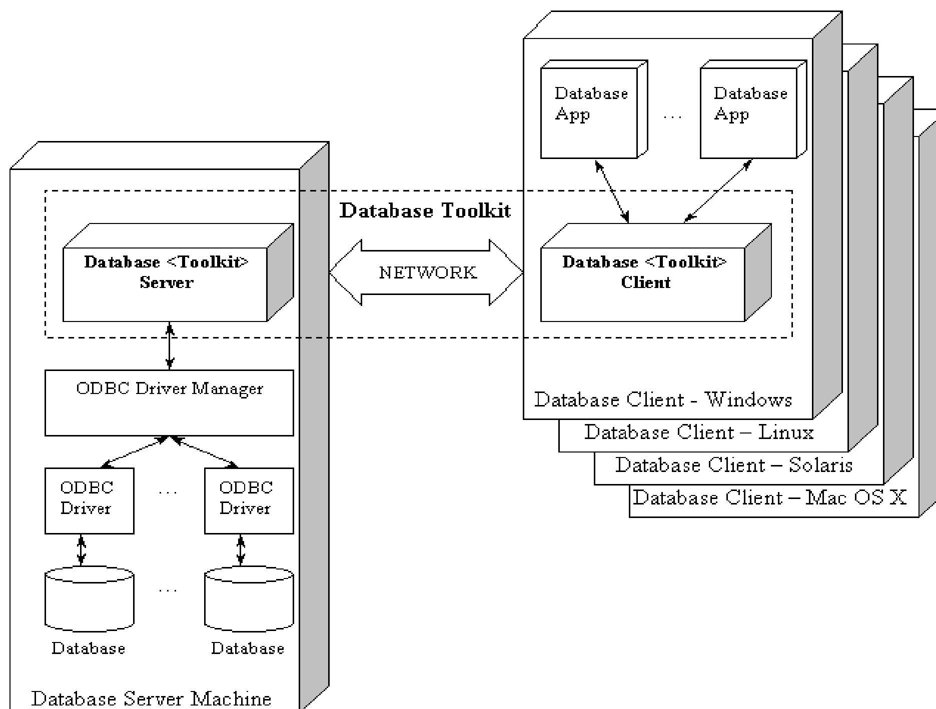

Fig. 2: Architecture of Database Toolkit

Benefits of using Database Toolkit

The Database Toolkit is a cross-platform C++ solution for accessing a wide range of ODBC-compliant databases. The purpose of the Database Toolkit is to reduce development costs. It achieves this goal by reducing the need to have multiple ODBC drivers for different databases installed on the client machines. It also reduces software configuration complexity by configuring and maintaining your ODBC drivers in just one location. Fig. 3 illustrates this scenario.

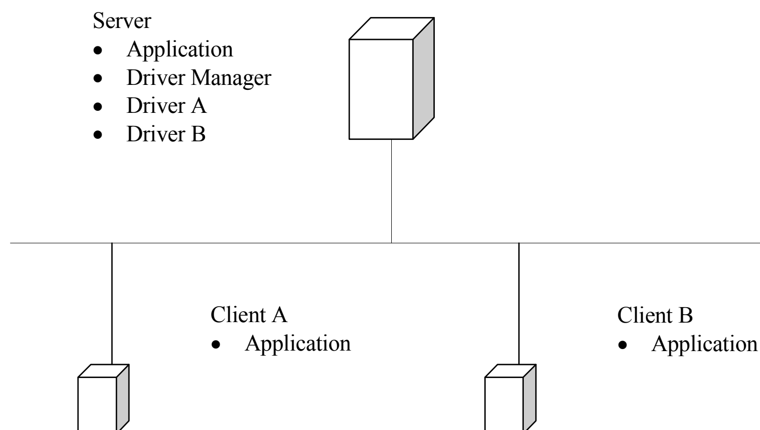

Fig. 3: Client access data from Server (Database Toolkit)

The Database Toolkit contains classes and methods exposing simple APIs for building portable database applications quickly. It eases the burden of dealing with the following issues: connection to the database, environment handle management, statement handle creation, execution and disconnecting from the database by placing highly repeated code in an easy to use API. There is no need for the application layer to do the basic set up and clean up routine. Since the Database Toolkit Connection class takes care of the necessary initialization, maintenance and clean up.

Using the Database Toolkit is easy. It typically has the following steps:

1. Library initialization
2. Connecting to Databases
3. Executing SQL Statement

Listing 1 is a complete example that illustrates these steps:

RECURSION SOFTWARE, Inc.

