**Using Oracle-JDBC**

This write-up covers the very simple basics of using JDBC with Oracle. For more information on JDBC, use other links such as <http://docs.oracle.com/javase/tutorial/jdbc/index.html>

First, we will have to get the Oracle JDBC driver and make it available to our Java IDE. I use Eclipse Helios IDE, and I show the steps for this IDE.

Get the Oracle JDBC driver from <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-111060-084321.html> (note that we are using Oracle DB version 11.1.0.7.0)

I will get ojdbc6.jar – I have tested that this JDBC driver works with JDK 7 as well.

Now, in the Java Project, I must make this library available. This is done by (<http://www.ugrad.cs.ubc.ca/~cs304/2009W2/tutorials/JDBC/OracleFromEclipse.htm>):

* In the Java perspective right-click the project name in the Package Explorer window.
* In the pop-up menu choose Properties
* On the left  list of the new window choose Java Build Path
* On the right part of the window choose the Libraries
* Click the Add External JARs button
* In the file chooser search and find the jar file with the oracle drivers you downloaded
* Click Open to add it to Libraries.

To check that we are all set, let us write a simple test program:

The following are the main things that you will need to have in your Java code:

Load the driver (needed only for older versions of JDBC – prior to 4.0) as:

Class.forName ("oracle.jdbc.driver.OracleDriver");

Connect to the database: Connection conn = DriverManager.getConnection ("jdbc:oracle:thin:@db.csep.umflint.edu:1521:csep", "<uname>", "<passwd>");

Replace <uname> and <passwd> with your Oracle user name and password.

Here db.csep.umflint.edu is our oracle DB server machine, 1521 is the port where Oracle server listens to for JDBC connections, and csep is the database instance where we create our tables.

Now you are all set to prepare and execute statements.

Create a statement by Statement stmt = conn.createStatement ();

To execute a query, we use either

stmt.executeQuery (sql) (or)

stmt.executeUpdate (sql)

Here sql is a string representing the SQL query. We use executeQuery for SELECT statements (statements that return a set of rows), and executeUpdate for DDL statements (creating tables etc), and for data modification (inserting/deleting/updating tuples).

Let us look at a full example, which will create a table a (col1 int, col2 int), insert two rows in it, and select those rows and display them.

import java.sql.\*;

import java.util.\*;

import java.io.\*;

public class Test {

public static void main (String args []) {

Connection conn = null;

Statement stmt = null;

try {

conn = DriverManager.getConnection

("jdbc:oracle:thin:@db.csep.umflint.edu:1521:csep",

"*<userName>*", "*<passwd>*"); // Connect to DB

stmt = conn.createStatement (); // Create statement

} catch (SQLException ex) {

System.out.println ("Could not create Statement:" +

ex.toString ());

}

String sql = "CREATE TABLE a (col1 int, col2 int)";

try {

stmt.executeUpdate (sql); // create the table

} catch (SQLException sqlEx) {

System.out.println ("Could not create table:" +

sqlEx.toString ());

}

String ins1 = "INSERT INTO a values (1, 11)";

String ins2 = "INSERT INTO a values (2, 12)";

try {

stmt.executeUpdate (ins1); // insert the values

stmt.executeUpdate (ins2);

} catch (SQLException sqlEx) {

System.out.println ("Could not create table:"

+ sqlEx.toString ());

}

sql = "SELECT \* from a";

try {

ResultSet r = stmt.executeQuery (sql); // execute the query

while (r.next ()) {

System.out.println ("col1 = " + r.getInt ("col1") +

": col2 = " + r.getInt ("col2"));

}

} catch (SQLException sqlEx) {

System.out.println ("Could not get results:" +

sqlEx.toString ());

}

try {

stmt.close (); // Close the statement

conn.close (); // Close the connection

} catch (SQLException sqlEx) {

System.out.println ("Could not close connection:" +

sqlEx.toString ());

}

}

}

Transaction Example

Let us first create 2 tables and insert 1 row of data into the 2 tables as follows (in SQL Developer):

create table test1 (col1 int primary key, col2 int);

create table test2 (col3 int primary key, col4 int);

insert into test1 values (1, 11);

insert into test2 values (101, 1001);

commit;

We will do the following:

Transaction 1 inserts rows (2, 12) into test1 and (102, 1002) into test2 – this transaction should succeed.

Transaction 2 inserts rows (3, 13) into test1 and (101, 1003) into test2 – this transaction should fail. We should see that row (3, 13) was not inserted into test1.

Transaction 3 inserts rows (3, 15) into test1 – this transaction should succeed as transaction 2 failed.

Remember: As we are interacting with the DB through 2 clients now: one is SQL Developer, and another is our Java program, we will have to commit so that the changes made by one client is visible to the other client.

import java.sql.\*;

import java.util.\*;

import java.io.\*;

public class TestTransaction {

public static void main (String args []) throws Exception {

Connection conn = null;

Statement stmt = null;

try {

conn = DriverManager.getConnection

("jdbc:oracle:thin:@db.csep.umflint.edu:1521:csep",

"mmani", "mmani"); // Connect to DB

stmt = conn.createStatement (); // Create statement

} catch (SQLException ex) {

System.out.println ("Could not create Statement:" +

ex.toString ());

}

try { // set auto commit to false, so that we can have multi-statement transactions

conn.setAutoCommit(false);

} catch (SQLException sqlEx) {

System.err.println ("Could not set auto commit to false:"

+ sqlEx.toString());

}

System.out.println ("before first transaction");

String ins1 = "INSERT INTO test1 values (2, 12)";

String ins2 = "INSERT INTO test2 values (102, 1002)";

try {

stmt.executeUpdate (ins1); // insert the values

stmt.executeUpdate (ins2);

conn.commit ();

} catch (SQLException sqlEx) {

System.out.println ("Could not perform inserts:"

+ sqlEx.toString ());

conn.rollback(); // if exception, rollback

}

System.out.println ("before second transaction");

ins1 = "INSERT INTO test1 values (3, 13)";

ins2 = "INSERT INTO test2 values (101, 1003)";

try {

stmt.executeUpdate (ins1); // insert the values

stmt.executeUpdate (ins2);

conn.commit ();

} catch (SQLException sqlEx) {

System.out.println ("Could not perform inserts:"

+ sqlEx.toString ());

conn.rollback(); // if exception, rollback

}

System.out.println ("before third transaction");

ins1 = "INSERT INTO test1 values (3, 15)";

try {

stmt.executeUpdate (ins1); // insert the values

conn.commit ();

} catch (SQLException sqlEx) {

System.out.println ("Could not perform inserts:"

+ sqlEx.toString ());

conn.rollback(); // if exception, rollback

}

System.out.println ("closing statement/connection");

try {

stmt.close();

conn.close();

} catch (SQLException sqlEx) {

System.out.println ("Could not close statement/connection:"

+ sqlEx.toString ());

}

}

}