Manejo de conectores (2ª Parte)

La primera parte de la unidad ha sido una introducción al acceso a datos. Hemos repasado de forma breve algunos conceptos sobre acceso a datos, hemos conocido nuevas bases de datos y las hemos manejado un poco desde la línea de comandos.

En lo que queda de la unidad veremos cómo acceder a esos mismos datos desde un programa Java utilizando conectores. Para ello empezaremos por introducir una serie de conceptos que luego pondremos en práctica.

1. Protocolos de acceso a datos

Muchos servidores de BBDDs utilizaban protocolos específicos de la marca, lo que obligaba a aprender un lenguaje nuevo para trabajar con cada uno de ellos. Para solucionar este problema se han desarrollado diversos protocolos que proporcionan una interfaz común (basada en SQL) para el acceso homogéneo a los datos. La idea es que el protocolo de acceso a datos sea una capa intermedia entre el programa de usuario y el SGBD que abstrae las complejidades subyacentes de cada producto y proporciona una interfaz común sea cual sea el SGBD al que se acceda.

Diversas son las tecnologías que realizan esta tarea, aquí tienes algunas de las más importantes:

1.1. ODBC

Microsoft estableció una norma común para comunicarse con las bases de datos, llamada ODBC (*Open DataBase Conectivity*). Se apoya en un amplio conjunto de drivers ODBC. Cada driver está orientado a un sistema de gestión de Base de Datos concreto. Estos drivers entienden una serie de funciones que permiten utilizar el lenguaje SQL estándar y las traduce a la base de datos subyacente. ODBC permite la conexión fácil desde varios lenguajes de programación, y tiene mucha aceptación, sobre todo en entornos Windows. En plataformas Linux/Unix su uso no es tan cómodo y suelen utilizarse otras alternativas. Sobre ODBC Microsoft ha construido sus extensiones de Ole DB (aunque inicialmente creado para acceso a base de datos, por ejemplo Access, actualmente se suele utilizar para orígenes de datos que no son bases de datos, como por ejemplo hojas de cálculo), ADO (una evolución del DAO y que se puede utilizar con lenguajes de guiones como VBcSript y JScript), ADO.NET (el que más se está utilizando en aplicaciones Microsoft).

1.2. Perl:DBI

Perl es uno de los lenguajes utilizados para programación en la Web y proporciona su propia interfaz de acceso a datos, llamada DBI (*DataBase Interface*). Es especialmente utilizado bajo plataformas Linux/Unix, solucionando las complejidades de ODBC en estos sistemas. DBI actúa como una abstracción para un conjunto de módulos *DBD* (*DataBase Driver*). Cada módulo DBD actúa como manejador de un sistema gestor de base de datos distinto. Existen módulos para prácticamente cualquier SGBD (Oracle, Informix, MySQL, etc.) y puentes hacia otras tecnologías como ADO, JDBC ...

Actualmente algunas distribuciones de Perl se acompañan con el DBI y los drivers más habituales, aunque en otras no (como por ejemplo Red Hat).

1.3. JDBC

Java intenta proporcionar una plataforma segura, flexible y completa para desarrollo de aplicaciones en Internet. En ella no podía faltar una solución a la conectividad con bases de datos, JDBC (Java DataBase Connectivity). JDBC es el estándar para la conectividad entre el lenguaje Java y un amplio abanico de SGBDs.

Como es el que vamos a utilizar para las prácticas, lo veremos con un poco más de detalle en el punto siguiente.

2. El protocolo JDBC

JDBC son las siglas de Java Database Connectivity, que en castellano significa: Conectividad Java con Base de Datos. Consta de un conjunto de clases e interfaces Java que nos permiten acceder de una forma genérica a las Bases de Datos independientemente del proveedor del SGBD.

Cada proveedor del SGBD, dispondrá de una implementación de dichas interfaces. Dicha implementación se sabe comunicar con el SGBD de ese proveedor.

Estas clases e interfaces JDBC se encuentran en el paquete java.sql.*

Básicamente, una aplicación que usa JDBC realiza los siguientes pasos:

- Establece una conexión con una Base de Datos.
- Crea y envía una sentencia SQL a la Base de Datos.
- Procesa el resultado.
- Cierra la conexión

2.1. Tipos de drivers JDBC

Los drivers JDBC son la implementación que cada proveedor del SGBD ha realizado del API JDBC.

