Tema 8

JDBC.

Departament d'Informàtica. Universitat de València

Índice

1. Introducción

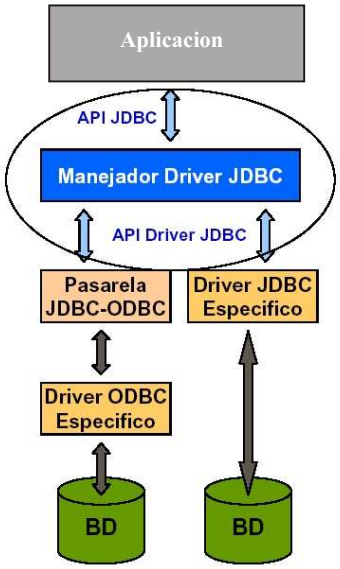
¿Qué es JDBC? es una biblioteca de clases que permite la conexión con Bases de Datos utilizando Java.

Permite realizar operaciones (consultas, actualizaciones, ...) sobre base de datos relacionales utilizando SQL (Structured Query Language).

¿Que ventajas ofrece acceder a la base de datos utilizando JDBC y Java? Que la aplicación será independiente de la plataforma y que se puede mover la aplicación de un sistema gestor de bases de datos a otro (por ejemplo de Oracle a MySQL o a Microsoft SQL Server o a...).

JDBC está compuesto por dos capas:

- API JDBC se encarga de comunicar con la API del administrador de controladores JDBC enviándole las sentencias SQL.
- Un administrador de controladores de JDBC que se comunica de forma transparente para el programador con los distintos controladores disponibles para la base de datos.



Los controladores JDBC están clasificados como:

- Controladores tipo 1: traducen JDBC a ODBC y se delega en ODBC para la comunicación con la base de datos. Sun incluye uno de estos controladores con el J2SE.
- Controlador tipo 2: está escrito parcialmente en Java y en código nativo.
- Controlador tipo 3: es una biblioteca cliente escrita completamente en Java que utiliza un protocolo independiente de la BD para comunicar las peticiones a un servidor que las traduce a un protocolo específico de la Base de Datos.
- Controlador tipo 4: es una biblioteca escrita completamente en Java que traduce las peticiones a un protocolo específico de la Base de Datos.

Lo habitual es que los distribuidores de Bases de Datos suministren controladores del tipo 3 o 4.

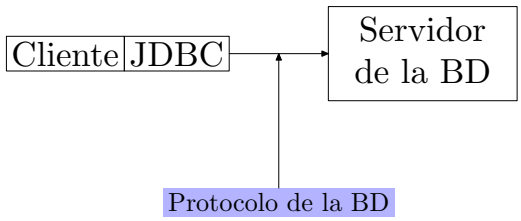
Resumiendo:

- Los programadores pueden escribir aplicaciones Java utilizando que accedan a Bases de Datos utilizando sentencias SQL estándar.
- Las clases proporcionadas en el paquete java.sql y javax.sql definen el API JDBC.
- Sun únicamente ofrece un controlador tipo 1 (puente JDBC/ODBC).
- Los distribuidores de Bases de Datos pueden ofrecer los controladores optimizándolos para sus productos.

1.1. Arquitecturas típicas

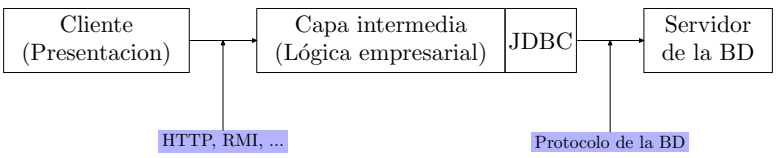
1.2. Arquitecturas típicas con JDBC

- El cliente se conecta directamente a la Base de Datos



La desventaja que tiene esta arquitectura es que la Base de Datos es directamente accesible y es complicado el control sobre el tipo de consultas que se pueden realizar.

