



Índice

1. Introducción	1
1.1. JDBC	2
1.2. Arquitecturas típicas con JDBC	3
2. Programación con JDBC	4
2.1. La clase con el controlador y el URL a la Base de Datos	4
2.2. Realización de la conexión	6
2.3. Obtención de metadatos	6
2.4. Ejecución de sentencias SQL	8
2.4.1. La interfaz Statement	8
2.4.2. La interfaz PreparedStatement	9
2.5. Obtención de los resultados	10

1. Introducción

¿Qué vamos a necesitar?

-  Un sistema gestor de base de datos que permita la conexión mediante JDBC (MySQL)
-  Las clases del driver para JDBC
-  Una plataforma Java y su documentación (J2SE 1.4.2)
-  Un editor para Java (Eclipse 2.1.3 que es más que un editor para Java...)

¿Donde se pueden obtener?

-  J2SE: java.sun.com
-  Eclipse: www.eclipse.org
-  MySQL: www.mysql.org

Obtención de las clases que representan el driver.

En la dirección:

<http://dev.mysql.com/downloads/connector/j/3.0.html>

Se obtiene el archivo

mysql-connector-java-3.0.14-production.zip

Al descomprimirlo aparece (entre otros) el archivo

```
mysql-connector-java-3.0.12-production.jar
```

Este fichero debe estar en un sitio visible para Java tanto al compilar la aplicación como al ejecutarla.

```
javac -classpath mysql-connector-java-3.0.12-production.jar;. Aplicacion.java
java -classpath mysql-connector-java-3.0.12-production.jar;. Aplicacion
```

1.1. JDBC

JDBC: Java DataBase Connectivity (Conexión con Bases de Datos mediante Java).

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

Permite realizar operaciones (consultas, actualizaciones, ...) sobre bases 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...).

Los controladores JDBC se pueden clasificar como:

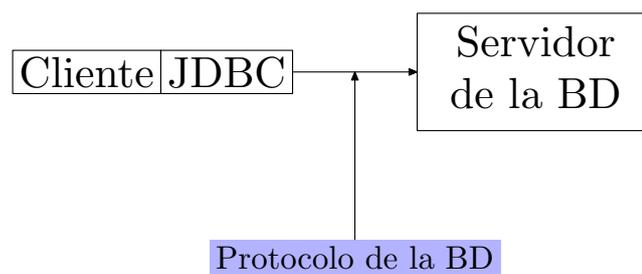
-  Controladores tipo 1: traducen JDBC a ODBC y se delega en ODBC para la comunicación con la base de datos. Sun incluye uno de estos controladores con el J2SE.
-  Controlador tipo 2: está escrito parcialmente en Java y en código nativo.
-  Controlador tipo 3: es una biblioteca cliente escrita completamente 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.

Resumiendo:

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

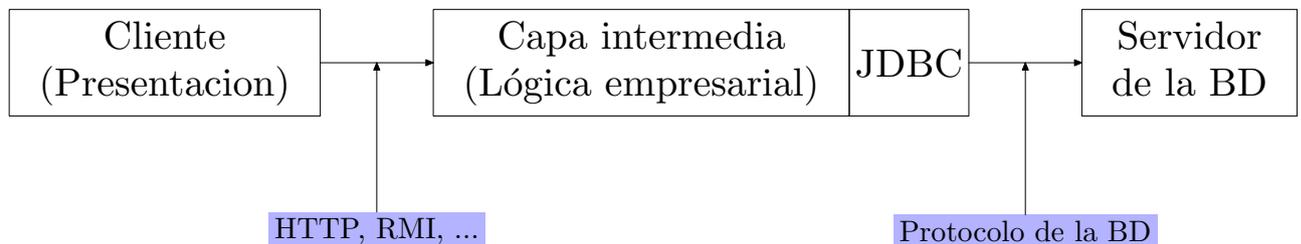
1.2. Arquitecturas típicas con JDBC

- 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).

2. Programacion con JDBC

Los pasos a realizar para utilizar JDBC en una aplicación Java 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. Cerrar 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 cualquier controlador que utilicemos debe implementar.

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");
```

Por supuesto, al inicio de nuestra aplicación debe aparecer (al menos) la siguiente biblioteca de clases:

```
import java.sql.*;
```

El siguiente paso es construir una URL a la base de datos para JDBC.

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][?propName1]=[propValue1][&propName2]=[propValue2]...
```

Ejemplos:

```
jdbc:mysql://localhost:3306/Libros?user=usuario&password=clave
```

Se especifica que la base de datos está en la máquina local, que el puerto donde escucha mysql es el 3306, que la base de datos a utilizar es Libros, que el usuario es usuario y que el password es clave



```
jdbc:mysql://epi.uv.es:3306/Viajes?user=admin&password=admin48
```

Se especifica que la base de datos está en la máquina cuyo nombre es epi.uv.es, que el puerto donde escucha mysql en esa máquina es el 3306, que la base de datos a utilizar es Viajes, que el usuario es admin y que el password es admin48

