Administración y Programación de Bases de Datos
Laboratorio JDBC

Ingeniería Civil Informática / DCCTI / **Universidad del Bío-Bío**

Profesor: Gilberto Gutiérrez R.

Otoño 2019

1 Un ejemplo sencillo

JDBC (Java Database Connectivity) es una API que permite ejecutar operaciones sobre bases de datos desde programas escritos en Java. Con JDBC es posible realizar operaciones de bases de datos implementadas en diferentes productos (SABD) y ejecutándose en diferentes sistemas operativos. Además un mismo programa Java puede establecer diferentes conexiones. Es decir, puede accesar bases de datos ubicadas en diferentes servidores y realizar un procesamiento interno complejo. Algunas de las clases que ofrece el paquete `java.sql` permite:

1. `DriverManager:` Para gestionar los drives (controladores) de comunicación con el SABD.
2. `Connection:` Para establecer conexiones con las bases de datos.
3. `Statement:` Para enviar y ejecutar sentencias SQL.
4. `ResultSet:` Para almacenar y procesar el resultado de la consulta.

En este laboratorio trabajaremos con el SABD Oracle, por lo tanto utilizaremos algunos de los drivers (controladores) definidos para este SABD. En todo caso, como se indicó arriba, con JDBC es posible conectarse a otros SABDs y obviamente que para ello habrá que utlizar los drivers adecuado.

En los ejemplos de este laboratorio suponemos que tiene disponible la tabla `AMIGOS` (creada en laboratorios anteriores). Asumiendo que está conectado al servidor *colvin.chillan.ubiobio.cl* (donde reside el SABD Oracle), escriba y pruebe el siguiente programa ejemplo[1].

```
0) import java.sql.*;
1) class getOneRow {
2) public static void main (String args []) throws SQLException  {
3)    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
4)    Connection conn = DriverManager.getConnection
```

[1]Los programas `getOneRow` y `getRows` se encuentran en la página del curso (`ejemJDBC`) Pueden bajarlo a la máquina *colvin.chillan.ubiobio.cl*, cambiar el username y password, compilarlo y ejecutarlo

```
      ("jdbc:oracle:thin:@oracle.localdomain:1521:orcl", "username", "password");
5)    Statement stmt = conn.createStatement();
6)    ResultSet rs = stmt.executeQuery("SELECT NOMBRE  FROM AMIGOS WHERE ID =500");
7)    rs.next();
8)    String elNombre = rs.getString("NOMBRE");
9)    System.out.println("Mi amigo/a con ID = 500 Se llama " +  elNombre);
10)    stmt.close();
11)  }
12) } // end clase getOneRow
```

En el programa `getOneRow` se han agregado números de líneas para facilitar la explicación. Obviamente que estos números no forman parte del programa. Debido a la portabilidad que promueve Java, muchos desarrolladores de SABDs proporcionan drivers para JDBC. Un driver JDBC es básicamente una implementación de las funciones especificadas en la interfaz de programación de aplicaciones (API) JDBC para el SABD particular (Oracle, Postgres, MySQL, etc). Asi, y como ya dijimos, un programa java con llamadas JDBC puede acceder a cualquier SABD que tenga un driver JDBC.

Antes de procesar las llamadas a funciones JDBC, es necesario importar las bibliotecas de clase JDBC, que se denominan `java.sql.*`.

A continuación explicamos brevemente cada una de las instrucciones del programa ejemplo.

1. `Importar bibliotecas de clases JDBC`. El programa Java debe importar la biblioteca de clases de JDBC. Estas se denomina `java.sql.*`; ver línea 0.

2. `Cargar el o los drivers`. Como dijimos un programa java puede acceder a varias bases de datos las cuales puede estar, incluso, implementadas en diferentes SABDs. El programa debe cargar los drivers de cada uno de los tipos de bases de datos que utilizará. En el programa ejemplo, en la línea 3 se registra el driver de Oracle. El programa utiliza un sólo driver.

3. `Crear una conexión`. Mediante la función `getConnection()` de la clase `DriverManager` se crea un objeto de tipo `Connection`. Ver línea 4. Los argumentos son la url de la fuente de datos, el nombre del usuario oracle y su password.

4. `Crear un objeto Statement`. Ver línea 5. En JDBC, hay un tipo de sentencia, `Statement` básica, con dos subclases especializadas : `PrepararedStatement` y `CallableStatement` que ya hablaremos de estas últimas. Notar que el objeto de tipo `Statement stmt` está "asociado" a un objeto de tipo `Connection`. El objeto `stmt` es utilizado para comunicar las instrucciones SQL que se quieren ejecutar.