Existen cuatro tipos:

- Tipo 1: JDBC-ODBC Bridge.
- Tipo 2: Native-API partly-Java.
- Tipo 3: JDBC-Net pure Java.
- Tipo 4: Native protocol pure Java.

Los SGBDs tendrán un fichero JAR o ZIP con las clases del Driver JDBC que habrá que añadir a la variable CLASSPATH del sistema para poder utilizarlo desde nuestras aplicaciones Java.

JDBC-ODBC Bridge

Hay uno genérico que viene incluido con el JDK: **sun.jdbc.odbc.JdbcOdbcDriver**

Traduce llamadas JDBC en llamadas ODBC. Este tipo de driver requiere de la instalación y configuración del cliente ODBC.

Native-API partly-Java

Viene incluido con cada Gestor de Base de Datos.

Traduce llamadas JDBC a llamadas propietarias del SGBD. Al igual que ocurriera con el tipo 1, requiere instalación y configuración del cliente del SGBD.

JDBC-Net pure Java

Viene incluido con cada Gestor de Base de Datos. Conecta de manera remota (vía TCP/IP) con un daemon (listener o servicioescuchador) del SGBD (local o remoto). Y es este daemon integrado con el SGBD quien traduce las llamadas al SGBD.

No requiere ninguna instalación previa en los clientes.

Native-protocol pure Java

Viene incluido con cada Gestor de Base de Datos. Conecta de manera remota (vía TCP/IP) con el SGBD (local o remoto) directamente sin necesidad de servicios intermedios. Y al igual que el tipo 3, no requiere ninguna instalación previa en los clientes.

2.2. Ejemplos de Drivers JDBC

Algunos ejemplos de Drivers JDBC de los SGBD más populares son:

- Genérico ODBC:

- Tipo 1: **sun.jdbc.odbc.JdbcOdbcDriver**
- IBM DB2 v7 y anteriores:
  - Tipo 2: **COM.ibm.db2.jdbc.app.DB2Driver**
  - Tipo 3: **COM.ibm.db2.jdbc.net.DB2Driver**
- IBM DB2 v8 y posteriores:
  - Tipo 2 y 4: **com.ibm.db2.jcc.DB2Driver**
- Oracle:
  - Tipo 3: **oracle.jdbc.OracleDriver**
- Sybase:
  - Tipo 3: **com.sybase.jdbc2.jdbc.SybDriver**
- MySQL:
  - Tipo 4: **com.mysql.jdbc.Driver**
- Informix:
  - Tipo 3: **com.informix.jdbc.IfxDriver**
- Apache Derby:
  - Tipo 2: **org.apache.derby.jdbc.EmbeddedDriver**
  - Tipo 4: **org.apache.derby.jdbc.ClientDriver**

Nota: Apache Derby ha sido incluido en la JDK bajo el nombre de Java DB.

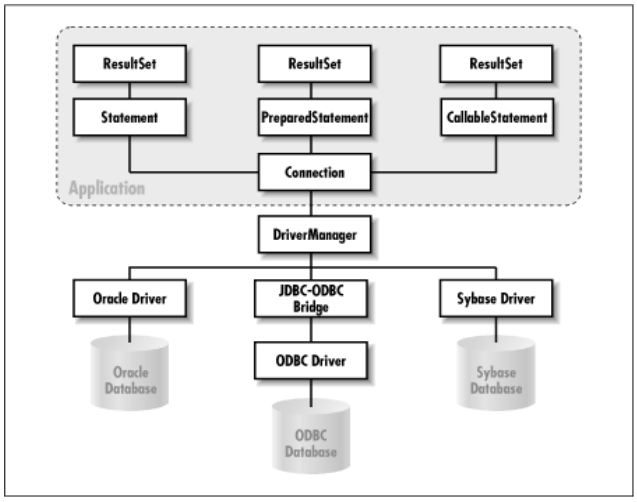
2.3. Componentes JDBC

Los componentes del API JDBC son:

- El gestor de los drivers (**java.sql.DriverManager**).
- La conexión con la Base de Datos (**java.sql.Connection**).
- La sentencia a ejecutar (**java.sql.Statement**).
- El resultado (**java.sql.ResultSet**).

Para la sentencia a ejecutar, existen distintas implementaciones. En la enumeración anterior hemos mencionado **java.sql.Statement**, pero existen otras dos opciones: **java.sql.PreparedStatement** y **java.sql.CallableStatement**

A continuación veremos con más detalles cada uno de estos componentes y practicaremos con ellos. Aquí tienes un diagrama resumen de estos componentes:



3. Establecimiento de conexiones

3.1. java.sql.DriverManager

El Driver Manager lleva el control de los drivers JDBC cargados en memoria. Además es el encargado de realizar las conexiones con la Base de Datos.

