JDBC – Java Database Conectivity

Bases de Datos 1

La API JDBC permite acceder desde Java a datos almacenados de forma tabular, especialmente Bases de datos Relacionales.

Las actividades que controla el JDBC son:

- Conexión a la fuente de datos (Base de Datos)
- Envío de instrucciones de consulta (*queries*) o actualización hacia la base de datos.
- Recuperación y procesamiento de los resultados recibidos desde la base de datos.

JDBC incluye cuatro componentes:

1. **JDCB API:** Provee acceso a datos relacionales desde JAVA, permite ejecutar sentencias SQL, recuperar resultados y enviar cambios a la fuente de datos. Permite también, interactuar con múltiples fuentes de datos en un entorno distribuido.
2. **JDBC Driver Manager**: Define objetos con los que se puede realizar la conexión al driver JDBC.
3. **JDBC Test Suite:** Ayuda a determinar que drivers manejará el programa.
4. **JDBC – ODBC Bridge:** Provee acceso a JDBC a través de los drivers ODBC. Este código ODBC debe cargarse en cada máquina cliente.

Arquitectura

El JDBC permite trabajar con arquitecturas de software de 2 o 3 capas. En una arquitectura de dos capas (aplicación y datos), el JDBC se ubica en la capa de aplicación; en una arquitectura de tres capas (aplicación, control y datos), el JDBC se ubica en la capa del control, desde donde se realiza el acceso a los datos.

En cualquier caso, el JDBC debe entenderse como el intermediario entre la aplicación que se realiza en Java y el DBMS correspondiente que se esté usando (por ejemplo, Oracle), lo que da como ventaja, entre otras cosas, que se pueda cambiar fácilmente de manejador para la base de datos, sin que esto afecte las sentencias de la aplicación. La siguiente figura presenta un esquema de las conexiones entre la aplicación, el JDBC y el DBMS.

USANDO EL JDBC

Para acceder a una Base de Datos desde Java, lo primero que se debe hacer es establecer la conexión con la Base de Datos.

Generalmente una aplicación JDBC se conecta a una fuente de datos usando dos mecanismos:

1. **DriverManager:** Permite cargar un driver específico usando una dirección URL (lo que permite acceder a una Base de Datos que se encuentre en otro sitio físico). Como parte de la inicialización, el DriverManager carga las clases *driver* referenciadas.

2. **DataSource:** Esta interface es preferida sobre el DriverManager, ya que permite manejar detalles adicionales sobre la fuente de datos.

Observación: La mayoría de los métodos de la API JDBC pueden generar la excepción *SQLException*, por lo que el compilador exigirá que las instrucciones respectivas se ubiquen dentro de una cláusula try – catch que atrape dicha excepción.

Para establecer la conexión se requieren dos pasos: cargar el driver y crear la conexión efectivamente.

1. Cargar el Driver

Para realizar la carga del driver, se debe verificar que este se encuentre disponible desde Java, si no es así, se debe buscar el driver en la página del fabricante e instalarlo en el directorio respectivo de Java. En la documentación asociada al driver se indica el nombre que debe usarse para su acceso.

Por ejemplo:

Para MySQL:

```
Class.forName("com.mysql.jdbc.Driver");
```

Para Oracle:

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

2. Establecer la Conexión

Para establecer la conexión, se emplea la sentencia *getConnection*, dentro de la cual se debe indicar el URL de ubicación de la Base de Datos, su nombre y se puede incluir el nombre de usuario y contraseña.

```
Connection conn = DriverManager.getConnection("jdbc:derby:COFFEES");
```

En el ejemplo anterior el nombre de la base de datos es COFFEES.

```
String url = "jdbc:derby:Fred";
Connection con = DriverManager.getConnection(url, "Fernanda", "J8");
```

En el ejemplo anterior el nombre de la base de datos es Fred, el login de la base de datos es Fernanda[1] y el password es J8.

Una vez se ha establecido la conexión, se empleará esta para realizar las diferentes operaciones que se requieran sobre la base de datos.