- Arquitectura básica de tres capas:



La ventaja es que la Base de Datos queda oculta tras la capa intermedia. Es mucho más fácil controlar el tipo de consultas (ya que estas se realizan desde la capa intermedia sobre la que se tiene pleno control).

1.3. Una base de datos de ejemplo con MySQL

- El sistema gestor de base de datos que se va a utilizar es MySQL
- La biblioteca con las clases del controlador es el fichero JAR que se encuentra en la siguiente ruta dentro de la instalación de MySQL:
  `/lib/jdbc`
- He creado una base de datos llamada `Libros`
- He creado un usuario con nombre `lptel` y password `lptelclave` que tiene los siguientes permisos: lectura, escritura, borrado de datos y de tablas para la base de datos `Libros`.

La secuencia de instrucciones ha sido:

```
mysql -u usuario -p -b
```

Lanzo el cliente mysql (que está en el directorio `bin` de la instalación de MySQL). Utilizo un usuario con permisos de administrador. La opción `-p` indica que se pida un password y la opción `-b` indica que no produzca un sonido cuando se produzcan errores.

```
mysql>create database Libros;
```

Se crea una Base de Datos con nombre `Libros`

```
mysql>grant select,insert,update,alter,create,delete,drop on Libros.* to lptel identified
    by 'lptelclave';
```

Se un usuario cuyo nombre es `lptel` y password `lptelclave` con permiso para seleccionar registros (select), insertar registros (insert), actualizar registros (update), cambiar la estructura de una tabla (alter), crear tablas (create), borrar registros de una tabla (detele) borrar tablas (drop) en cualquier tabla de la base de datos `Libros`

```
mysql>quit
```

Se sale del cliente mysql

```
mysql -u lptel -p -b
```

Se lanza el cliente utilizando el nuevo usuario.

```
mysql>use Libros;
```

Se indica que se desea utilizar la Base de Datos `Libros`

```
mysql>source creaTablas.sql;
```

Se indica que se deben ejecutar las sentencias SQL contenidas en el fichero `creaTablas.sql`

```
mysql>source insertaValores.sql;
```

Indico que deseo ejecutar las sentencias SQL contenidas en el fichero `insertaValores.sql`

El fichero `creaTablas.sql` contiene instrucciones SQL para la creación de tablas:

```
create table Autores (
   CodigoAutor int(5) unsigned NOT NULL,
   Nombre varchar(80) NOT NULL,
   Direccion varchar(80) NOT NULL,
   Telefono varchar(20) NOT NULL,
   PRIMARY KEY (CodigoAutor)
)TYPE=InnoDB;

create table Libros (
   CodigoLibro int(5) unsigned NOT NULL,
   ISBN varchar(20) NOT NULL,
   Titulo varchar(80) NOT NULL,
   Editorial varchar(20) NOT NULL,
   Fecha date NOT NULL,
   PRIMARY KEY (CodigoLibro)
)TYPE=InnoDB;

create table AutorLibro (
   CodAutor int(5) unsigned NOT NULL,
   CodLibro int(5) unsigned NOT NULL,
   PRIMARY KEY  ( CodAutor , CodLibro ),
   KEY c_a (CodAutor),
   KEY c_l (CodLibro),
   FOREIGN KEY (CodAutor) REFERENCES Autores (CodigoAutor) ON UPDATE CASCADE ON DELETE CASCADE
      ,
   FOREIGN KEY (CodLibro) REFERENCES Libros (CodigoLibro) ON UPDATE CASCADE ON DELETE CASCADE
)TYPE=InnoDB;
```