Listing 1: Database programming using Toolkit

```
#include <ospace/std/iostream>
#include <ospace/network.h>
#include <ospace/stream.h>
#include <ospace/database.h>


/**
 * Usage {program} [dbserver_hostname [dsn [user [auth]]]]
 * @param argc - Command line argument count
 * @param argv - Command line arguments
 */
int main(int argc, char** argv)
 {

 /*
  * Library initialization
  */
 os_init_network_toolkit();   // Initialize dependency
 os_init_streaming_toolkit(); // Initialize dependency
 os_init_database_toolkit();  // Initialize Database<ToolKit>

 /*
  * Connecting to the Database
  *
  * Establishing a database connection from the Database<Toolkit>
  * Client consists of connecting to the Database<Toolkit> Server
  * machine, identifying a Data Source Name (DSN) that an
  * administrator has configured within the remote machine's ODBC
  * environment, and providing any necessary authentication
  * information
  *
  */

 os_string target_hostname;
 os_string dsn;      // dsn name
 os_string user;     // user name
 os_string auth;     // auth info

 // Use local host if none specified on command line
 (argc > 1) ? target_hostname = argv[1] : target_hostname = os_host::my_host().name();
 (argc > 2) ? dsn  = argv[2] : dsn  = "dbtoolkit";
 (argc > 3) ? user = argv[3] : user = "guest";
 (argc > 4) ? auth = argv[4] : auth = "";

 try
   {
   os_ip_address iaddr( target_hostname );
   os_db_connection connection( os_socket_address(iaddr, 3006),
                                dsn,
                                user,
                                auth );
```

```
  /*
   * Executing SQL Statements
   *
   * 1. Construct an os_db_statement object for an SQL statement
   * 2. Attach the os_db_statement object to an existing database
   *    connection
   * 3. Execute the statement
   */

  os_db_statement stmt( "SELECT * FROM Supplier" );

  stmt.attach( connection );
  stmt.execute();

  const int NAME_LEN = 50+1;
  const int PHONE_LEN = 20+1;
  char supplier_name[NAME_LEN], supplier_phone[PHONE_LEN];
  unsigned long supplier_id;

  /*
   * Binding data buffers with bind() before calling fetch()
   * when a successful call to fetch() is made the data
   * will automatically be placed in all of the bound buffers
   *
   */

  stmt.bind(1, supplier_id);
  stmt.bind(2, (char *)supplier_name, sizeof(supplier_name));
  stmt.bind(9, (char *)supplier_phone, sizeof(supplier_phone));

  while (stmt.fetch())    // Get each row
   {
   os_stringstream oss;
   oss << supplier_id
       << ":"
       << supplier_name
       << ":"
       << supplier_phone;
   cout << oss.str() << endl;
   }
  }
catch ( os_db_error& error )
  {
  cout << "Caught database error:" << endl;
  cout << "\t" << error.description( error.get_code() ) << endl;
  cout << "\t" << error.what() << endl;
  cout << "Native Error Code: " << error.get_native() << endl;
  }
```

```
 catch ( os_streaming_toolkit_error& error )
   {
   cout << "Caught streaming error:" << endl;
   cout << "\t" << error.description( error.code() ) << endl;
   cout << "\t" << error.what() << endl;
   cout << "Native Error Code: " << error.native() << endl;
   }
  catch ( os_network_toolkit_error& error )
   {
   cout << "Caught network toolkit error:" << endl;
   cout << "\t" << error.description( error.code() ) << endl;
   cout << "\t" << error.what() << endl;
   cout << "Native Error Code: " << error.native() << endl;
   }

  return 0;
  }
```

[END LIST]

Listing 1 above illustrates just one variant of the connection mechanism that is offered by the Database Toolkit. We can also build a connection to the database server using another approach:

Listing 2: An alternative way of making database connection

```
// ...

        /*
         * Instead of using
         * os_db_connection connection1( os_socket_address( iaddr, 3006),    // target machine, port 3006
         *                               dsn,          // ODBC Data Source Name
         *                               user,         // user
         *                              auth );       // authentication
         * use a driver specific connection attribute
         *
         */
        os_string connect_info =
             "DSN=" + dsn +
             ";UID=" + user +
             ";PWD=" + auth +
             ";";

        os_db_connection connection2( os_socket_address(iaddr, 3006),
                              connect_info );

// ...
```

[END LIST]

Another advantage to the Database Toolkit is its portability. It behaves the same regardless of the underlying operating system. Programs that are developed using the Database Toolkit work on different platforms and compilers. Since there is no need to have an ODBC driver on the client platform, you can run the database client on platforms that do not support ODBC.

If we have a database server running on the Windows environment and a database client needs to access database info from a Linux box, with the help of the Database Toolkit, you can retrieve the data across different platforms without any code changes. All you need to do is provide an IP address of the server machine and use it as a parameter for os_ip_address iaddr. Following is the code snippet.