5. `Ejecutar instrucciones SQL`. En la línea 6 se ejecuta la instrucción SQL. Notar que la instrucción SQL se especifica como parámetro del método `executeQuery` del objeto `stmt`. En general la instrucción SQL corresponde a un `string` y por lo tanto se puede reemplazar por una variable de este tipo. JDBC tiene un método `execute` genérico, más dos funciones especializadas: `executeUpdate` y `executeQuery.executeUpdate` las cuales se utilizan para modificar la Base de Datos (insertar, eliminar, actualizar, etc). Estos métodos devuelven un valor entero para indicar el número de tuplas afectadas.

6. `Procesamiento de los resultados de la ejecución de la Instrucción SQL`. El método `executeQuery` se utiliza para la recuperación de tuplas (instrucciones SELECT) y devuelve un objeto de tipo `ResultSet` (ver línea 6) y que es muy similar a nuestro concepto de `CURSOR` visto en PL/SQL.

En el ejemplo, sabemos que sólo se devuelve una sola tupla. Para acceder a ella usamos el método `next` de la clase `ResultSet` (ver línea 7).

7. `Acesos a los atributos o columnas de una tupla`. En la línea 8 se muestra como se accede, en nuestro ejemplo, a la única columa de la fila recuperada. Se utiliza el método `getString` al cual se le pasa como parámetro el nombre del atributo, en este caso **NOMBRE**. Existen diversos métodos `get` para recuperar las datos de las columnas. Por ejemplo `getString, getInt, getDouble`, etc. También es posible utilizar las posiciones que ocupan los atributos o columnas dentro de la tupla recuperada. Asi es posible utilizar también `getString(1)` en la línea 8, pues el atributo **NOMBRE** es el primero de la tupla.

En general el programador puede comprobar (`try ... catch`) las excepciones SQL después de cada llamada a una función JDBC. En el ejemplo (línea 2) la excepción `SQLException` es "lanzada" por el programa `main()`. Por ejemplo

```
 ...
Connection conn=null;
 try {
      conn = DriverManager.getConnection
      ("jdbc:oracle:thin:@oracle.localdomain:1521:orcl", "user", "passwd");
    } catch(SQLException e) {
       System.out.println(" Error en coneccion a la BD ");
       System.exit(-1);
    }
 ...
```

2 Otro ejemplo considerando el procesamiento de varias tuplas

JDBC no distingue entre consultas que devuelven una sola tupla y consultas que devuelven varias tuplas. El ejemplo anterior devuelve una sola tupla y el siguiente muestra un caso en que se devuelven varias tuplas.

```
import java.sql.*;
class getRows {
public static void main (String args []) throws SQLException
 {
   DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
   Connection conn = DriverManager.getConnection
     ("jdbc:oracle:thin:@oracle.localdomain:1521:orcl", "user", "password");
   Statement stmt = conn.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT ID, NOMBRE  FROM AMIGOS ORDER BY NOMBRE");
   while (rs.next()) {
     int id = rs.getInt("ID");
     String Nombre = rs.getString("NOMBRE");
     System.out.println(id + "  " + Nombre);
   }
```

```
    stmt.close();
  }
 } // end clase getRows
```

1. Escriba el programa y pruebelo.
2. Modifíquelo para ejecutar varias instrucciones `SELECT` sobre la tabla `MIGOS`; por ejemplo,

   ```
   Select Sexo, Count(*) from AMIGOS group by Sexo
   ```

   para determinar cuantos amigos de cada sexo existen. En este caso, es conveniente recuperar las columnas de las filas por medio de las posiciones relativas de las columnas, por ejemplo, `getString(1)` y `getInt(2)` para recuperar el Sexo y la cantidad amigos respectivamente.
3. Modifique el programa para que ahora acepte un argumento indicando si quiere listar mujeres u hombres. La idea es ejecutar el programa asi `java getRows mujeres` o asi `java getRows hombres`

3 Modificando/actualizando la Base de Datos. Inserción de una tupla

El siguiente programa Java inserta una tupla en la tabla `AMIGOS`.

```
import java.sql.*;
 class Insert {
 public static void main (String args []) throws SQLException
  {
    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
    Connection conn = DriverManager.getConnection
     ("jdbc:oracle:thin:@oracle.localdomain:1521:orcl", "user", "passwd");
    Statement stmt = conn.createStatement();
    stmt.executeUpdate("INSERT INTO AMIGOS VALUES(500, 'Lorena', 56488829, date '1970-08-1', 'F')");
    stmt.close();
  }
 } //
```

1. Modifique el programa de arriba de tal manera que se pueda insertar una tupla dando en que los valores de los atributos se ingresan como argumentos del programa.

   ```
   java Insert ID Nombre Fono FNacimiento Sexo
   ```

   Por ejemplo

   ```
   java Insert 1000 Rebeca 0896989 1975-10-2 F
   ```