El fichero `insertaValores.sql` contiene instrucciones SQL para insertar valores en las tablas creadas:

```
INSERT INTO Autores VALUES (1,'CAY S. HORSTMANN','Valencia','12345678');
```

```
INSERT INTO Autores VALUES (2,'GARY CORNELL','Castellon','21345678');
INSERT INTO Autores VALUES (3,'DAVID REILLY','Alicante','32145678');
INSERT INTO Autores VALUES (4,'MICHAEL REILLY','Elche','43215678');
INSERT INTO Autores VALUES (5,'BRUCE ECKEL','Benidorm','54321678');



INSERT INTO Libros VALUES (1,'0-201-71037-4','Java Network Programming and Distributed
    Computing','Addison Wesley','2002-01-01');
INSERT INTO Libros VALUES (2,'84-205-3701-2','Java 2. Volumen 2. Características Avanzadas','
    Pearson Educación','2003-01-01');
INSERT INTO Libros VALUES (3,'0-131-00287-2','Thinking in Java (3rd Edition)','Prentice Hall
    PTR','2002-01-01');
INSERT INTO Libros VALUES (4,'0-130-35313-2','Thinking in C++, Vol. 2: Practical Programming,
    Second Edition','Prentice Hall','2003-01-01');



INSERT INTO AutorLibro VALUES (1,2);
INSERT INTO AutorLibro VALUES (2,2);
INSERT INTO AutorLibro VALUES (3,1);
INSERT INTO AutorLibro VALUES (4,1);
INSERT INTO AutorLibro VALUES (5,3);
INSERT INTO AutorLibro VALUES (5,4);
```

2. Programacion con JDBC

Los pasos a realizar para utilizar JDBC en una aplicación son los siguientes:

1. Cargar la clase que representa al controlador
2. Establecer una conexión
3. Ejecución de sentencias SQL
4. Obtención de los resultados
5. Cierre de la conexión

2.1. La clase con el controlador y el URL a la Base de Datos

En el paquete java.sql se define la interfaz

```
public interface Driver
```

Esta es la interfaz que deben implementar los controladores JDBC.

Lo primero que hemos de hacer es ver cual es la clase que implementa a esta interfaz en el controlador JDBC que vamos a utilizar:

- Por ejemplo, en el controlador proporcionado por MySQL la clase es com.mysql.jdbc.Driver
- En el controlador proporcionado para la conexión con DB2 la clase a utilizar es COM.ibm.db2.jdbc.app.DB2Driver).

Una vez identificado el controlador debe ser registrado para poder ser utilizado en la aplicación.

Existen dos posibilidades:

- Como un argumento pasado a la máquina virtual desde la línea de órdenes

```
java -Djdbc.drivers=com.mysql.jdbc.Driver Aplicacion
```

- En el código de la aplicación mediante la siguiente sentencia:

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

El siguiente paso es construir una URL a la base de datos. Esta URL se utiliza para especificar la máquina en la que se encuentra la base de datos, el puerto, la base de datos a utilizar, el usuario, el password,...

El formato es el siguiente:

```
jdbc:nombre:otros elementos
```

Estos otros elementos dependen del controlador.

Por ejemplo, el formato de URL para MySQL sería el siguiente:

```
jdbc:mysql://[host][:port]/[database][?propertyName1][=propertyValue1][&propertyName2][=
    propertyValue2]...
```

El URL a la base de datos creada anteriormente sería (para una conexión local):

```
jdbc:mysql://localhost:3306/Libros?user=lptel&password=lptelclave
```

2.2. Realización de la conexión

Una vez que se ha registrado la clase del controlador y se dispone de la URL se puede una instancia de la clase Connection.

Para ello se utiliza el método estático getConnection() de la clase DriverManager.[1]

[1] La interfaz DataSource, nueva en JDBC 2.0, proporciona otra forma de conectarse con una fuente de datos. El uso de un objeto DataSource es el método recomendado para conectarse a una fuente de datos desde una aplicación web.

```
public static Connection getConnection(String url) throws SQLException
```

Cuando se llama a este método, el DriverManager intentará localizar un controlador apropiado entre los que se hayan registrado.

La siguiente porción de código muestra cómo se puede realizar esto:

```
...
try {

   String url = "jdbc:mysql://localhost:3306/Libros?user=lptel&password=lptelclave";

   Connection conn = DriverManager.getConnection(url);

   // Realizar algo con la conexion

   ...
} catch (SQLException ex) {
   // Mostrar errores
   System.out.println("SQLException: " + ex.getMessage());
   System.out.println("SQLState: " + ex.getSQLState());
   System.out.println("VendorError: " + ex.getErrorCode());
}
...
```

2.3. Obtención de metadatos

Mediante la interfaz DatabaseMetaData es posible obtener información del sistema gestor de bases de datos o de una determinada base de datos.

