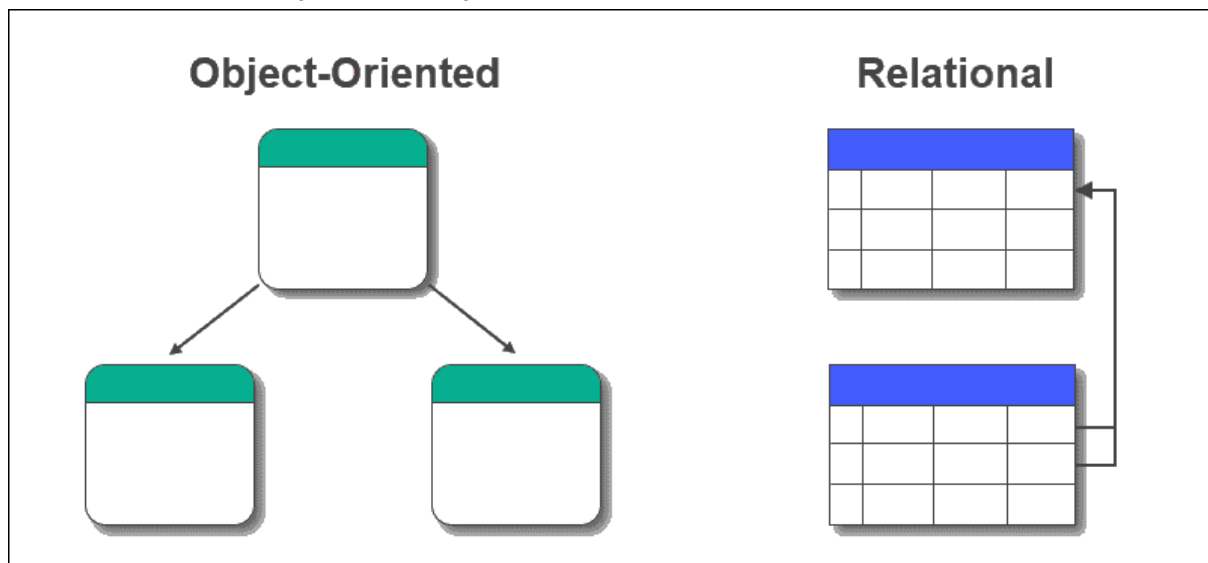


# UD 2

## MANEJO DE CONECTORES

### El desfase objeto-relacional

Con el auge de la programación orientada a objetos desde finales de los años ochenta se pusieron de manifiesto las dificultades, tanto conceptuales como técnicas que plantea el almacenamiento de objetos complejos en bases de datos relacionales.



Para referirse a estas dificultades se acuñó el término **desfase objeto-relacional** (en inglés **object-relational impedance mismatch**) también llamada **impedancia objeto-relacional**.

Estos aspectos se puede presentar en cuestiones como:

Paradigma Orientado a Objetos	Paradigma Relacional
Los <b>datos</b> están <b>encapsulados</b> detrás de un conjunto de métodos de acceso	Los <b>datos</b> se manejan <b>directamente</b>
Los objetos tienen <b>interfaces</b> que en conjunto proporcionan el único acceso a las partes internas de ese objeto	El modelo utiliza <b>vistas</b> para proporcionar diferentes perspectivas y <b>restricciones</b> para asegurar la integridad
Las clases pueden tener <b>asociaciones</b> (un objeto tiene un atributo que es otro objeto)	Las tablas pueden tener <b>interrelaciones</b> (a través de claves ajenas)
No existe el concepto de <b>transacción</b>	Permite el control de <b>transacciones</b>

Permite la <b>herencia</b> y el <b>polimorfismo</b>	No existen los conceptos de <b>herencia</b> y <b>polimorfismo</b>
Los <b>objetos no solamente</b> almacenan <b>datos</b> en sus <b>atributos</b> : pueden almacenar también colecciones de objetos o interrelaciones entre objetos	Las <b>filas</b> de las tablas <b>solamente</b> almacenan <b>datos</b> en sus <b>columnas</b>
Una <b>consulta</b> devuelve una <b>colección de objetos</b>	Una <b>consulta</b> devuelve un <b>conjunto de filas</b>

Sin embargo, a pesar de todas estas dificultades los SGBD relacionales siguen siendo los más utilizados con mucha diferencia sobre los demás.

## Protocolos de acceso a bases de datos.

### Conectores

- Los **SGBD** (relacionales, documentales, orientados a objetos, ...) → tienen sus propios **lenguajes especializados** para operar con los **datos** que almacenan
- Las **aplicaciones** → se desarrollan en **lenguajes de programación de propósito general** (Java, Python, C#, ...)

Para que los programas de aplicación puedan interactuar con los SGBD se necesitan mecanismos que permitan a los programas de aplicación comunicarse con las bases de datos de estos lenguajes → estos mecanismos se denominan **conectores**

### Conectores de bases de datos relacionales

- Cada **SGBD** relacional implementa
  - su propia **versión** del lenguaje SQL
  - su propia **interfaz** de acceso
- Existen **arquitecturas de conectores** (ODBC, JDBC, ...) que implementan interfaces para **diferentes SGBD** relacionales

- La primera arquitectura de conectores a SGBD relacionales fue **ODBC** (Open Data Base Connectivity)
  - fue desarrollado por **Microsoft** a principios de los **años noventa** para el **lenguaje C**
  - actualmente está implementada para sistemas Windows, Linux y Unix
- Otros conectores para SGBD relacionales, documentales y otros para **Windows** son:
  - **OLE-DB** (Object Linking and Embedding for Databases)
  - **ADO** (ActiveX Data Objects)

- **PHP** posee para poder implementar la arquitectura **LAMP** (Linux, Apache, MySQL, PHP)
  - la extensión **MySQLi** que proporciona
    - una API para **MySQL**
    - una API para **BBDD orientadas a objetos**
  - una API llamada **PDO** (Portable Data Objects) con *drivers* para
    - MySQL
    - Oracle
    - PostgreSQL, ...
- A finales de los **años noventa** surgió **JDBC** (Java Data Base Connectivity) como la arquitectura equivalente a ODBC para el lenguaje de programación Java
  - La API de la arquitectura de conectores JDBC está compuesta por una serie de **clases** contenidas en el paquete **java.sql**
  - Existe una implementación diferente de la **JDBC para cada SGBD relacional**
  - Existe también una implementación de **JDBC** para conectarse a **ODBC** (por si acaso ese SGBD no tuviese implementado su conector JDBC)

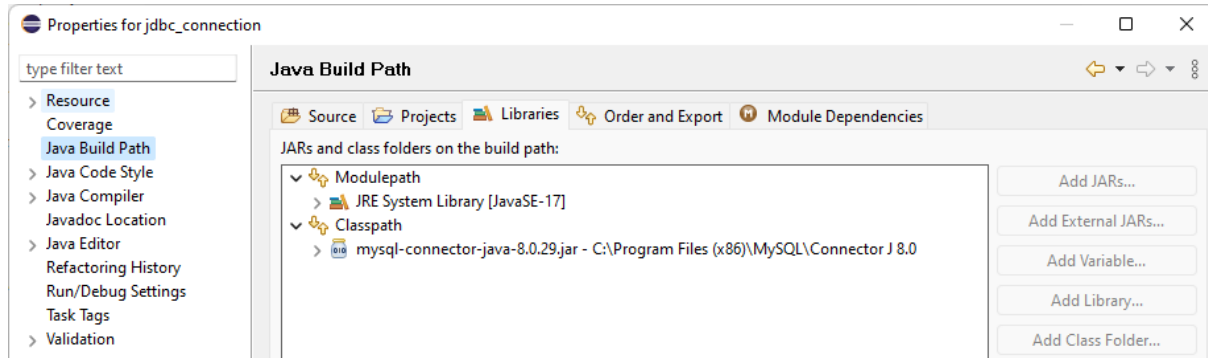
**¡RECUERDA!**

- **Todos los SGBD relacionales importantes** actuales implementan **ODBC y JDBC**
- Existe **drivers JDBC para ODBC** → permiten conectar **Java** con un **SGBD relacional** si no existe un conector JDBC para ese SGBD relacional