CONSULTAS SOBRE LA BASE DE DATOS

Para realizar una consulta sobre la base de datos, se emplearán las sentencias SQL estándar, atendiendo el siguiente proceso:

1. Debe crearse una sentencia (*statement*) para realizar la consulta, esta sentencia será un objeto de alguna clase que implemente java.sql.Statement

```
Statement stmt = con.createStatement( );
```

2. Debe construirse la consulta a realizar como una cadena, esta consulta debe escribirse tal como se haría en el manejador sin incluir el punto y coma final.

```
String query = "SELECT * FROM Customer";
```

3. Finalmente se ejecuta el método *executeQuery()* para enviar la consulta al manejador. El ejecutar una consulta (query), el método retorna un objeto de clase Resultset, por tanto debe recibirse en una variable de este tipo (más adelante se da una visión general de esta clase).

```
ResultSet rs = stmt.executeQuery(query);
```

ACTUALIZACIÓN DE LA BASE DE DATOS

Si se desea realizar una operación de inserción, borrado o actualización de datos sobre la Base de Datos, debe emplearse el método *executeUpdate()* en lugar del executeQuery, este método retorna un entero donde indica cuántas filas o registros de la Base de Datos se afectaron al ejecutar la instrucción.

```
String request = "INSERT INTO Customer (ssn, cust_name, address)" + "VALUES ('222','Juan','Calle 10')";
int rowsAffected = stmt.executeUpdate(request);
```

```
String request = "DELETE FROM Customer WHERE ssn='222'";
int rowsAffected = stmt.executeUpdate(request);
```

```
String request = "UPDATE Customer SET cust_name='Jose', address='Carrera 10' WHERE ssn='222'";
int rowsAffected = stmt.executeUpdate(request);
```

EL RESULTSET

ResultSet es una tabla de datos que representa el conjunto de resultados que se obtienen de una consulta a una base de datos; un objeto derivado de ResultSet mantiene un cursor a una fila de la tabla, inicialmente se ubica antes de la primera posición y va cambiando a la siguiente cada vez que se da la instrucción next(), cuando no hay más filas en la tabla, este método retornará false.

Algunos métodos de la interface ResultSet son:

Método	Accion
next()	mueve el cursor a la siguiente fila.
previous()	mueve el cursor a la fila anterior.
first()	mueve el cursor a la primera fila.
last()	mueve el cursos a la última fila.
beforeFirst()	posiciona el cursor al inicio del *ResultSet*, antes de la primera fila.
afterLast()	posiciona el cursor al final del *ResultSet*, después de la última fila.
relative(int rows)	mueve el cursos a una posición relativa a la actual.
absolute(int row)	mueve el cursos a una posición absoluta.
getXxx()	retorna el valor de la columna especificada por el nombre o indice. Los tipos válidos son: double, byte, int, Date, String, float, short, long, Time, Object

La forma más sencilla de recorrer un ResultSet es la siguiente:

```
String query = “SELECT ssn, cust_name, address FROM Customer”;
ResultSet rs = stmt.executeQuery(query);

while (rs.next()) {
    System.out.println(“SSN: “+ rs.getInt(“ssn”).trim());
    System.out.println(“Name: “+rs.getString(“cust_name”).trim());
    System.out.println(“Address:  “+rs.getString(“address”).trim();
    System.out.println(“ ”);
}
```

En este caso, se ejecuta el ciclo mientras existan más registros o filas en la tabla del resultSet, para cada registro se recupera el campo denominado ssn, el campo cust_name y el campo address; estos campos pueden recuperarse con el nombre del campo o con el número de la posición en que se encuentra el campo, por ejemplo, ssn sería el campo 1, cust_name el campo 2 y address el campo, por lo tanto, la misma porción de código se podría escribir así:

```
String query = “SELECT ssn, cust_name, address FROM Customer”;
ResultSet rs = stmt.executeQuery(query);

while (rs.next()) {
    System.out.println(“SSN: “+ rs.getInt(1).trim());
    System.out.println(“Name: “+rs.getString(2).trim());
    System.out.println(“Address:  “+rs.getString(3).trim();
    System.out.println(“ ”);
}
```

Observe que, en la primera forma, los nombres de las columnas se escriben entre comillas, ya que son cadenas; en la segunda forma son enteros, por tanto, no deben incluirse comillas.

Para realizar una correcta recuperación de datos, es preciso conocer el tipo de cada columna en la Base de datos, en este caso, el ssn es un entero y por ello se recupera con getInt(), a diferencia del nombre y dirección que son cadenas y se recuperan con getString().