Esta clase dispone de entre otros (consúltese el API para un listado completo) los siguientes métodos:

```
public String getDatabaseProductName()
   //Nombre del SGBD

public String getDatabaseProductVersion()
   //Versión del SGBD

public String getDriverName()
   //Nombre del controlador

public String getDriverVersion()
   //Versión del controlador

public ResultSet getCatalogs() throws SQLException
   //Listado de las bases de datos existentes

public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern,
    String[] types)
  //Listado de las tablas en una base de datos
  Ejemplo: referencia.getTables("Libros","%","%",null);}


public ResultSet getTableTypes() throws SQLException
  //Listado con los tipos de tablas soportadas por el SGBD
```

A continuación se muestra un ejemplo completo:

```
import java.sql.*;

public class Metadatos{

   public static void main(String[] args){

      try{
         String URL = "jdbc:mysql://localhost:3306/Libros?user=lptel&password=lptelclave";

         Connection conn = DriverManager.getConnection(url);

         DatabaseMetaData meta = conn.getMetaData();

         System.out.println("El SGBD es:");

         System.out.println(meta.getDatabaseProductName());
         System.out.println(meta.getDatabaseProductVersion());

         conn.close();

      }catch(SQLException ex) {
         System.out.println("SQLException: " + ex.getMessage());
         System.out.println("SQLState: " + ex.getSQLState());
         System.out.println("VendorError: " + ex.getErrorCode());
      }

   }
}
```

2.4. Ejecución de sentencias SQL

La ejecución de sentencias en la base de datos a través de JDBC se realiza mediante las interfaces Statement o PreparedStatement.

Los objetos de estos tipos se obtienen a partir del objeto de tipo Connection

2.4.1. La interfaz Statement

Mediante objetos de este tipo se pueden ejecutar sentencias SQL sencillas y obtener los resultados mediante la clase ResulSet.

Para obtener un objeto del tipo Statement se llama al método createStatement() del objeto Connection.

Una vez que se dispone del objeto se pueden ejecutar sentencias SELECT (que no modifican las tablas) utilizando el método

```
Resulset executeQuery(String SQL)
```

Para ejecutar sentencias que contengan UPDATE, INSERT o DELETE hay que utilizar el método

```
int executeUpdate(String SQL)
```

Este método devuelve el número de filas afectadas por la sentencia.

Si no se conoce de antemano qué contiene la sentencia SQL se puede utilizar el método

```
boolean execute(String SQL)
```

Este método devuelve true si la sentencia contenía un SELECT o false en caso contrario.

Si la sentencia contenía un SELECT se pueden obtener los resultados llamando al método getResultSet() que devuelve un objeto del tipo ResulSet.

En caso contrario se puede obtener el número de filas afectadas llamando al método getUpdateCount().

Ejemplos:

```
...
Statement s = DriverManager.getConnection(url).createStatement();
ResulSet rs;
String sentenciaSQL = "SELECT * FROM Libros";
rs = s.executeQuery(sentenciaSQL);
```

```
...
Statement s = DriverManager.getConnection(url).createStatement();
int filasMod;
String sentenciaSQL = "UPDATE Libros SET Fecha='2004-01-01' WHERE CodigoLibro=1;";
filasMod = s.executeUpdate(sentenciaSQL);
```

```
...
Statement s = DriverManager.getConnection(url).createStatement();
boolean tipoSentencia;
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
sentenciaSQL = br.readLine();
tipoSentencia = s.execute(sentenciaSQL);

if (tipoSentencia){
   ResulSet rs = s.getResultSet();
   ...
}else
   int filasMod = s.getUpdateCount();
   ...
}
```

2.4.2. La interfaz PreparedStatement

La interfaz PreparedStatement extiende a la interfaz Statement y utiliza una plantilla para crear la sentencia SQL. Se utiliza para enviar sentencias SQL precompiladas con uno o más parámetros.