# Establecimiento de conexiones

Debemos indicar a nuestro entorno de desarrollo donde se encuentra el driver **JDBC**

**Ejemplo:** para Eclipse sobre Windows con MySQL 8.0 (el driver forma parte de la instalación del SGBD)



Para la conexión a una BDD se usa un método de la clase **java.sql.DriverManager**:

- **getConnection**(String *URL\_conexión*, String *usuario*, String *contraseña*)
  - devuelve un objeto que implementa la interfaz **Connection**
  - puede producir la excepción **SQLException**

Para MySQL la *URL\_conexión* valdrá ***jdbc:mysql://host:puerto/base\_de\_datos***

## **Nota:**

**Connection** implementa **AutoCloseable**

- se puede inicializar en un **try con recursos**
- si no se cierra en el try con recursos entonces debe cerrarse con **close()**

**Ejemplo 1:** Creamos la clase **JDBC\_Connection** que se conecta a la BDD **world** de un servidor **MySQL** en mi máquina local con el usuario **root**

```
JDBC_Connection.java X
1 package jdbc_connection;
2
3 import java.sql.DriverManager;
4 import java.sql.Connection;
5 import java.sql.SQLException;
6
7 public class JDBC_Connection {
8
9     public static void muestraErrorSQL(SQLException e) {
10         System.out.println("SQL ERROR mensaje: " + e.getMessage());
11         System.out.println("SQL Estado: " + e.getSQLState());
12         System.out.println("SQL código específico: " + e.getErrorCode());
13     }
14
15     public static void main(String[] args) {
16         String basedatos = "world";
17         String host = "localhost";
18         String port = "3306";
19         String urlConnection = "jdbc:mysql://" + host + ":" + port + "/" + basedatos;
20         String user = "root";
21         String pwd = "vistaalegre";
22
23         try (Connection c=DriverManager.getConnection(urlConnection, user, pwd)) {
24             System.out.println("Conexión realizada.");
25         } catch (SQLException e) {
26             muestraErrorSQL(e);
27         } catch (Exception e) {
28             e.printStackTrace(System.err);
29         }
30     }
31
32 }
```

Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage

<terminated> JDBC\_Connection [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 11:38:53 - 11:38:54) [pid: 21624]

Conexión realizada.

Si probamos con el servicio **MySQL** detenido → **error 0** y **SQLState 08S01**

Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage

<terminated> JDBC\_Connection [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 11:38:53 - 11:38:54) [pid: 21624]

SQL ERROR mensaje: Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.

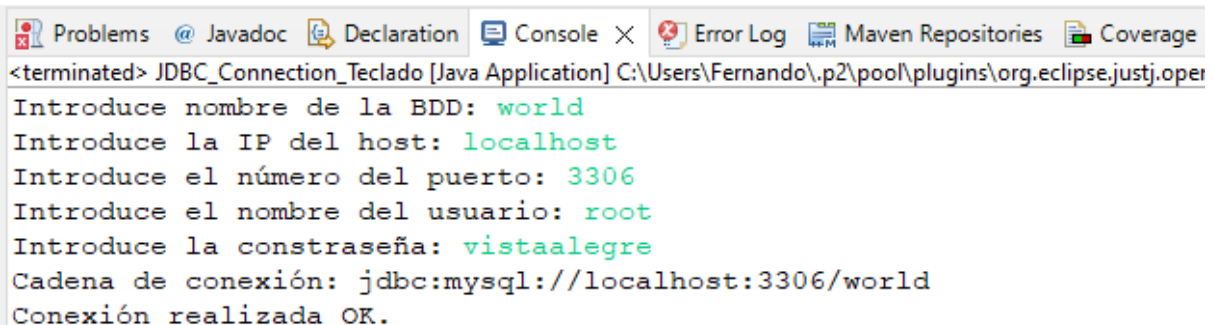
SQL Estado: 08S01

SQL código específico: 0

**Ejemplo 2:** Modificamos la clase anterior para introducir por teclado las diferentes partes de la cadena de conexión

```
1 package jdbc_connection;
2
3 import java.sql.DriverManager;
4 import java.sql.Connection;
5 import java.sql.SQLException;
6 import java.util.Scanner;
7
8 public class JDBC_Connection_Teclado {
9
10     public static void muestraErrorSQL(SQLException e) {
11         System.out.println("SQL ERROR mensaje: " + e.getMessage());
12         System.out.println("SQL Estado: " + e.getSQLState());
13         System.out.println("SQL código específico: " + e.getErrorCode());
14     }
15
16     public static void main(String[] args) {
17         //Inicializamos el escáner
18         Scanner scanner = new Scanner(System.in);
19
20         //Pedimos los datos del estudiante
21         System.out.print("Introduce nombre de la BDD: ");
22         String basedatos = scanner.nextLine();
23         System.out.print("Introduce la IP del host: ");
24         String host = scanner.nextLine();
25         System.out.print("Introduce el número del puerto: ");
26         String port = scanner.nextLine();
27         System.out.print("Introduce el nombre del usuario: ");
28         String user = scanner.nextLine();
29         System.out.print("Introduce la contraseña: ");
30         String pwd = scanner.nextLine();
31
32         String urlConnection = "jdbc:mysql://" + host + ":" + port + "/" + basedatos;
33         System.out.println("Cadena de conexión: " + urlConnection);
34
35         try (Connection c = DriverManager.getConnection(urlConnection, user, pwd)) {
36             System.out.println("Conexión realizada OK.");
37         } catch (SQLException e) {
38             muestraErrorSQL(e);
39         } catch (Exception e) {
40             e.printStackTrace(System.err);
41         } finally {
42             scanner.close();
43         }
44     }
45 }
```

I. Probamos que con los parámetros correctos se realiza la conexión:



```
<terminated> JDBC_Connection_Teclado [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.open
Introduce nombre de la BDD: world
Introduce la IP del host: localhost
Introduce el número del puerto: 3306
Introduce el nombre del usuario: root
Introduce la contraseña: vistaalegre
Cadena de conexión: jdbc:mysql://localhost:3306/world
Conexión realizada OK.
```

## II. Probamos qué ocurre si no existe la BDD → **error 1049 y SQLState 42000**

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> JDBC_Connection_Teclado [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 3:17:35 - 3:18:00)
Introduce nombre de la BDD: word1
Introduce la IP del host: localhost
Introduce el número del puerto: 3306
Introduce el nombre del usuario: root
Introduce la contraseña: vistaalegre
Cadena de conexión: jdbc:mysql://localhost:3306/word1
SQL ERROR mensaje: Unknown database 'word1'
SQL Estado: 42000
SQL código específico: 1049
```

## III. Probamos qué ocurre si el puerto es incorrecto → **error 0 y SQLState 08S01**

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> JDBC_Connection_Teclado [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 3:17:35 - 3:18:00)
Introduce nombre de la BDD: world
Introduce la IP del host: localhost
Introduce el número del puerto: 3309
Introduce el nombre del usuario: root
Introduce la contraseña: vistaalegre
Cadena de conexión: jdbc:mysql://localhost:3309/world
SQL ERROR mensaje: Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
SQL Estado: 08S01
SQL código específico: 0
```

## IV. Probamos qué ocurre si la IP es incorrecta → **error 0 y SQLState 08S01**

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> JDBC_Connection_Teclado [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 11:31:36 - 11:32:00)
Introduce nombre de la BDD: world
Introduce la IP del host: 192.168.1.34
Introduce el número del puerto: 3306
Introduce el nombre del usuario: root
Introduce la contraseña: vistaalegre
Cadena de conexión: jdbc:mysql://192.168.1.34:3306/world
SQL ERROR mensaje: Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
SQL Estado: 08S01
SQL código específico: 0
```

## V. Probamos qué ocurre si el usuario es incorrecto → **error 1045 y SQLState 28000**

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> JDBC_Connection_Teclado [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 11:31:36 - 11:32:00)
Introduce nombre de la BDD: world
Introduce la IP del host: localhost
Introduce el número del puerto: 3306
Introduce el nombre del usuario: rut
Introduce la contraseña: vistaalegre
Cadena de conexión: jdbc:mysql://localhost:3306/world
SQL ERROR mensaje: Access denied for user 'rut'@'localhost' (using password: YES)
SQL Estado: 28000
SQL código específico: 1045
```