2.2. Realización de la conexión

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

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

```
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 {  
    Class.forName("com.mysql.jdbc.Driver");  
    String url = "jdbc:mysql://localhost:3306/estancias?user=usuario&password=clave";  
    Connection conn = DriverManager.getConnection(url);  
    // Realizar algo con la conexion  
    ...  
} catch (SQLException ex) {  
    // handle any errors  
    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.

¹La 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.



Utilizando esta clase se podrían obtener las capacidades de un determinado sistema gestor de bases 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("estancias", "%", "%", 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/estancias?user=usuario&password=clave";  
  
            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());  
        }  
  
    }  
}
```



Las instrucciones para compilar y ejecutar han sido²

```
javac -classpath mysql_jdbc.jar;. Metadatos.java  
java -Djdbc.drivers=com.mysql.jdbc.Driver -classpath mysql_jdbc.jar;. Metadatos
```

Ejercicio 1

Realizar una conexión con la base de datos de otra máquina y mostrar la siguiente información:

- El nombre del SGBD
- La versión
- El nombre del controlador
- La versión del controlador

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 `ResultSet`.

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

```
ResultSet executeQuery(String SQL)
```

El resultado de la consulta se devuelve en un objeto del tipo `ResultSet`, es decir, para acceder a la información habrá que utilizar los métodos de esta clase.

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.

²Nota: he renombrado el fichero jar con el controlador para que tenga un nombre más corto.

Puede haber situaciones donde no se conoce de antemano si la sentencia SQL es de consulta o de modificación (por ejemplo si se permite que el usuario introduzca la sentencia). En estos casos se utiliza el método:

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

Este método devuelve `true` si la sentencia contenía un `SELECT` y `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 `ResultSet`.

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

Ejemplos:

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

```
...
Statement s = DriverManager.getConnection(url).createStatement();
int filasMod;
String sentenciaSQL = " update casas set DispHasta='2004-11-30' where IdCasa=7";
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){
    ResultSet 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 en aquellas situaciones en las que se necesita ejecutar varias veces una consulta en la que pueden cambiar únicamente los parámetros. De esta forma (si el SGBD lo permite) se optimiza la consulta una sola vez.

Par obtener un objeto del tipo `PreparedStatement` se llamará al método `prepareStatement(.)` del objeto del tipo `Connection`.

A este método hay que pasarle la plantilla donde se especifican 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 se ejecuta cuando se llama a los métodos `executeQuery()` o `executeUpdate()`

————— 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 son³:

```
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 ()
```

³Por 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

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 getColumnName(int column)
```

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

Ejemplo:

```
import java.sql.*;
public class EjemploResultSet{
```



```
public static void main(String [] args){
    try{
        String url = "jdbc:mysql://localhost:3306/estancias?user=usuario&password=clave";

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

        Connection con = DriverManager.getConnection(url);
        Statement s = con.createStatement();
        ResultSet resultado;
        String sentenciaSQL = "SELECT Ciudad,DispDesde,DispHasta FROM casas";
        resultado = s.executeQuery(sentenciaSQL);

        ResultSetMetaData rsmd = resultado.getMetaData();

        System.out.println();
        System.out.println("Numero de columnas: " + rsmd.getColumnCount());
        System.out.println("Nombre de la primera columna: " + rsmd.getColumnName(1));
        System.out.println("Tipo de la primera columna: " + rsmd.getColumnTypeName(1));
        System.out.println();

        resultado.beforeFirst();

        while (resultado.next()){
            String ciu = resultado.getString("Ciudad");
            java.sql.Date fech1= resultado.getDate("DispDesde");
            java.sql.Date fech2= resultado.getDate("DispHasta");
            System.out.print("Casa en: " + ciu);
            System.out.print(" Desde: " + fech1);
            System.out.print(" Hasta: " + fech2);
            System.out.println();
        }

        con.close();

    } catch (SQLException ex) {
        System.out.println("SQLException: " + ex.getMessage());
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("VendorError: " + ex.getErrorCode());
    } catch (ClassNotFoundException e){
        System.out.println("No se encuentra el controlador");
    }
}
}
```

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).

Ejercicio 2

Realizar una consulta en la que se muestren las familias del Reino Unido donde el precio de estancia por día sea menor o igual a 18 euros y que muestre por pantalla el nombre de la familia, la ciudad y el tipo de casa.

Ejercicio 3

Debido a la devaluación de la libra esterlina respecto al euro se desea incrementar en un 5 por ciento el precio por día de todas las casas del Reino Unido. Realizar la actualización de los precios.

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 Conexion
// 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.TYPE_SCROLL_SENSITIVE` se puede recorrer en cualquier dirección y las actualizaciones se reflejan en cuanto se producen.
-  `ResultSet.TYPE_SCROLL_INSENSITIVE` 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 actualizable

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

```
// Supongo que conn es del tipo Conexion
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 updateTIPO(.) (donde TIPO 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 modificar 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();
```

Ejercicio 4

Realizar la actualización de los precios del ejercicio 3 del siguiente modo:

Crear un Statement en el que se permita el movimiento en cualquier dirección y en el que las actualizaciones se reflejan en cuanto se producen y además que el resultado sea actualizable.

Ejecutar una sentencia de consulta en la que se seleccione el país y el precio de todas las casas.

Recorrer el ResultSet obtenido mostrando el país y el precio y si además el país es el Reino Unido actualizar el precio por día subiéndolo un 5 por ciento.

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