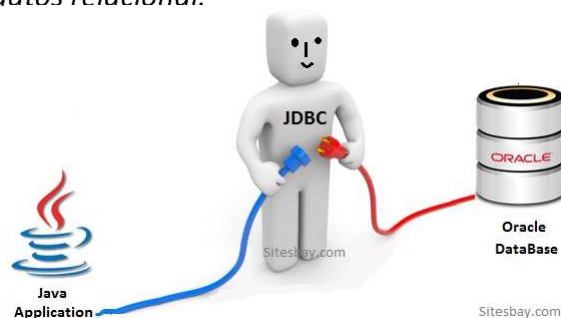


ACCESO A BASES DE DATOS RELACIONALES

INTRODUCCIÓN

2

- En esta unidad veremos el acceso a los datos de una base de datos relacional desde programas Java.
- Para ello se necesitan **conectores**: *software que se necesita para realizar las conexiones desde nuestro programa Java a la base de datos relacional.*



EL DESFASE OBJETO RELACIONAL

3

- En esta unidad trabajaremos con los problemas del desfase objeto relacional.
 - No podemos almacenar objetos en una tabla de una base de datos relacional.
 - En su lugar almacenaremos cada uno de los datos miembro de un objeto, uno a uno.
 - Al escribir en la base de datos tendremos que descomponer un objeto en cada uno de sus datos. Al leer de la base de datos, leeremos varias columnas y de cada fila podremos montar un objeto con los datos leídos.
- En la unidad 3 usaremos herramientas de mapeo objeto-relacional que hacen que al programador le parezca que está grabando y leyendo objetos en la BD.






BASES DE DATOS EMBEBIDAS

4

- Se utilizarán si no se almacenan grandes cantidades de información.
- El motor de almacenamiento está inscrustado en la aplicación y es exclusivo para ella.
- La BD se inicia al comenzar la aplicación y termina cuando se cierra la aplicación.
- Casi todas son OpenSource
- Una **base de datos embebida** es un sistema de gestión de bases de datos (SGBD) que está integrado con la aplicación software que requiere acceso a los datos

BASES DE DATOS EMBEBIDAS RELACIONALES

5

SQLite	
Apache Derby	
H2	
Firebird	
SQL Server CE	

JDBC

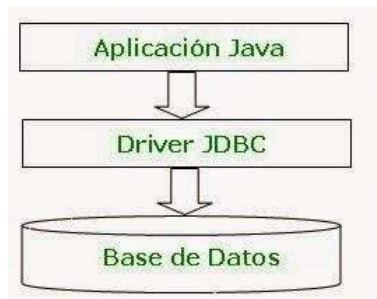
6

- Son las siglas de Java Database Connectivity
- 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
- Se encuentra dentro del paquete java.sql
- JDBC nos permite escribir aplicaciones Java para gestionar las siguientes tareas con una base de datos relacional:
 - ▣ **Conectarse** a la base de datos (local o remota)
 - ▣ **Enviar consultas e instrucciones** de actualización a la base de datos
 - ▣ **Recuperar y procesar** los resultados recibidos de la base de datos en respuesta a las consultas

Drivers

7

- Llamamos Drivers al conjunto de clases que implementan las interfaces JDBC
- El driver proporciona la comunicación entre la aplicación Java y la base de datos



Drivers

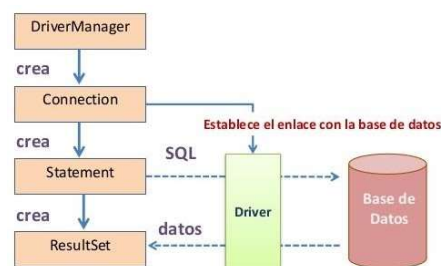
8

- Cada tipo de bases de datos (Oracle, MySQL, PostgreSQL, etc) tienen su propio driver.
- Los drivers los proporcionan los fabricantes de las bases de datos. Normalmente se descarga desde la página web del fabricante.
- Por lo tanto, el primer paso para trabajar con bases de datos desde Java es conseguir el driver adecuado.
- En nuestro caso vamos a cargar el driver que requiere para conectar con la base de datos de MySQL.
 - ▣ <https://dev.mysql.com/downloads/windows/installer/8.0.html>

Componentes del JDBC

9

- ❑ 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`)
- ❑ Sentencias preparadas (`java.sql.PreparedStatement`)
- ❑ El resultado (`java.sql.ResultSet`)

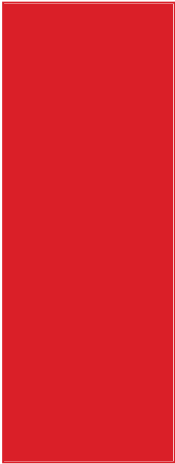


10

El objeto `Statement` (sentencia) sirve para procesar una sentencia SQL estática y obtener los resultados producidos por ella.