## VI. Probamos qué ocurre si la contraseña es incorrecta → **error 0 y SQLState 08S01**

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> JDBC_Connection_Teclado [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_18.0.1.v20220515-1614\jre\bin\javaw.exe (11 oct 2022 11:31:36 - 11:32:
Introduce nombre de la BDD: world
Introduce la IP del host: 192.168.1.34
Introduce el número del puerto: 3306
Introduce el nombre del usuario: root
Introduce la contraseña: vistaalegre
Cadena de conexión: jdbc:mysql://192.168.1.34:3306/world
SQL ERROR mensaje: Communications link failure

The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
SQL Estado: 08S01
SQL código específico: 0
```

**Ejemplo 3:** Podemos crear una clase **Conector\_JDBC** que implementa el método estático **Conexion** que permita a otras clases conectarse a la BDD

```
1 package jdbc_connection;
2
3 import java.sql.DriverManager;
4 import java.sql.Connection;
5 import java.sql.SQLException;
6
7 public class Conector_JDBC {
8
9     public static void muestraErrorSQL(SQLException e) {
10         System.out.println("SQL ERROR mensaje: " + e.getMessage());
11         System.out.println("SQL Estado: " + e.getSQLState());
12         System.out.println("SQL código específico: " + e.getErrorCode());
13     }
14
15     public static void muestraConexionOK() {
16         System.out.println("Conexión realizada OK.");
17     }
18
19     public static Connection Conexion(String basedatos, String host, String port,
20                                     String user, String pwd)
21     throws SQLException {
22         String urlConnection = "jdbc:mysql://" + host + ":" + port + "/" + basedatos;
23         return DriverManager.getConnection(urlConnection, user, pwd);
24     }
25
26 }
```

Vemos que la clase **Prueba\_conexion\_OK** se conecta correctamente a la BDD

```
Prueba_conexion_OK.java X
3 import java.sql.Connection;
4 import java.sql.SQLException;
5
6 public class Prueba_conexion_OK {
7
8     public static void main(String[] args) {
9         try (Connection con = Conector_JDBC.Conexion("world", "localhost", "3306", "root", "vistaalegre")) {
10             Conector_JDBC.muestraConexionOK();
11         } catch (SQLException e) {
12             Conector_JDBC.muestraErrorSQL(e);
13         } catch (Exception e) {
14             System.err.println(e);
15         }
16     }
17
18 }
```

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> Prueba_conexion_OK [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_18.0.1.v20220515-1614\jre\bin\javaw.exe (12 c
Conexión realizada OK.
```



Vemos que la clase **Prueba\_conexion\_KO** no puede conectarse a una BDD llamada **wordl**

```
Prueba_conexion_KO.java X
1 package jdbc_connection;
2
3 import java.sql.Connection;
4 import java.sql.SQLException;
5
6 public class Prueba_conexion_KO {
7
8     public static void main(String[] args) {
9         try (Connection con = Conector_JDBC.Conexion("wordl", "localhost", "3306", "root", "vistaalegre")) {
10             Conector_JDBC.muestraConexionOK();
11         } catch (SQLException e) {
12             Conector_JDBC.muestraErrorSQL(e);
13         } catch (Exception e) {
14             System.err.println(e);
15         }
16     }
17 }
18 }
19 }
```

Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage

<terminated> Prueba\_conexion\_KO [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.1.v20220515-1614\jre\bin\javaw.exe (12 oct)

SQL ERROR mensaje: Unknown database 'wordl'

SQL Estado: 42000

SQL código específico: 1049

**NOTA:** Conectarse mediante la clase **Connection** tiene algunos problemas

1. Es muy **inseguro** que el **usuario** y **contraseña** sean visibles en el código de una aplicación
    - se puede utilizar la clase **Properties** para ocultar las claves de acceso <https://javiergarciaescobedo.es/programacion-en-java/15-ficheros/358-archivo-de-propiedades-properties>
  2. Abrir una conexión a una BDD supone un consumo importante de tiempo y de recursos → si no se cierran adecuadamente todas las conexiones la BDD puede funcionar muy lentamente
    - Es más eficiente disponer de un **pool de conexiones**: un conjunto de conexiones siempre disponibles que se van repartiendo entre los diferentes procesos que la demanden
- Ejemplos:
- i. **Apache Commons DBCP** (Database connection pooling services) → disponible para Java SE (versión estándar)
  - ii. Servidor de aplicaciones **Apache Tomcat**
  - iii. Servidor de aplicaciones **BlueFish**

**Java EE** posee para aplicaciones corriendo en un **servidor de aplicaciones** opciones de conexión a BBDD más seguras y sofisticadas

- más simple sintácticamente
- más general (vale para otros sistemas de almacenamiento además de SGBD relacionales)
- permite **transacciones distribuidas**
- proporciona **pools de conexiones**

## Métodos de la interfaz **Connection**

Algunos métodos de **java.sql.Connection** son:

<b>void close()</b>	cierra la conexión
<b>void commit()</b>	valida la transacción en curso
<b>Statement createStatement()</b>	crea una sentencia SQL
<b>boolean getAutoCommit()</b>	devuelve <b>true</b> si está en modo <b>autocommit</b>
<b>CallableStatement prepareCall(String sql)</b>	crea una sentencia preparada
<b>PreparedStatement prepareStatement (String sql)</b>	crea una sentencia para llamar a procedimientos o funciones almacenados
<b>void rollback()</b>	anula la transacción en curso
<b>void setAutoCommit(boolean autoCommit)</b>	fija el valor de <b>autocommit</b>

## Ejecución de consultas

Para ejecutar consultas se necesita

- un objeto que implemente la interfaz **Statement** del paquete **java.sql** para crear la consulta → debe cerrarse explícitamente con **close()**
- un objeto de la clase **ResultSet** del paquete **java.sql** para recoger el resultado de la consulta (todas las filas y columnas) → debe cerrarse explícitamente con **close()**
  - usamos el método **createStatment()** de **Connection** para crear el objeto **Statement**
  - usamos el método **executeQuery()** de **Statement** para ejecutar la consulta y obtener el **ResultSet**
  - usamos el método **next()** de **ResultSet** para obtener secuencialmente cada fila de la consulta

**Nota:** si se cierra explícitamente el **Statement** → se cierra automáticamente el **ResultSet** (no hace falta cerrarlo explícitamente)

**Ejemplo:**

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("select * from departments");
```



# Utilización del resultado de una consulta

Para obtener las columnas de una consultas usamos los métodos accesorios de **ResultSet**

- **getString**(String nombre\_columna)
- **getString**(int numero\_columna)
- **getDate**(String nombre\_columna)
- **getDate**(int numero\_columna)
- **getInt**(String nombre\_columna)
- **getInt**(int numero\_columna)
- **getLong**(String nombre\_columna)
- **getLong**(int numero\_columna)
- ...

**Ejemplo 1:** seleccionamos todas las filas de la tabla **departments** y extraemos las columnas por su nombre