Ejemplo 1:

La siguiente clase, permite consultar los datos de la tabla llamada PROFESORES, en la base de datos denominada OBJETOS.

```
public class ConsultaProfesores2 {

  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver ";

  // URL de la base de datos
 static final String DATABASE_URL = "jdbc:mysql:/// objetos";

  // declaración de la conexión y el statement
  // para acceso a la base de datos
  private Connection connection;
  private Statement statement;

  // constructor que crea el Frame y hace la conexión a la base de datos
  public ConsultaProfesores2()
  {
    // conecta a la base de datos
    try {
      // carga el driver correspondiente
      Class.forName( JDBC_DRIVER );

      // establece la conexión a la base de datos, login y password son system
      connection = DriverManager.getConnection( DATABASE_URL,"root","root" );

      //  crea un statement para consulta
      statement = connection.createStatement();

      // consulta a la base de datos
      ResultSet resultSet =
        statement.executeQuery( "SELECT nombre,email,cedula,cod_dpto FROM profesores" );
```

```
    // Procesamiento de los resultados
    StringBuffer results = new StringBuffer();

    //la MetaData contiene los datos de la estructura de la
    //respuesta, por ejemplo, los nombres de las columnas
    ResultSetMetaData metaData = resultSet.getMetaData();

    //se consulta el número de columnas de la respuesta
    int numberOfColumns = metaData.getColumnCount();

    //se adiciona el nombre de cada columna y un tabulador
    for ( int i = 1; i <= numberOfColumns; i++ )
      results.append( metaData.getColumnName( i ) + "\t" );
    results.append( "\n" );

    //se adicionan los resultados que estan en resultSet
    while ( resultSet.next() ) {
      for ( int i = 1; i <= numberOfColumns; i++ )
        results.append( resultSet.getObject( i ) + "\t" );

      results.append( "\n" );
    }

    // estalece la GUI, un area de Texto
    System.out.println(results);

  } // end try

  //detecta los problemas al interactual con la base de datos
  catch ( SQLException sqlException ) {
    JOptionPane.showMessageDialog( null, sqlException.getMessage(),
      "Database Error", JOptionPane.ERROR_MESSAGE );
    System.exit( 1 );
  }

  // detecta problemas al cargar el driver
  catch ( ClassNotFoundException classNotFound ) {
    JOptionPane.showMessageDialog( null, classNotFound.getMessage(),
      "Driver Not Found", JOptionPane.ERROR_MESSAGE );
    System.exit( 1 );
  }

  // se asegura que al finalizar se cierre el statement y la conexión
  // no importa si ocurrió una excepción o no en ejecución
  finally {
    try {
      statement.close();
```

```
      connection.close();
    }
    // Maneja las excepciones que puedan ocurrir en el cierre
    catch ( SQLException sqlException ) {
      JOptionPane.showMessageDialog( null,
        sqlException.getMessage(), "Database Error",
        JOptionPane.ERROR_MESSAGE );
      System.exit( 1 );
    }
  }

 } // fin del constructor
```

Ejemplo 2:

La siguiente clase, permite la inserción de datos en una tabla llamada DEPARTAMENTO, en la base de datos denominada OBJETOS.

```
public class InsertarDpto extends JFrame {

  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver ";

  // URL de la base de datos
 static final String DATABASE_URL = "jdbc:mysql:///objetos";

  // declaración de la conexión y el statement  para acceso a la base de datos
  private Connection connection;
  private Statement statement;

  // constructor que crea el Frame y hace la conexión a la base de datos
  public InsertarDpto()
  {
    super( "Ingreso de datos" );

    // conecta a la base de datos
    try {
      // carga el driver correspondiente
      Class.forName( JDBC_DRIVER );
      // establece la conexión a la base de datos
      connection = DriverManager.getConnection( DATABASE_URL,"root","root" );
      //  crea un statement para consulta
      statement = connection.createStatement();
```

```
    // inserta los datos en la tabla correspondiente
    int L = statement.executeUpdate( "INSERT INTO departamentos VALUES ('Biologia', 600, 300)");
  } // end try

  //detecta los problemas al interactual con la base de datos
  catch ( SQLException sqlException ) {
    JOptionPane.showMessageDialog( null, sqlException.getMessage(),
      "Database Error", JOptionPane.ERROR_MESSAGE );
    System.exit( 1 );
  }

  // detecta problemas al cargar el driver
  catch ( ClassNotFoundException classNotFound ) {
    JOptionPane.showMessageDialog( null, classNotFound.getMessage(),
      "Driver Not Found", JOptionPane.ERROR_MESSAGE );
    System.exit( 1 );
  }

  // se asegura que al finalizar se cierre el statement y la conexión
  // no importa si ocurrió una excepción o no en ejecución
  finally {
    try {
     statement.close();
      connection.close();
    }

    // Maneja las excepciones que puedan ocurrir en el cierre
    catch ( SQLException sqlException ) {
      JOptionPane.showMessageDialog( null,
        sqlException.getMessage(), "Database Error",
        JOptionPane.ERROR_MESSAGE );
      System.exit( 1 );
    }
  }
} // fin del constructor
```

[1] En el caso de MySQL el usuario por defecto es root, así que este debería ser el login aemplear.