Las `PreparedStatement` amplían la interfaz `Statement` y proporcionan soporte para añadir parámetros a sentencias SQL..

11



```

import java.sql.Connection; import
java.sql.DriverManager; import
java.sql.SQLException; import
java.util.logging.Level; import
java.util.logging.Logger;

```

12

```

public class Conexion {

    String BD ="viajes"; //nombre de la base de datos
    String URL_BD = "jdbc:mysql://localhost:3306/" + BD; //URL conexion BD
    String LOGIN = "root"; //usuario de la BD
    String PASS= ""; //Contraseña
    public Connection conexion;

    public Conexion(){}

    public Connection conexionMySQL(){
        conexion = null;
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conexion = DriverManager.getConnection(URL_BD, LOGIN, PASS);
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException ex) {
            System.err.println("Error driver JDBC. " + ex.toString());
            Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            System.err.println("Error de conexión a la BD");
            Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE, null, ex);
        }
        return conexion;
    }
}

```

13

Ya tenemos una clase llamada conexión con un método llamado: `conexionMySQL()`, que nos devuelve el identificador de la conexión.

Desde cualquier otra clase podemos usarla así:

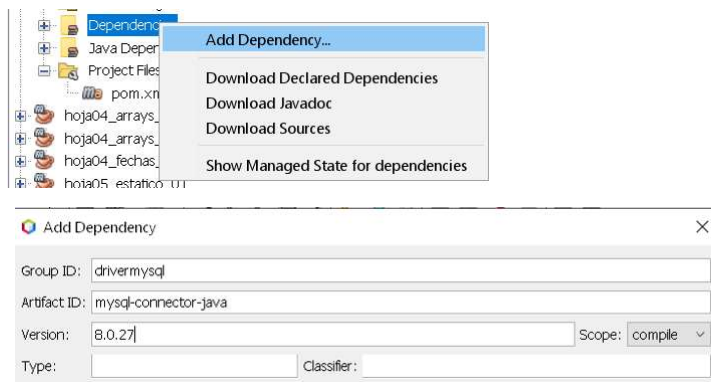
```
Conexion conexion = new Conexion();
```

```
Connection con = conexion.conexionMySQL();
```

Añadir el Driver al proyecto Java Maven

14

- El JAR del conector para esta versión se denomina `mysql-connector-java-8.0.*.jar`
- Abrir el proyecto e ir al directorio **Dependencies**
- Y situados en ese directorio con botón derecho aparece la opción



Añadir el Driver al proyecto Java Maven

15

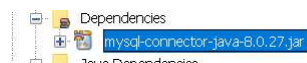
- Una vez añadido aparece:
- Situado en el conector con botón derecho aparece la opción **Manually install artifact**
- Seleccionar donde se encuentra el Jar, si es una instalación por defecto de MySQL
- C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connector-java-8.0.27.jar



Añadir el Driver al proyecto Java Maven

16

- Una vez instalado desaparece la exclamación
- En el output sale reflejado
- Y en el fichero POM.XML, inserta la dependencia
- Para los siguientes proyectos sólo tenemos que añadir la dependencia ya que el driver que instalado localmente



```
<dependency>
  <groupId>drivermysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.27</version>
</dependency>
```


Paso 1 – Establecer conexión

17

Código para conectarnos a la BD, previamente arrancado el SGBD y creada la BD

Patrón Singleton

Ejemplo1

```
public class Conexion {
    private static Connection conn;
    private static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String USER = "conexion";
    private static final String PASS = "";
    private static final String BD = "Eutaxia";
    private static final String DB_URL = "jdbc:mysql://localhost:3306/"+BD;

    private static class ConexionPoseedor {
        public static final Conexion INSTANCE = new Conexion();
    }

    private Conexion() {
        try {
            Class.forName(JDBC_DRIVER);
            Properties properties = new Properties();
            properties.setProperty("user", USER);
            properties.setProperty("password", PASS);
            properties.setProperty("useSSL", "false");
            properties.setProperty("autoReconnect", "true");
            conn = (Connection) DriverManager.getConnection(DB_URL, properties);
            if (conn != null) {
                System.out.println("Conexión a la base de datos "+DB_URL+".....CORRECTA");
            }
        } catch (SQLException ex) {
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("ErrorCode: " + ex.getErrorCode());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        }
    }
}
```

Ejemplo

18

Ejemplo Gráfico Conexión

Paso 2 – Crear y ejecutar sentencias

19

- Para enviar sentencias SQL al controlador de la BD se utiliza el objeto **Statement**, suministrando el método SQL con la sentencia a ejecutar.
 - ▣ `Statement sentencia=conn.createStatement();`
- Para sentencias insert, delete, update, create o create table, el método a utilizar es **executeUpdate()**
 - ▣ `Statement sentencia= con.createStatement();`
 - ▣ `sentencia.executeUpdate ("CREATE TABLE CAFES " + "(CAFE_NAME VARCHAR(32), " + "SUP_ID INT, PRECIO FLOAT, " + "DESCUENTO INT, "+ "TOTAL INT)");`