```
JDBC_select_por_nombre.java X
1 package JDBC_select;
2
3 import java.sql.Connection;
4 import java.sql.Statement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import jdbc_connection.Connector_JDBC;
8
9 public class JDBC_select_por_nombre {
10
11     public static void main(String[] args) {
12         try (Connection con = Connector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
13             Statement stmt = con.createStatement();
14             ResultSet rs = stmt.executeQuery("select * from departments");
15             while (rs.next()) {
16                 System.out.print("dept_no: " + rs.getString("dept_no") + " | ");
17                 System.out.println("dept_name: " + rs.getString("dept_name"));
18             }
19             // Cerramos el Statement --> cierra automaticamente el ResultSet
20             stmt.close();
21         } catch (SQLException e) {
22             Connector_JDBC.muestraErrorSQL(e);
23         } catch (Exception e) {
24             System.err.println(e);
25         }
26     }
27
28 }
```

Problems | Javadoc | Declaration | Console X | Error Log | Maven Repositories | Coverage

<terminated> JDBC\_select\_por\_nombre [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.1.v20220515-1614\jre\bin\javaw.exe (13 oct 2022)

dept_no: d009		dept_name: Customer Service
dept_no: d005		dept_name: Development
dept_no: d002		dept_name: Finance
dept_no: d003		dept_name: Human Resources
dept_no: d001		dept_name: Marketing
dept_no: d004		dept_name: Production
dept_no: d006		dept_name: Quality Management
dept_no: d008		dept_name: Research
dept_no: d007		dept_name: Sales

**NOTA:** vemos que como no hemos puesto cláusula **ORDER BY** entonces no se obtienen las filas ordenadas en el **ResultSet**

**Ejemplo 2:** seleccionamos todas las filas de la tabla **departments** ordenadas por nombre del departamento y extraemos las columnas por su número

```
JDBC_select_por_numero.java ×
1 package JDBC_select;
2
3 import java.sql.Connection;
4 import java.sql.Statement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import jdbc_connection.Conector_JDBC;
8
9 public class JDBC_select_por_numero {
10
11     public static void main(String[] args) {
12         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
13             Statement stmt = con.createStatement();
14             ResultSet rs = stmt.executeQuery("select dept_no, dept_name from departments order by 1");
15             while (rs.next()) {
16                 System.out.print("dept_no: " + rs.getString(1) + " | ");
17                 System.out.println("dept_name: " + rs.getString(2));
18             }
19             // Cerramos el Statement --> cierra automaticamente el ResultSet
20             stmt.close();
21         } catch (SQLException e) {
22             Conector_JDBC.muestraErrorSQL(e);
23         } catch (Exception e) {
24             System.err.println(e);
25         }
26     }
27
28 }
```

Problems @ Javadoc Declaration Console × Error Log Maven Repositories Coverage

<terminated> JDBC\_select\_por\_numero [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.1.

dept_no: d001		dept_name: Marketing
dept_no: d002		dept_name: Finance
dept_no: d003		dept_name: Human Resources
dept_no: d004		dept_name: Production
dept_no: d005		dept_name: Development
dept_no: d006		dept_name: Quality Management
dept_no: d007		dept_name: Sales
dept_no: d008		dept_name: Research
dept_no: d009		dept_name: Customer Service

**NOTA:** vemos que como hemos puesto cláusula **ORDER BY** entonces se obtienen las filas ordenadas en el **ResultSet**

## Scrollable

# Ejecución de sentencias de descripción de datos

En ocasiones podemos necesitar obtener información (nombres de tablas, columnas, claves, ...) de una base de datos a la que estamos conectados.

## Clase DatabaseMetaData

El paquete `java.sql` proporciona la interfaz **DatabaseMetaData** que permite obtener información del diccionario de datos de la BDD a la cual estamos conectados.

Algunos métodos de **DatabaseMetaData** son:

<b>ResultSet</b> <code>getTables(String catalogo, String esquema, String patronDeTabla, String[] tipos)</code>  tipos → array con tipos de objetos que queremos obtener (TABLE, VIEW, ...)	devuelve un <i>ResultSet</i> con los datos de las <b>tablas</b> y <b>vistas</b> de la BDD → arroja <b>SQLException</b>
<b>ResultSet</b> <code>getColumns(String catalogo, String esquema, String patronNombreDeTabla, String patronNombreDeColumna)</code>	devuelve un <i>ResultSet</i> con los datos de las <b>columnas</b> de la BDD → arroja <b>SQLException</b>
<b>ResultSet</b> <code>getProcedures(String catalogo, String esquema, String patronNombreDeProcedimiento)</code>	devuelve un <i>ResultSet</i> con los datos de las <b>procedimientos almacenados</b> en la BDD → arroja <b>SQLException</b>
<b>ResultSet</b> <code>getPrimaryKeys(String catalogo, String esquema, String tabla)</code>	devuelve un <i>ResultSet</i> con los datos de las <b>columnas</b> que forman las claves primarias de <b>una tabla</b> → arroja <b>SQLException</b>
<b>ResultSet</b> <code>getExportedKeys(String catalogo, String esquema, String tabla)</code>	devuelve un <i>ResultSet</i> con los datos de las <b>claves ajenas</b> que usan una clave primaria de <b>la tabla</b> → arroja <b>SQLException</b>
<b>ResultSet</b> <code>getImportedKeys(String catalogo, String esquema, String tabla)</code>	devuelve un <i>ResultSet</i> con los datos de las <b>claves ajenas</b> <b>la tabla</b> → arroja <b>SQLException</b>
<b>String</b> <code>getDatabaseProductName()</code>	devuelve el nombre del SGBD → arroja <b>SQLException</b>
<b>String</b> <code>getDriverName()</code>	devuelve el nombre del <i>driver</i> → arroja <b>SQLException</b>
<b>String</b> <code>getURL()</code>	devuelve la cadena de

	conexión → arroja <b>SQLException</b>
<b>String</b> <code>getUserName()</code>	devuelve las claves de acceso a la BDD → arroja <b>SQLException</b>

### Nota:

- En algunos SGBD como **Oracle** una única BDD (catálogo) puede tener varios esquemas (subconjunto de objetos del catálogo)
- En algunos SGBD como **MySQL** cada BDD (catálogo) sólo puede tener un esquema  
→ el esquema será siempre **NULL**

### Ejemplo: implementamos la clase **EjemploDatabaseMetadata**

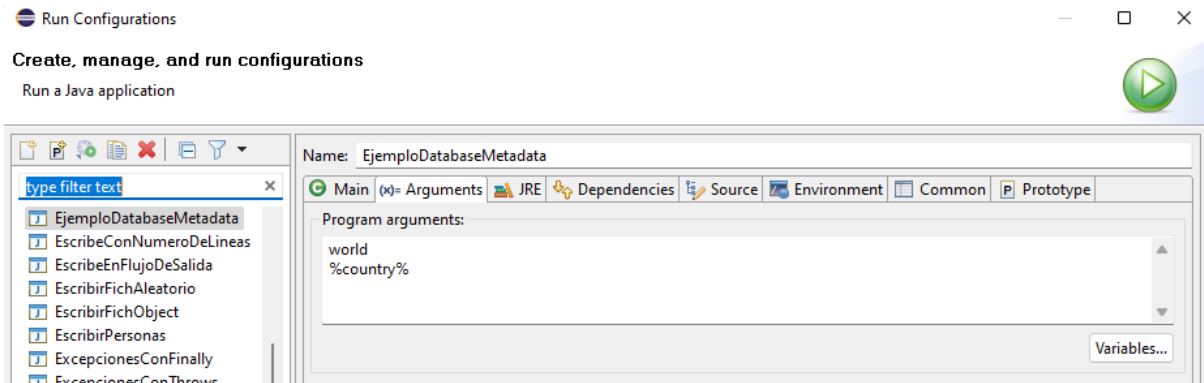
- obtenemos de un objeto de la clase **DatabaseMetaData** los datos de la conexión
- obtenemos del método **getTables()** del objeto de la clase **DatabaseMetaData** los nombres de las tablas de la base de datos **arg[0]** que contengan la palabra **arg[1]** (usamos el comodín de SQL '%')