Se crea un objeto PreparedStatement especificando la plantilla y los lugares donde irán los parámetros.

Los parámetros son después especificados utilizando los métodos setXXX(.) indicando el número de parámetro y el dato a insertar en la sentencia.

La sentencia SQL y los parámetros se envían a la base de datos cuando se llama al método executeXXX(.)

PreparedStatement de consulta

Por ejemplo supongamos que hay un campo de texto en el que el usuario puede introducir su dirección de correo electrónico y con este dato se desea buscar al usuario:

```
...
Connection con = DriverManager.getConnection(url);
String consulta = "SELECT usuario FROM registro WHERE email like ?");
PreparedStatement pstmt = con.prepareStatement(consulta);

pstmt.setString(1,campoTexto.getText());
ResultSet resultado = ps.executeQuery();
...
```

PreparedStatement de modificación

En el siguiente ejemplo se va a insertar un nuevo registro en una tabla

```
...
Connection con = DriverManager.getConnection(url);
String insercion = "INSERT INTO registro(usuario,email,fechaNac) values (?,?,?)");
PreparedStatement pstmt = con.prepareStatement(consulta);

String user = ...;
String email = ...;
Date edad = ...;

pstmt.setString(1,user);
pstmt.setString(2,email);
pstmt.setDate(3,edad);

ps.executeUpdate();
...
```

2.5. Obtención de los resultados

Como hemos visto en los ejemplos anteriores el resultado de una consulta es devuelto en un objeto del tipo ResultSet.

Podemos imaginar que el resultado se devuelve en forma de tabla donde las filas corresponden a los registros y las columnas a los datos.

Primero hay que colocarse en una determinada fila y a continuación acceder a la columna deseada.

Para ello, la clase ResultSet dispone de métodos para moverse en filas y de métodos para seleccionar una determinada columna.

Algunos de los métodos disponibles para recorrer los registros son[2]:

[2]Por defecto los objetos del tipo ResultSet no son actualizables y tienen un cursor que sólo puede ir hacia adelante. Por tanto, se puede iterar a lo largo de él una sola vez y desde el inicio hasta el final. Es posible crear ResultSet que se puedan actualizar y/o que permitan libertad de movimientos. Se verá más adelante

```
void beforeFirst()
```

Posición por defecto, coloca el cursor antes del primer resultado.

```
void first()
```

Coloca el cursor en la primera fila del resultado

```
void afterLast()
```

Coloca el cursor en la después de la última fila del resultado.

```
void last()
```

Coloca el cursor en la última fila del resultado.

```
boolean next()
```

Avanza el cursor una posición.

```
boolean previous()
```

Retrocede el cursor una posición.

Para seleccionar columnas una vez que nos hemos colocado en un determinado registro se dispone de dos conjuntos de métodos.

Por un lado métodos que reciben un entero (que indica el número de la columna) y por otro métodos que reciben el nombre de la columna.

Por ejemplo si se desea obtener el valor de una columna que es de tipo float se dispone de los dos métodos:

```
float getFloat(int numeroColumna)
float getFloat(String nombreColumna)
```

Si el valor de la columna es de tipo int entonces disponemos de

```
int getInt(int numeroColumna)
int getInt(String nombreColumna)
```

Se pueden obtener metadatos relacionados con el ResultSet utilizando el método getMetaData() que devuelve un objeto del tipo ResultSetMetaData

Algunos de los métodos disponibles en esta clase son:

```
int getColumnCount()
```

Devuelve el número de columnas que hay en el ResultSet.

```
String getColumnName(int column)
```

Obtiene el nombre de la columna cuyo número de orden se pasa como parámetro.

```
String getColumnTypeName(int column)
```

Obtiene el tipo de dato que hay en una determinada columna.