Hay que cargar en memoria los drivers JDBC para registrarlos al Driver Manager.

Se cargan mediante el método estático **forName()** de la clase **java.lang.Class**, por ejemplo:

Class.forName("com.mysql.jdbc.Driver"); carga el driver JDBC del SGBD mysql.

Veamos un ejemplo completo, en el ejemplo que tienes a continuación solo se ha redactado lo específico a la carga del driver.

- Ejemplo: Carga del Driver JDBC del SGBD DB2 de IBM:

```
importjava.sql.*;
public class TestJDBC
 {
 public static void main(String[] args)
{
 try
{
 Class.forName("com.ibm.db2.jcc.DB2Driver");
 ........
........
}
 catch(ClassNotFoundException ex)
{
 ex.printStackTrace();
 }
 }
}
```

Nota: Para poder utilizar el driver, recuerda cargarlo en las librerías del proyecto con la opción Tools--> Libraries, si Netbeans incorpora el driver, lo podrás seleccionar de la lista Libraries, si no, utiliza el botón Add Jar/folder con el que le indicarás en qué carpeta lo tienes descargado

3.2. java.sql.Connection

Representa una conexión con la Base de Datos. El encargado de abrir una conexión es el Driver Manager mediante el método estático:

```
public static Connection getConnection(url, user, pwd) throwsSQLException;
```

donde:

- url: es el identificador de la Base de Datos.
- user: usuario con el que se abre la conexión (opcional).
- pwd: contraseña del usuario (opcional).

Las URLs JDBC constan de las siguientes partes:

```
jdbc:%subprotocolo%:%subnombre%
```

donde:

- jdbc: es el protocolo (es fijo, siempre se pone jdbc).
- %subprotocol%: especifica el tipo o nombre del driver.
- %subnombre%: especifica la BBDD. Depende del tipo de driver tendrá una información u otra.

Algunos ejemplos de URLs JDBCs de los SGBD más populares son:

- Genérico ODBC:
  - Tipo 1: "jdbc:odbc:sample"
- IBM:
  - Tipo 2: "jdbc:db2:sample"
  - Tipo 3/4: "jdbc:db2://127.0.0.1:6789/sample"
- Oracle:
  - Tipo 3: "jdbc:oracle:thin:@127.0.0.1:1521:sample"
- Sybase:

- Tipo 3: "jdbc:sybase:Tds:127.0.0.1/sample"
- MySQL:
  - Tipo 3: "jdbc:mysql://127.0.0.1/sample"
- Informix:
  - Tipo 3: "jdbc:informix-sqli://127.0.0.1:1526/sample"
- Apache Derby:
  - Tipo 2: "jdbc:derby:sample"
  - Tipo 4: "jdbc:derby://127.0.0.1:1527/sample"

A través de la conexión, nos comunicaremos con la Base de Datos, enviándole sentencias SQL. Las sentencias SQL se envían a través de 'statements'.

Existen tres tipos de 'statements' y un método para generar cada tipo:

- java.sql.Statement: createStatement();
- java.sql.PreparedStatement: prepareStatement();
- java.sql.CallableStatement: prepareCall();

Veremos con más detalles estos componentes en la siguiente parte de la unidad.

Una vez hayamos terminado de trabajar con una conexión debemos liberarla.

Una conexión abierta significa recursos consumiéndose en el SGBD. Las conexiones se cierran mediante el método:

```
public void close() throws java.sql.SQLException;
```

Ejemplo: Establecimiento de una conexión con el SGBD DB2 de IBM (la base de datos a la que conectamos se llama *sample)*:

```
importjava.sql.*;
public class TestJDBC  {
  public static void main(String[] args)   {
      try  {
            Class.forName("com.ibm.db2.jcc.DB2Driver");
            Connection con =DriverManager.getConnection("jdbc:db2:sample");
            ... aquí trataríamos los datos
            ........
            con.close();
      }
      catch(ClassNotFoundException ex)  {
            ex.printStackTrace();
      }
      catch(SQLException ex) {
            ex.printStackTrace();
      }
  }
}
```

Una vez establecida la conexión podremos acceder a los datos ejecutando instrucciones SQL como veremos en la segunda parte del tema.