```

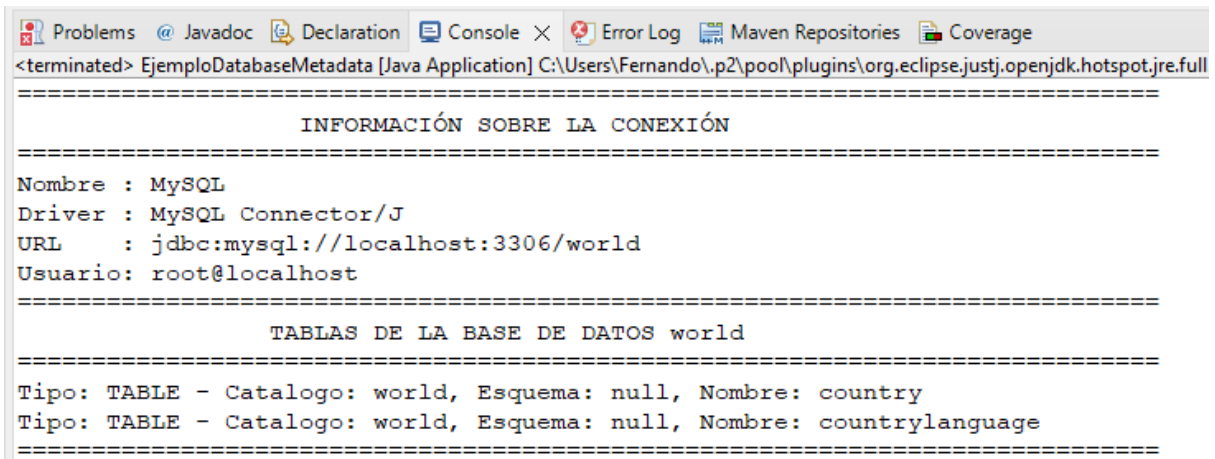
1 package JDBC_metadada;
2
3 import java.sql.*;
4 import jdbc_connection.Conector_JDBC;
5
6 public class EjemploDatabaseMetadata {
7     public static void main(String[] args) {
8         //String db = "world";
9         try (Connection con = Conector_JDBC.Conexion(args[0], "localhost", "3306", "root", "vistaalegre")) {
10             DatabaseMetaData dbmd = con.getMetaData();
11             ResultSet rs = null;
12             String nombre = dbmd.getDatabaseProductName();
13             String driver = dbmd.getDriverName();
14             String url = dbmd.getURL();
15             String usuario = dbmd.getUserName();
16
17             System.out.println("=====");
18             System.out.println("                INFORMACIÓN SOBRE LA CONEXIÓN");
19             System.out.println("=====");
20             System.out.printf("Nombre : %s %n", nombre);
21             System.out.printf("Driver : %s %n", driver);
22             System.out.printf("URL      : %s %n", url);
23             System.out.printf("Usuario: %s %n", usuario);
24             System.out.println("=====");
25             rs = dbmd.getTables(args[0], null, args[1], null);
26             System.out.printf("                TABLAS DE LA BASE DE DATOS %s %n", args[0]);
27             System.out.println("=====");
28             while (rs.next()) {
29                 String catalogo = rs.getString(1);
30                 String esquema = rs.getString(2);
31                 String tabla = rs.getString(3);
32                 String tipo = rs.getString(4);
33                 System.out.printf("Tipo: %s - Catalogo: %s, Esquema: %s, Nombre: %s %n", tipo, catalogo, esquema, tabla);
34             }
35             System.out.println("=====");
36             // Cerramos el ResultSet
37             rs.close();
38         } catch (SQLException e) {
39             Conector_JDBC.muestraErrorSQL(e);
40         } catch (Exception e) {
41             System.err.println(e);
42         }
43     }
44 }

```

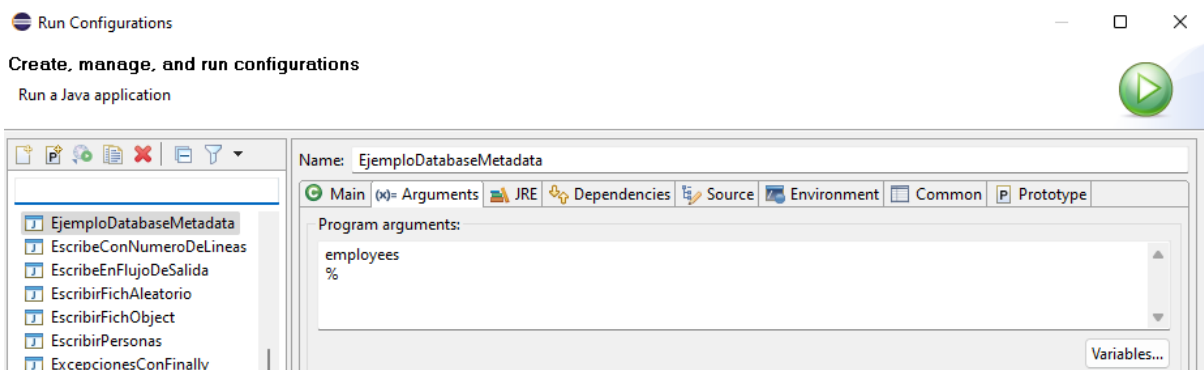
- **Prueba 1:** pasamos los parámetros al método **main()**
  - BDD → **world**
  - Nombre de la tabla → like '**%country%**'



Obtenemos dos tablas: **country** y **countrylanguage**



- **Prueba 2** → pasamos los parámetros al método **main()**
  - BDD → **employees**
  - Nombre de la tabla o vista → like '**%**' (todas las tablas y vistas)





Obtenemos el nombre de todas las tablas y vistas de la BDD **employees**

```
Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage
<terminated> EjemploDatabaseMetadata [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.
=====
                                INFORMACIÓN SOBRE LA CONEXIÓN
=====
Nombre : MySQL
Driver : MySQL Connector/J
URL    : jdbc:mysql://localhost:3306/employees
Usuario: root@localhost
=====
                                TABLAS DE LA BASE DE DATOS employees
=====
Tipo: TABLE - Catalogo: employees, Esquema: null, Nombre: departments
Tipo: TABLE - Catalogo: employees, Esquema: null, Nombre: dept_emp
Tipo: TABLE - Catalogo: employees, Esquema: null, Nombre: dept_manager
Tipo: TABLE - Catalogo: employees, Esquema: null, Nombre: employees
Tipo: TABLE - Catalogo: employees, Esquema: null, Nombre: salaries
Tipo: TABLE - Catalogo: employees, Esquema: null, Nombre: titles
Tipo: VIEW - Catalogo: employees, Esquema: null, Nombre: current_dept_emp
Tipo: VIEW - Catalogo: employees, Esquema: null, Nombre: dept_emp_latest_date
=====
```

## Clase ResultSetMetaData

Se pueden obtener metadatos de un objeto **ResultSet** (número de columnas devueltas, tipo de las columnas, nombre de las columnas, ...).

Para ello necesitamos el método **getMetadata()** de **ResultSet** que devuelve un objeto que implemente la interfaz **ResultSetMetaData**.

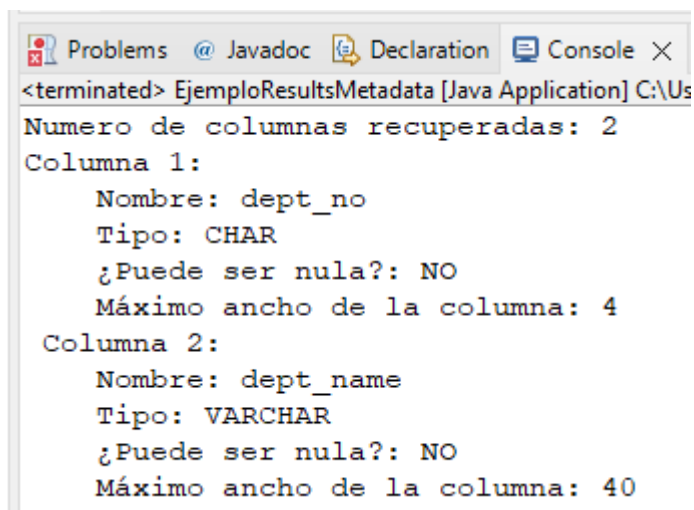
Algunos métodos de **ResultSetMetaData** son

<b>int</b> getColumnCount()	Devuelve el número de columnas devuelto por una consulta
<b>String</b> getColumnName (int indiceColumna)	Devuelve el nombre de la columna cuya posición se indica
<b>String</b> getColumnNameType (int indiceColumna)	Devuelve el tipo de dato de la columna cuya posición se indica
<b>int</b> isNullable(int indiceColumna)	Devuelve cero si la columna no puede tener valores nulos
<b>int</b> getColumnDisplaySize(int indiceColumna)	Devuelve el máximo ancho en caracteres de la columna cuya posición se indica

**Ejemplo:** en la BDD **employees** seleccionamos todas las filas y columnas de la tabla **departments**