Ejemplo

20

```
public void crearTablas () {
    try {
        Statement sentencia=conn.createStatement();
        // en String tabla codigo sql con el create table
        String tabla="create table ejemplo(\n" +
            "    codigo smallint NOT NULL AUTO_INCREMENT,\n" +
            "    producto varchar(25) NOT NULL,\n" +
            "    cantidad int NOT NULL,\n" +
            "    PRIMARY KEY (`codigo`)\n" +
            ") ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8mb4 ; ";
        sentencia.executeUpdate(tabla);

        System.out.println("Tablas creadas con éxito!!");
        sentencia.close();

    } catch (SQLException ex) {
        System.out.println("Error al ejecutar la sentencia");
    }
}
```

Paso 2 – Crear y ejecutar sentencias

21

Cuando ejecutas insert, update y delete, que también se hace con executeUpdate(), devuelve el número de filas afectadas.

```
public void insertarDatos() {
    try {
        Statement sentencia=conn.createStatement();
        // dentro de executeUpdate codigo del insert, update o delete
        int resul=sentencia.executeUpdate("INSERT INTO ejemplo"
            + "(producto,cantidad)"
            + "VALUES ('melocotones',8),"
            + "('platanos',12),('peras',3);");
        System.out.println("Filas afectadas: "+resul);
        sentencia.close();
    } catch (SQLException ex) {
        System.out.println("Error en la inserción de datos");
    }
}
```

Paso 3 – Recuperar y procesar los resultados

22

- Se utiliza el método **executeQuery()** para sentencias SELECT

```
public void mostrarDatos() {
    Statement sentencia;
    try {
        sentencia = conn.createStatement();
        // dentro de executeQuery Codigo de la select
        ResultSet rs = sentencia.executeQuery("select codigo,producto,cantidad"
            + "from ejemplo");
        while (rs.next()) {
            //cada columna se indica, el tipo en el get, y que posicion o
            //que nombre tiene en el argumento
            System.out.print(rs.getInt(1) + " ");
            System.out.print(rs.getString("producto") + " ");
            System.out.println(rs.getInt(3));
        }
        rs.close();
        sentencia.close();
    } catch (SQLException ex) {
        System.out.println("Error en la consulta");
    }
}
```

Paso 3 – Recuperar y procesar los resultados

23

- JDBC devuelve los datos en un objeto **ResultSet**, donde se almacenarán los datos obtenidos de la consulta.
- Para obtener cada uno de los datos recuperados de la consulta se usará el método **next**, que permitirá ir posicionándonos en cada una de las filas devueltas.
- Con los métodos **getXXX** obtendremos cada uno de los campos de la fila.

Paso 3 – Recuperar y procesar los resultados

24

- El objeto **ResultSet** tiene entre otros los siguientes métodos:
 - **next()**; // accede al siguiente registro y devuelve false cuando no hay más registros
 - **first()**; // accede al primero
 - **previous()**; // al anterior
 - **last()**; // al último
 - **getInt("campo")** ó **getInt(indice columna)**; devuelve el contenido numérico entero de la columna
 - **getDouble("campo")** ó **getDouble(indice columna)**; // devuelve el contenido numérico double de la columna
 - **getString("campo")** ó **getString(indice columna)**; // devuelve el contenido de la cadena de la columna etc.....
 - **getObject(indice columna)**; // devuelve un objeto en general de cualquier tipo.
- NOTA : los índices de columna siempre comienzan por 1.

Sentencias Preparadas

25

Cuando se vayan a ejecutar varias consultas similares es más apropiado utilizar objetos **PreparedStatement** que hereda de la clase **Statement**, ya que se reducirá el tiempo de ejecución. Cuando se vaya a usar, basta con reemplazar los parámetros

```
try {
    System.out.println("Introduce la cantidad:");
    int cantidad = new Scanner(System.in).nextInt();

    //Consulta preparada
    String sql = "SELECT codigo,producto,cantidad from ejemplo "
        + "where cantidad > ? ";
    PreparedStatement ps = conn.prepareStatement(sql);
    // indico que para el primer parámetro el valor de la nota
    ps.setInt(1, cantidad);
    ResultSet rs = ps.executeQuery();

    System.out.println("Productos con cantidad > que : "
        + cantidad+ "\n");
    while (rs.next()) {
        System.out.println(rs.getString("producto")
            + " con cantidad: "
            + rs.getInt(3));
    }
    rs.close();
    conn.close();
} catch (SQLException ex) {
    LOG.log(Level.SEVERE, null, ex);
}
```