Ejemplo:

```
...
Connection con = DriverManager.getConnection(url);
Statement s = con.createStatement();
ResulSet resultado;
String sentenciaSQL = "SELECT * FROM Libros";
resultado = s.executeQuery(sentenciaSQL);

while (resultado.next()){
   String tit = resultado.getString("Titulo");
   java.sqlDate fech = resultado.getDate("Fecha");
   System.out.println("TITULO: " + tit);
   System.out.println("FECHA: " + fech);
}
...
```

Equivalencia tipos de datos SQL y Java

Tipo de dato SQL	Tipo de dato Java
INTEGER o INT	int
SMALLINT	short
NUMERIC(m,n), DECIMAL(m,n) o DEC(m,n)	java.sql.Numeric
FLOAT(n)	double
REAL	float
DOUBLE	double
CHARACTER(n) o CHAR(n)	String
VARCHAR(n)	String
BOOLEAN	boolean
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BLOB	java.sql.Blob
ARRAY	java.sql.Array

Ojo, puede que alguno de los tipos de la izquierda no estén soportados por todas las Bases de Datos (y puede que añadan alguno más).

Las versiones anteriores a JDBC 2.0 devolvían los resultados con cursores que sólo podían ir hacia adelante. Cada elemento se obtenía llamando al método next().

A partir de JDBC 2.0 se puede recorrer el resultado en las dos direcciones y se puede actualizar el resultado *si esta operación está soportada por la base de datos.*

Una ventaja es que se pueden actualizar una serie de columnas sin necesidad de enviar ninguna llamada executeUpdate(). La actualización se realiza de forma automática.

A la hora de obtener tanto un Statement como un PreparedStatement es posible especificar (si la base de datos lo acepta) el tipo de desplazamiento deseado y si se desea que las modificaciones en el resultado se reflejen en la base de datos.

```
// Supongo que conn es del tipo Connexion

// Para crear un Statement
Statement stmt = conn.createStatement(int tipoResultado, int tipoActualizacion);

// Para crear un PreparedStatement
PreparedStatement pstmt = conn.prepareStatement(String sql, int tipoResultado, int
    tipoActualizacion);
```

El valor para el tipo de resultado puede ser:

- ResultSet.TYPE_FORWARD_ONLY sólo se puede avanzar
- ResultSet.SCROLL_SENSITIVE se puede recorrer en cualquier dirección y las actualizaciones se reflejan en cuanto se producen.
- ResultSet.SCROLL_INSENTITIVE se puede recorrer en cualquier dirección pero las actualizaciones no son visibles hasta que no vuelva a realizar una consulta.

El valor para el tipo de actualización puede ser

- ResultSET.CONCUR_READ_ONLY el resultado es únicamente de lectura
- ResultSET.CONCUR_UPDATABLE el resultado es acutalizable

Es posible verificar si la Base de Datos soporta estos tipos utilizando el siguiente método

```
// Supongo que conn es del tipo Connexion
int tipoResultado = ResultSet....;
int tipoActualizacion = ResultSet....;
boolean esPosible = conn.getMetaData().supportsResultSetConcurrency(tipoResultado,
    tipoActualizacion);
```

En cuanto a la actualización del resultado se pueden utilizar los métodos updateXXX(.) (donde `XXX` es el tipo de dato a actualizar) que ofrece la interfaz ResultSet.

A continuación se envía el mensaje updateRow() al objeto del tipo ResultSet para que actualice la fila en la base de datos.

Por ejemplo, el siguiente código se puede utilizar para actualizar el valor de la fecha en el resultado y actualizar la base de datos:

```
// Asumo que rs es una referencia del tipo ResultSet

rs.absolute(5); // mueve el cursor a la fila 5

// actualiza la fecha en el ResultSet
rs.updateDate("Fecha", "2004-04-01");

// Actualiza la fecha en la base de datos para el
//registro seleccionado
rs.updateRow();
```

Finalmente antes de abandonar la aplicación se cierra la conexión utilizando el método close() de Connection.