```
1 package JDBC_metadada;
2
3 import java.sql.*;
4 import jdbc_connection.Conector_JDBC;
5
6 public class EjemploResultsMetadata {
7     public static void main(String[] args) {
8         //String db = "world";
9         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
10             Statement sentencia = con.createStatement();
11             ResultSet rs = sentencia.executeQuery("select * from departments");
12
13             ResultSetMetaData rsmd = rs.getMetaData();
14
15             int nColumnas = rsmd.getColumnCount();
16             String nula;
17             System.out.printf("Numero de columnas recuperadas: %d\n", nColumnas);
18
19             for (int i = 1; i <= nColumnas; i++) {
20                 System.out.printf("Columna %d: %n ", i);
21                 System.out.printf("    Nombre: %s %n    Tipo: %s %n ", rsmd.getColumnName(i), rsmd.getColumnTypeName(i));
22                 if (rsmd.isNullable(i) == 0)
23                     nula = "NO";
24                 else
25                     nula = "SI";
26                 System.out.printf("    ¿Puede ser nula?: %s %n ", nula);
27                 System.out.printf("    Máximo ancho de la columna: %d %n ", rsmd.getColumnDisplaySize(i));
28             }
29             // Cerramos el Statement --> cierra automaticamente el ResultSet
30             sentencia.close();
31         } catch (SQLException e) {
32             Conector_JDBC.muestraErrorSQL(e);
33         } catch (Exception e) {
34             System.err.println(e);
35         }
36     }
37 }
```

y obtenemos:



```
<terminated> EjemploResultsMetadata [Java Application] C:\Us
Numero de columnas recuperadas: 2
Columna 1:
    Nombre: dept_no
    Tipo: CHAR
    ¿Puede ser nula?: NO
    Máximo ancho de la columna: 4
Columna 2:
    Nombre: dept_name
    Tipo: VARCHAR
    ¿Puede ser nula?: NO
    Máximo ancho de la columna: 40
```

# Ejecución de sentencias de modificación de datos

## Ejecución de sentencias DDL

- Necesitamos el método **createStatment()** de **Connection** para crear un objeto **Statement**
- Necesitamos el método **execute()** de **Statement** para ejecutar la **sentencia DDL**
  - arroja la excepción **SQLException**

**Nota:** en la vida real no tiene sentido que la creación de objetos en una BDD se haga desde una aplicación

### Ejemplo 1: creamos la tabla **ELIMINAME** en la BDD **employees**

```
JDBC_create_table.java X
1 package JDBC_DDL;
2
3 import java.sql.Connection;
4 import java.sql.Statement;
5
6 import jdbc_connection.Conector_JDBC;
7
8 import java.sql.SQLException;
9
10 public class JDBC_create_table {
11
12     public static void main(String[] args) {
13         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
14             Statement stmt = con.createStatement();
15             String sentencia = null;
16             sentencia = "create table ELIMINAME (          \n";
17             sentencia += "  DNI          char(9) not null,  \n";
18             sentencia += "  APELLIDOS  varchar(32) not null, \n";
19             sentencia += "  CP          char(5),          \n";
20             sentencia += "  PRIMARY KEY (DNI)  )";
21             System.out.println (sentencia);
22             stmt.execute(sentencia);
23             stmt.close();
24             System.out.println ("Se ha creado OK");
25         } catch (SQLException e) {
26             Conector_JDBC.muestraErrorSQL(e);
27         } catch (Exception e) {
28             System.err.println(e);
29         }
30     }
31 }
32
33 }
```

Problems @ Javadoc Declaration Console Error Log Maven Repositories Coverage  
terminated> JDBC\_create\_table [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.1.v20220515-1614\jre\bin\javaw.exe (14 oct 2022 1:0)

```
create table ELIMINAME (
  DNI          char(9) not null,
  APELLIDOS    varchar(32) not null,
  CP           char(5),
  PRIMARY KEY (DNI) )
Se ha creado OK
```

Podemos comprobar en **Workbench** que se ha creado:

Query 1 X

Limit to 1000 rows

1 • desc ELIMINAME;

2

3

Result Grid Filter Rows: Export: Wrap Cell Content: [IA](#)

Field	Type	Null	Key	Default	Extra
DNI	char(9)	NO	PRI	NULL	
APELLIDOS	varchar(32)	NO		NULL	
CP	char(5)	YES		NULL	

**Ejemplo 2:** eliminamos la tabla **ELIMINAME** en la BDD **employees** y comprobamos con **getTables()** de **DatabaseMetaData** que se ha eliminado

```

1 package JDBC_DDL;
2
3 import java.sql.Connection;
4 import java.sql.DatabaseMetaData;
5 import java.sql.ResultSet;
6 import java.sql.Statement;
7 import jdbc_connection.Conector_JDBC;
8 import java.sql.SQLException;
9
10 public class JDBC_drop_table {
11     public static void muestra(Connection con, String momento, String bdd, String nomTabla) throws SQLException {
12         DatabaseMetaData dbmd = con.getMetaData();
13         ResultSet rs = null;
14         String[] tipos = {"table"};
15         System.out.println("=====");
16         rs = dbmd.getTables(bdd, null, null, tipos);
17         System.out.printf ("          TABLAS DE LA BASE DE DATOS %s %s %n", bdd, momento);
18         System.out.println("=====");
19         while (rs.next()) {
20             String catalogo = rs.getString(1);
21             String tabla = rs.getString(3);
22             String tipo = rs.getString(4);
23             System.out.printf("Tipo: %s - Catalogo: %s, Nombre: %s %n", tipo, catalogo, tabla);
24         }
25         System.out.println("=====");
26         // Cerramos el ResultSet
27         rs.close();
28     }
29     public static void main(String[] args) {
30         String bdd = "employees";
31         String tabla = "ELIMINAME";
32         try (Connection con = Conector_JDBC.Conexion(bdd, "localhost", "3306", "root", "vistaalegre")) {
33             muestra(con, "ANTES", bdd, tabla);
34             Statement stmt = con.createStatement();
35             String sentencia = "drop table " + tabla;
36             System.out.println ("Sentencia:" + sentencia);
37             stmt.execute(sentencia);
38             stmt.close();
39             System.out.println ("Se ha eliminado OK");
40             muestra(con, "DESPUES", bdd, tabla);
41         } catch (SQLException e) {
42             Conector_JDBC.muestraErrorSQL(e);
43         } catch (Exception e) {
44             System.err.println(e);
45         }
46     }
47 }

```

Vemos que efectivamente se ha eliminado la tabla **ELIMINAME**

Problems @ Javadoc Declaration Console X Error Log Maven Repositories Coverage

<terminated> JDBC\_drop\_table [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot

```

=====
          TABLAS DE LA BASE DE DATOS employees ANTES
=====
Tipo: TABLE - Catalogo: employees, Nombre: departments
Tipo: TABLE - Catalogo: employees, Nombre: dept_emp
Tipo: TABLE - Catalogo: employees, Nombre: dept_manager
Tipo: TABLE - Catalogo: employees, Nombre: eliminame
Tipo: TABLE - Catalogo: employees, Nombre: employees
Tipo: TABLE - Catalogo: employees, Nombre: salaries
Tipo: TABLE - Catalogo: employees, Nombre: titles
=====
Sentencia:drop table ELIMINAME
Se ha eliminado OK
=====
          TABLAS DE LA BASE DE DATOS employees DESPUES
=====
Tipo: TABLE - Catalogo: employees, Nombre: departments
Tipo: TABLE - Catalogo: employees, Nombre: dept_emp
Tipo: TABLE - Catalogo: employees, Nombre: dept_manager
Tipo: TABLE - Catalogo: employees, Nombre: employees
Tipo: TABLE - Catalogo: employees, Nombre: salaries
Tipo: TABLE - Catalogo: employees, Nombre: titles
=====

```

## Ejecución de sentencias DML

- Necesitamos el método **createStatment()** de **Connection** para crear un objeto **Statement**
- Necesitamos el método **execute()** ("sentencia DML") de **Statement** para ejecutar la **sentencia DML**
  - arroja la excepción **SQLException**

**Ejemplo 1:** en una misma clase vamos a ejecutar en la BDD employees

1. Una sentencia DDL para crear una tabla si no existe
2. Una sentencia DDL para truncar la tabla para no violar la clave primaria en las inserciones que se hacen a continuación si ya existieran
3. Una sentencia DML que inserta varias filas en la tabla
4. Una consulta para comprobar que las inserciones se realizaron correctamente

```
1 package JDBC_DML;
2
3 import java.sql.Connection;
4 import java.sql.ResultSet;
5 import java.sql.Statement;
6 import java.sql.SQLException;
7 import jdbc_connection.Conector_JDBC;
8
9 public class JDBC_delete {
10
11     public static void main(String[] args) {
12         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
13             //Hacemos autocommit a ON en nuestra conexión
14             con.setAutoCommit(true);
15             Statement stmt = con.createStatement();
16             String sentenciaDDL = null;
17             sentenciaDDL = "create table if not exists VISITANTE_TEMPORAL (      \n";
18             sentenciaDDL += "    DNI          char(9)      not null,          \n";
19             sentenciaDDL += "    NOMBRE     varchar(20) not null,          \n";
20             sentenciaDDL += "    AP1        varchar(20) not null,          \n";
21             sentenciaDDL += "    AP2        varchar(20) null,             \n";
22             sentenciaDDL += "    DIRECCION  varchar(30),                   \n";
23             sentenciaDDL += "    TFNO      NUMERIC(9),                     \n";
24             sentenciaDDL += "    PRIMARY KEY (DNI)                        )      ";
25             System.out.println ("-----");
26             System.out.printf ("Sentencia DDL: \n %s \n", sentenciaDDL);
27             stmt.execute(sentenciaDDL);
28             System.out.println ("-----");
29             System.out.println ("DDL OK");
30             System.out.println ("-----");
31
32             sentenciaDDL = "truncate table VISITANTE_TEMPORAL";
33             stmt.execute(sentenciaDDL);
34             System.out.printf ("Sentencia DDL: \n %s \n", sentenciaDDL);
35             System.out.println ("-----");
36             System.out.println ("DDL OK");
37         }
```

```

38     String sentenciaDML = null;
39     sentenciaDML = "insert into VISITANTE_TEMPORAL values          \n";
40     sentenciaDML += " ('12345678A', 'Mario', 'Rodrigues', null, null, 917458722), \n";
41     sentenciaDML += " ('23456789B', 'Laura', 'Perez', 'Jimenez', null, null), \n";
42     sentenciaDML += " ('34567890C', 'John', 'Smith', null, 'Desconocida', null) ";
43     System.out.println ("-----");
44     System.out.printf ("Sentencia DML: \n %s \n", sentenciaDML);
45     stmt.execute(sentenciaDML);
46     System.out.println ("-----");
47     System.out.println ("DML OK");
48     System.out.println ("-----");
49
50     String consultal = "select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO "; ;
51     String consulta2 = "from VISITANTE_TEMPORAL ";
52     String consulta3 = "order by DNI";
53     System.out.printf ("Consulta: \n %s \n %s \n %s \n", consultal, consulta2, consulta3);
54     System.out.println ("-----");
55     ResultSet rs = stmt.executeQuery(consultal+consulta2+consulta3);
56     while (rs.next()) {
57         System.out.print(rs.getString(1) + " | ");
58         System.out.print(rs.getString(2) + " | ");
59         System.out.print(rs.getString(3) + " | ");
60         System.out.print(rs.getString(4) + " | ");
61         System.out.print(rs.getString(5) + " | ");
62         System.out.println(rs.getString(6));
63     }
64     System.out.println ("-----");
65     // Cerramos el Statement --> cierra automaticamente el ResultSet
66     stmt.close();
67
68
69     } catch (SQLException e) {
70         Conector_JDBC.muestraErrorSQL(e);
71     } catch (Exception e) {
72         System.err.println(e);
73     }
74
75 }
76
77 }

```

Y obtenemos el resultado esperado:

```

-----
Sentencia DDL:
create table if not exists VISITANTE_TEMPORAL (
  DNI          char(9)      not null,
  NOMBRE       varchar(20)  not null,
  AP1          varchar(20)  not null,
  AP2          varchar(20)  null,
  DIRECCION    varchar(30),
  TFNO         NUMERIC(9),
  PRIMARY KEY (DNI)
)
-----
DDL OK
-----
Sentencia DDL:
truncate table VISITANTE_TEMPORAL
-----
DDL OK
-----
Sentencia DML:
insert into VISITANTE_TEMPORAL values
('12345678A', 'Mario', 'Rodrigues', null, null, 917458722),
('23456789B', 'Laura', 'Perez', 'Jimenez', null, null),
('34567890C', 'John', 'Smith', null, 'Desconocida', null)
-----
DML OK
-----
Consulta:
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO
from VISITANTE_TEMPORAL
order by DNI
-----
12345678A | Mario | Rodrigues | null | null | 917458722
23456789B | Laura | Perez | Jimenez | null | null
34567890C | John | Smith | null | Desconocida | null
-----

```

**Ejemplo 2:** creamos una copia modificada de la clase del **Ejemplo 1** anterior para probar la eliminación de filas con la sentencia **DELETE**

```
1 package JDBC_DML;
2
3 import java.sql.Connection;
4 import java.sql.ResultSet;
5 import java.sql.Statement;
6 import java.sql.SQLException;
7 import jdbc_connection.Conector_JDBC;
8
9 public class JDBC_delete {
10 public static void main(String[] args) {
11     try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
12         //Hacemos autocommit a ON en nuestra conexión
13         con.setAutoCommit(true);
14         Statement stmt = con.createStatement();
15         String sentenciaDDL = null;
16         sentenciaDDL = "create table if not exists VISITANTE_TEMPORAL (      \n";
17         sentenciaDDL += "    DNI          char(9)      not null,      \n";
18         sentenciaDDL += "    NOMBRE      varchar(20) not null,      \n";
19         sentenciaDDL += "    AP1         varchar(20) not null,      \n";
20         sentenciaDDL += "    AP2         varchar(20) null,         \n";
21         sentenciaDDL += "    DIRECCION   varchar(30),              \n";
22         sentenciaDDL += "    TFNO       numeric(9),               \n";
23         sentenciaDDL += "    PRIMARY KEY (DNI)                    )      ";
24         System.out.println ("-----");
25         System.out.printf ("Sentencia DDL: \n  %s \n", sentenciaDDL);
26         stmt.execute(sentenciaDDL);
27         System.out.println ("-----");
28         System.out.println ("CREATE TABLE OK");
29         System.out.println ("-----");
30
31         sentenciaDDL = "truncate table VISITANTE_TEMPORAL";
32         stmt.execute(sentenciaDDL);
33         System.out.printf ("Sentencia DDL: \n  %s \n", sentenciaDDL);
34         System.out.println ("-----");
35         System.out.println ("TRUNCATE OK");
36
37         String sentenciaINSERT = null;
38         sentenciaINSERT = "insert into VISITANTE_TEMPORAL values      \n";
39         sentenciaINSERT += "    ('12345678A', 'Mario', 'Rodrigues', null, null, 917458722), \n";
40         sentenciaINSERT += "    ('23456789B', 'Laura', 'Perez', 'Jimenez', null, null),    \n";
41         sentenciaINSERT += "    ('34567890C', 'John', 'Smith', null, 'Desconocida', null)   ";
42         System.out.println ("-----");
43         System.out.printf ("Sentencia DML: \n  %s \n", sentenciaINSERT);
44         stmt.execute(sentenciaINSERT);
45         System.out.println ("-----");
46         System.out.println ("INSERT OK");
47         System.out.println ("-----");
48     }
```