Una vez hemos aprendido a conectar con una base de datos desde un programa Java. Ahora veremos cómo manejar los datos utilizando esa conexión.

4. Acceso a los metadatos

Antes de empezar con las instrucciones SQL abordaremos un tema interesante que nos puede ser útil. El acceso a los metadatos para obtener información sobre la base de datos y sus componentes.

Generalmente, cuando desarrollamos una aplicación con acceso a datos conocemos la estructura de dicha base de datos, conocemos la definición de sus tablas y demás componentes, pero también puede ocurrir que no la conozcamos o que queramos hacer una aplicación independiente de la estructura de la base de datos.

En estos casos nos serán muy útiles los metaobjetos, que no son más que objetos que proporcionan información sobre la base de datos, permiten tener acceso a los metadatos (datos sobre los datos) de la BD.

Veamos algunos de ellos.

4.1. El objeto DatabaseMetaData

Proporciona una gran cantidad de información a través de múltiples métodos.

Este objeto se recupera a través del método .getMetaData() de la clase Connection.

Los nombres de los métodos que incluye son bastante explícitos: .getDatabaseProductName(), .getDriverName(), .getURL(), .getUserName()...

Lo mejor es verlo con un ejemplo:

```
public class MySQL_Metadata {
   public static void main(String[] args) {
      try {
         // Cargar el driver
         Class.forName("com.mysql.jdbc.Driver");
         // Establecemos la conexión con la base de datos
         Connection miconexion =
DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
         // Cargamos los metadatos
         DatabaseMetaData dbmd= miconexion.getMetaData();
         // Información general
         String nombre = dbmd.getDatabaseProductName();
         String driver = dbmd.getDriverName();
         String url = dbmd.getURL();
         String usuario = dbmd.getUserName();
         System.out.println("INFORMACION SOBRE LA BASE DE DATOS");
         System.out.println("---------------------------------");
         System.out.println("Nombre: " + nombre);
         System.out.println("Driver: " + driver);
         System.out.println("URL: " + url);
         System.out.println("User: " + usuario);
         // Información sobre las tablas
         ResultSet resul= dbmd.getTables(null, "Ejemplo",null, null);
         while (resul.next())
         {
            String catalogo = resul.getString(1);  // la primera columna devuelta
            String esquema  = resul.getString(2);  // la columna 2
            String tabla    = resul.getString(3);  // la columna 3
            String tipo  = resul.getString(4);
            System.out.println("Catalogo: " + catalogo + " Esquema: " + esquema + " tipo: "
+ tipo + " Nombre tabla: " + tabla);
         }
         miconexion.close();
      }
      catch (ClassNotFoundException cn) {cn.printStackTrace();}
      catch (SQLException e) {e.printStackTrace();}
     }// fin main
} // fin class
```

El método .getTables() devuelve un objeto ResultSet que proporciona información sobre las tablas y vistas de la BD como puedes apreciar en el ejemplo anterior.

Para acceder a las columnas del ResultSet hemos utilizado el índice de la columna (1,2,3...) pero también podíamos haber utilizado el nombre de la columna devuelta:

```
1--> TABLE_CAT
```

2--> TABLE_SCHEM
3--> TABLE_NAME
4--> TABLE_TYPE
5--> REMARKS.

Así en el código anterior podríamos sustituir la recuperación de los datos por:

```
{
   String catalogo = resul.getString("TABLE_CAT");  // la primera columna devuelta
   String esquema  = resul.getString("TABLE_SCHEM");  // la columna 2
   String tabla    = resul.getString("TABLE_NAME");  // la columna 3
   String tipo  = resul.getString("TABLE_TYPE");
....
```

Otros método importante del objeto DatabaseMetaData es .getColumns().

.getColumns(catalogo, esquema, nombreTabla, nombreColumna) devuelve información sobre las columnas de una tabla o tablas. Para el nombre de la tabla y columna se pueden utilizar los comodines _ y % (_:un carácter, %: 0..n caracteres) y para cualquiera de los parámetros se puede utilizar el valor null.