Listing 3: DB Client connect to a remote DB Server

```
// ...

   os_ip_address iaddr( "10.168.27.2" ); // where 10.168.27.2 is the IP of
                                         // Database Server
   os_db_connection connection( os_socket_address(iaddr, 3006),
                                dsn,
                                user,
                                auth );

// ...
```

[END LIST]

Sometimes we want to access or modify low-level remote ODBC environment or connection attributes before a database connection is made. The Database Toolkit provides this kind of capability as well. Please be aware that if no DSN is specified during construction of the `os_db_connection` object then a call to the object's `dsn_connect()` must be made before the connection can be used by `os_db_statement` objects. Listing 4 demonstrates how to construct an `os_db_connection` without initially specifying the database to connect to.

Listing 4: Retrieve env_handle and dbc_handle

```
// ...

   //
   // Build a connection to the database server, delaying DSN connection.
   //
   os_ip_address iaddr( target_hostname );
   os_db_connection connect( os_socket_address( iaddr, 3006) );

   OSSQLHANDLE env_handle = connect.get_env_handle();
   //
   // Can perform raw ODBC API calls on this env_handle now.
   // No dbc_handles exist yet for this env_handle.
   //
```

```
// ...

OSSQLHANDLE dbc_handle = connect.get_dbc_handle();
//
// Can perform raw ODBC API calls on this dbc_handle now.
// No DSN connection exists.
//

// ...

// Make the DSN connection
os_string connect_info =
  "DSN=" + dsn +
  ";UID=" + user +
  ";PWD=" + auth +
  ";";

if (connect.dsn_connect(connect_info))
  {
  cout << "connect succeeded" << endl;
  }

// ...
```

[END LIST]

In the Database Toolkit, the transaction control is handled on the os_db_connection class. All statement objects attached to a connection object are controlled by that connection object's transaction commit mode. The application developers have the flexibility of using either Automatic Transaction Control or Manual Transaction Control. In manual commit mode, transactions are initiated implicitly, but must be committed or rolled back explicitly. To gain manual control of transactions specify txn_manual in the set_commit_mode() method of the connection object. Listing 5 illustrates an os_db_connection object being set for manual commit mode.

Listing 5: Set manual commit mode

```
// ...

 os_db_connection connection2( os_socket_address(iaddr, 3006),
                                connect_info );

 //
 // Take manual control of transactions.
 // We will call commit() or rollback() as needed for this connection.
 //
 connection2.set_commit_mode( os_db_connection::txn_manual );


 // ...
```

[END LIST]

Database Toolkit can handle some advanced features such as stored procedure. Listing 6 demonstrates handling stored procedure calls using DB Toolkit.

Listing 6: Programming with stored procedure

```
// ...

os_ip_address iaddr( target_hostname );
os_db_connection connection( os_socket_address(iaddr, 3006),
                             dsn,
                             user,
                             auth );
os_db_statement stmt( "CALL UpdateSupplierByCountry(?,?");
stmt.attach( connection );
stmt.prepare();

unsigned long supplierId =10;


char countryName[51];
strcpy(countryName, "USA");
stmt.bind_param(1, supplierId, OSSQL_INTEGER);
stmt.bind_param(2, &countryName[0], OSSQL_CHAR, 50, sizeof(countryName));
long updated_rows = 0;


stmt.execute();  // Automatically sends the bound parameters
updated_rows += stmt.get_num_rows_affected();


cout << "Total updated rows = " << updated_rows << endl;  // ...
```

[END LIST]

The Database Toolkit also contains classes for handling exceptions. It uses the ANSI/ISO C++ exception mechanism to report errors. The toolkit has a specialized error class that it uses to encapsulate the problem details. Os_db_error is used to detect and recover from errors in any Recursion Software Database Toolkit.

RECURSION SOFTWARE, Inc.

Conclusion

Given the portability, simplicity, easy to use APIs and powerful features, the Database Toolkit is a very useful tool that all database programmers should have. It is portable and cost effective software that provides optimal solutions for data management in many different environments.

Additional Resources

ODBC Programmer's Reference. http://msdn2.microsoft.com/en-us/library/ms714177.aspx

Open Database Access and ODBC. http://www.firstsql.com

Copyright © 2008
Recursion Software, Inc.
All rights reserved. C++
Toolkits is a trademark of
Recursion Software, Inc. All
other names and trademarks
are the property of their
respective owners.

Recursion Software, Inc.
2591 North Dallas Parkway
Suite 200
Frisco, Texas 75034
1.800.727.8674 or 972.731.8800
www.recursionsw.com