Lab 07 – Database Application Development

Objective:

In this lab students learn:

- How to connect to an Oracle server from a C++ program.
- How to write and execute SQL queries in a C++ program.

Connecting to an Oracle database from a C++ Program

- Create a new C++ project in Visual Studio. Add a source code named databaseConnection.cpp to your project.
- At the beginning of the C++ program, add the following libraries.

```
#include <iostream>
#include <occi.h>

using oracle::occi::Environment;
using oracle::occi::Connection;

using namespace oracle::occi;
using namespace std;
```

- Before building the connection, you need to create an environment.
  The Environment class provides an OCCI environment to manage memory and other resources for OCCI objects.
  To define an environment instance define a reference of type Environment class.

```
Environment* env = nullptr;

env = Environment::createEnvironment(Environment::DEFAULT);

{
   . . .
}
Environment::terminateEnvironment(env);
```

  Terminate the environment when the connection is not required.

Opening and Closing a Connection

- To define a pointer (reference variable) to the Oracle database.

```
Connection* conn = nullptr;
```

- After creating the environment, create a connection.

```
conn = env->createConnection(user, pass, constr);
```

  You must close and terminate the collection at the end of a working session.

```
env->terminateConnection(conn);
```

See the following code:

```
env = Environment::createEnvironment(Environment::DEFAULT);
// environment scope starts
{
conn = env->createConnection(user, pass, constr);
. . . // work with the database
env->terminateConnection(conn);
}
// environment scope ends
Environment::terminateEnvironment(env);
```

- To establish a connection to the Oracle server write the following command:

```
conn = env->createConnection(user, pass, constr);
```

You need to declare the following variables before creating the connection:

```
string user = "username";
string pass = "password";
string constr = "myoracle12c.senecacollege.ca:1521/oracle12c";
```

Use your Oracle username and password to set the variable *user* and *pass*.

- Use try-catch statements to handle any errors as a result of a connection failure.

```
 try {
     env = Environment::createEnvironment(Environment::DEFAULT);
    {
     conn = env->createConnection(user, pass, constr);
     cout << "Connection is Successful!" << endl;
     env->terminateConnection(conn);
    }
     Environment::terminateEnvironment(env);
 }
 catch (SQLException& sqlExcp) {
     cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
 }
```

See the following sample code for establishing a connection to a database:

```
#include <iostream>
#include <occi.h>

using oracle::occi::Environment;
using oracle::occi::Connection;
using namespace oracle::occi;
using namespace std;

int main(void)
{
    /* OCCI Variables */
    Environment* env = nullptr;
    Connection* conn = nullptr;
    /* Used Variables */
    string user = "username";
    string pass = "password";
    string constr = "myoracle12c.senecacollege.ca:1521/oracle12c";
    try {
        env = Environment::createEnvironment(Environment::DEFAULT);
        conn = env->createConnection(user, pass, constr);
        cout << "Connection is Successful!" << endl;
        env->terminateConnection(conn);
        Environment::terminateEnvironment(env);
    }
    catch (SQLException& sqlExcp) {
        cout << sqlExcp.getErrorCode() << ": " << sqlExcp.getMessage();
    }
    return 0;
}
```

Creating and Terminating a Statement

- If the connection is successfully established, you can execute SQL queries in your C++ program. To execute a query, you need to create a statement object by calling a method of the connection object:

  ```
  // define a reference to an object statement
  Statement* stmt = nullptr;

  // call method createStatement() to create an statement object
  conn->createStatement("SELECT * FROM product_categories");
  ```

  Termonate a statement before closing the connection when you do not need that object any more.

  ```
  conn->terminateStatement(stmt);
  ```

Executing a Statement and Store the Query Result into a Result Set

- After declaring your SQL statement, you can execute it by calling the *executeQuery()* method:

  ```
  stmt->executeQuery();
  ```

  The *executeQuery( )* method returns a ResultSet Object. To store the returning result set, you need to declare a ResultSet object.

```
// define a reference to an object resultset
ResultSet* rs = nullptr;

// store the result set
rs = stmt->executeQuery();
```

Fetching Data from a result set

- After calling the *executeQuery( )* method, you can check if the result is empty or not.

```
if (!rs->next()) {
     // if the result set is empty
     cout << "ResultSet is empty." << endl;
}
```

The *next()* method of the *ResultSet* object is used to fetch the data. Every time you call this method, one row will be fetched from the result set if exists.

When there is no data to be fetched, this method returns false (0).

Be careful when you are using the *next()* method to see if the result set is empty. This method fetches a row from your result set. If you want to display all data in your result set, make sure you do not miss the first row.

If you want to print all the rows, you need a loop. See the following code:

```
if (!rs->next()) {
     // if the result set is empty
     cout << "ResultSet is empty." << endl;
}else{

    while (rs->next()) {
        cout << "Category ID: " << rs->getInt(1) << " Category Name: " << rs->getString(2) << endl;
    }

}
```

If you use the above code to check the result set and then read the data and print them out, you will miss printing the first since you have not printed the data fetch the first time you called the next() method.

To fix this problem, you first need to print the first row fetch by the first call of the next() method. Then, use a loop to read the rest of the result set.

```
if (!rs->next()) {
      // if the result set is empty
      cout << "ResultSet is empty." << endl;
 }
 else {
      // if the result set in not empty
      do {
          cout << "Category ID: " << rs->getInt(1) << " Category Name: " << rs->getString(2) << endl;
      } while (rs->next()); //if there is more rows, iterate
 }
```