Aquí va un ejemplo:

```
package conectores;
import java.sql.*;
      // Previamente iría el establecimiento de la conexión y se supone
      // que en el DatabaseMetaData dbmd hemos cargado los metadatos.
      // Recuperemos información sobre las columnas de la tabla departamentos:
      ResultSet columnas = dbmd.getColumns(null, "Ejemplo", "departamentos", null);
      System.out.println("         COLUMNAS TABLA DEPARTAMENTOS");
      System.out.println("-----------------------------------------");
      while (columnas.next())
      {
         String nombreCol = columnas.getString("COLUMN_NAME");
         String tipoCol  = columnas.getString("TYPE_NAME");
         String tamCol    = columnas.getString("COLUMN_SIZE");
         String nula  = columnas.getString("IS_NULLABLE");
         System.out.println("Columna: " + nombreCol + " tipo: " + tipoCol + " tamaño: " +
tamCol + " Admite null: " + nula);
      }
```

Otros métodos interesantes, el que nos devuelve las columnas que forman la clave primaria de una tabla, .getPrimaryKeys(), y el que nos devuelve la lista de todas las claves ajenas que apuntan a la tabla indicada: .getExportedKeys(). Con estos métodos también se puede utilizar el valor null para los parámetros.

```
      // Información sobre la clave primaria de una tabla.
      // Recuerda que solo hay una pero puede estar compuesta
      ResultSet pk = dbmd.getPrimaryKeys(null, "Ejemplo", "departamentos");
      System.out.println("COLUMNAS QUE FORMAN LA PK DE LA TABLA
DEPARTAMENTOS");
      System.out.println("-------------------------------------------------");
      String pkdep = "", separador= " - ";
      while (pk.next())
      {
         pkdep = pkdep + separador + pk.getString("COLUMN_NAME");
      }
      System.out.println("Clave primaria: " + pkdep);
      // Información sobre las claves ajenas que apuntan a la clave primaria de una tabla.
      ResultSet fk = dbmd.getExportedKeys(null, "Ejemplo", "departamentos");
       System.out.println("COLUMNAS QUE APUNTAN A LA PK DE LA TABLA
DEPARTAMENTOS");
```

```
System.out.println("---------------------------------------------------");
while (fk.next())
{
  String fk_nombre = fk.getString("FKCOLUMN_NAME");
  String pk_nombre = fk.getString("PKCOLUMN_NAME");
  String fk_tabla = fk.getString("FKTABLE_NAME");
  String pk_tabla = fk.getString("PKTABLE_NAME");
  System.out.println("Tabla hija: " + fk_tabla + " Clave Ajena: " + fk_nombre);
  System.out.println("Tabla padre: " + pk_tabla + " Clave Principal: " + pk_nombre);
}.....
```

De forma similar tenemos los métodos .getImportedKeys(catalogo,esquema,tabla) que devuelve información sobre las claves ajenas de la tabla, y .getProcedures(catalogo,esquema,procedimiento) que devuelve información sobre procedimientos almacenados.

4.2. El objeto ResultSetMetaData

También podemos conseguir información más detalladas sobre las columnas de una tabla en un objeto ResultSetMetaData ejecutando el método .getMetaData() del ResultSet.

Con un objeto ResultSetMetaData podemos recuperar las propiedades de las columnas de una tabla o tablas.

La mecánica a utilizar será:

- obtener en un ResultSet la columnas de la tabla,
- ejecutar sobre el ResultSet el método .getMetaData() guardando el resultado en un objeto ResultSetMetaData
- utilizar los métodos del ResultSetMetaData para obtener la información deseada.

Lo puedes ver aquí con el ejemplo:

```
Class.forName("com.mysql.jdbc.Driver");
        Connection miconexion =
DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
        Statement sentencia= miconexion.createStatement();
        ResultSet resultado= sentencia.executeQuery("SELECT * FROM departamentos");
        ResultSetMetaData rsmd= resultado.getMetaData();
        int ncols= rsmd.getColumnCount();
        String nula;
        System.out.println("Núm. de columnas recuperadas: " + ncols);
        for (int i=1;i<=ncols;i++){
          System.out.println("Columna " + i + ":");
          System.out.println("Nombre: " + rsmd.getColumnName(i));
          System.out.println("Tipo: " + rsmd.getColumnTypeName(i));
          System.out.println("Permite nulos: " + rsmd.isNullable(i));
          System.out.println("Ancho máximo de display: " +
rsmd.getColumnDisplaySize(i));
```

Aunque el ejemplo emplea un objeto Statement que todavía no hemos aprendido, se entiende que lo que hacemos es guardar en un objeto ResultSet que llamamos resultado el resultado de la SELECT que obtiene todas las columnas de la tabla departamentos y luego obtenemos la información sobre esas columnas mediante el ResultSetMetaData.