```

49 String consultal = "select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO "; ;
50 String consulta2 = "from VISITANTE_TEMPORAL ";
51 String consulta3 = "order by DNI";
52 System.out.printf ("Consulta: \n %s \n %s \n %s \n", consultal, consulta2, consulta3);
53 System.out.println ("-----ANTES DE LA SENTENCIA DELETE-----");
54 ResultSet rs = stmt.executeQuery(consultal+consulta2+consulta3);
55 while (rs.next()) {
56     System.out.print(rs.getString(1) + " | ");
57     System.out.print(rs.getString(2) + " | ");
58     System.out.print(rs.getString(3) + " | ");
59     System.out.print(rs.getString(4) + " | ");
60     System.out.print(rs.getString(5) + " | ");
61     System.out.println(rs.getString(6));
62 }
63 System.out.println ("-----");
64 rs.close();
65
66 String sentenciaDELETE = null;
67 sentenciaDELETE = "delete FROM VISITANTE_TEMPORAL\n ";
68 sentenciaDELETE += " where DNI in ('12345678A', '34567890C')";
69 System.out.println ("-----");
70 System.out.printf ("Sentencia DML: \n %s \n", sentenciaDELETE);
71 stmt.execute(sentenciaDELETE);
72 System.out.println ("-----");
73 System.out.println ("DELETE OK");
74 System.out.println ("-----");
75
76 System.out.printf ("Consulta: \n %s \n %s \n %s \n", consultal, consulta2, consulta3);
77 System.out.println ("-----DESPUES DE LA SENTENCIA DELETE-----");
78 rs = stmt.executeQuery(consultal+consulta2+consulta3);
79 while (rs.next()) {
80     System.out.print(rs.getString(1) + " | ");
81     System.out.print(rs.getString(2) + " | ");
82     System.out.print(rs.getString(3) + " | ");
83     System.out.print(rs.getString(4) + " | ");
84     System.out.print(rs.getString(5) + " | ");
85     System.out.println(rs.getString(6));
86 }
87 System.out.println ("-----");
88
89 // Cerramos el Statement --> cierra automaticamente el ResultSet
90 stmt.close();
91 } catch (SQLException e) {
92     Conector_JDBC.muestraErrorSQL(e);
93 } catch (Exception e) {
94     System.err.println(e);
95 }
96 }
97 }

```



Y obtenemos el resultado esperado:

Sentencia DDL:

```
create table if not exists VISITANTE_TEMPORAL (  
  DNI          char(9)      not null,  
  NOMBRE       varchar(20)  not null,  
  AP1          varchar(20)  not null,  
  AP2          varchar(20)  null,  
  DIRECCION    varchar(30),  
  TFNO         NUMERIC(9),  
  PRIMARY KEY (DNI)          )
```

CREATE TABLE OK

Sentencia DDL:

```
truncate table VISITANTE_TEMPORAL
```

TRUNCATE OK

Sentencia DML:

```
insert into VISITANTE_TEMPORAL values  
( '12345678A', 'Mario', 'Rodrigues', null, null, 917458722),  
( '23456789B', 'Laura', 'Perez', 'Jimenez', null, null),  
( '34567890C', 'John', 'Smith', null, 'Desconocida', null)
```

INSERT OK

Consulta:

```
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO  
  from VISITANTE_TEMPORAL  
 order by DNI
```

```
-----ANTES DE LA SENTENCIA DELETE-----  
12345678A | Mario | Rodrigues | null | null | 917458722  
23456789B | Laura | Perez | Jimenez | null | null  
34567890C | John | Smith | null | Desconocida | null
```

Sentencia DML:

```
delete FROM VISITANTE_TEMPORAL  
  where DNI in ('12345678A', '34567890C')
```

DELETE OK

Consulta:

```
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO  
  from VISITANTE_TEMPORAL  
 order by DNI
```

```
-----DESPUES DE LA SENTENCIA DELETE-----  
23456789B | Laura | Perez | Jimenez | null | null
```

**Ejemplo 3:** creamos una copia modificada de la clase del **Ejemplo 1** anterior para probar la eliminación de filas con la sentencia **UPDATE**

```
1 package JDBC_DML;
2
3 import java.sql.Connection;
4 import java.sql.ResultSet;
5 import java.sql.Statement;
6 import java.sql.SQLException;
7 import jdbc_connection.Conector_JDBC;
8
9 public class JDBC_update {
10     public static void main(String[] args) {
11         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
12             //Hacemos autocommit a ON en nuestra conexión
13             con.setAutoCommit(true);
14             Statement stmt = con.createStatement();
15             String sentenciaDDL = null;
16             sentenciaDDL = "create table if not exists VISITANTE_TEMPORAL (      \n";
17             sentenciaDDL += "    DNI          char(9)      not null,          \n";
18             sentenciaDDL += "    NOMBRE     varchar(20) not null,    \n";
19             sentenciaDDL += "    AP1        varchar(20) not null,    \n";
20             sentenciaDDL += "    AP2        varchar(20) null,        \n";
21             sentenciaDDL += "    DIRECCION  varchar(30),             \n";
22             sentenciaDDL += "    TFNO       NUMERIC(9),              \n";
23             sentenciaDDL += "    PRIMARY KEY (DNI)                   )      ";
24             System.out.println ("-----");
25             System.out.printf ("Sentencia DDL: \n  %s \n", sentenciaDDL);
26             stmt.execute(sentenciaDDL);
27             System.out.println ("-----");
28             System.out.println ("CREATE TABLE OK");
29             System.out.println ("-----");
30
31             sentenciaDDL = "truncate table VISITANTE_TEMPORAL";
32             stmt.execute(sentenciaDDL);
33             System.out.printf ("Sentencia DDL: \n  %s \n", sentenciaDDL);
34             System.out.println ("-----");
35             System.out.println ("TRUNCATE OK");
36
37             String sentenciaINSERT = null;
38             sentenciaINSERT = "insert into VISITANTE_TEMPORAL values          \n";
39             sentenciaINSERT += "    ('12345678A', 'Mario', 'Rodriguez', null, null, 917458722), \n";
40             sentenciaINSERT += "    ('23456789B', 'Laura', 'Perez', 'Jimenez', null, null),    \n";
41             sentenciaINSERT += "    ('34567890C', 'John', 'Smith', null, 'Desconocida', null)   ";
42             System.out.println ("-----");
43             System.out.printf ("Sentencia DML: \n  %s \n", sentenciaINSERT);
44             stmt.execute(sentenciaINSERT);
45             System.out.println ("-----");
46             System.out.println ("INSERT OK");
47             System.out.println ("-----");
48
49             String consultal = "select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO ";
```

```

50 String consulta2 = "from VISITANTE_TEMPORAL ";
51 String consulta3 = "order by DNI";
52 System.out.printf ("Consulta: \n %s \n %s \n %s \n", consultal, consulta2, consulta3);
53 System.out.println ("-----ANTES DE LA SENTENCIA UPDATE-----");
54 ResultSet rs = stmt.executeQuery(consultal+consulta2+consulta3);
55 while (rs.next()) {
56     System.out.print(rs.getString(1) + " | ");
57     System.out.print(rs.getString(2) + " | ");
58     System.out.print(rs.getString(3) + " | ");
59     System.out.print(rs.getString(4) + " | ");
60     System.out.print(rs.getString(5) + " | ");
61     System.out.println(rs.getString(6));
62 }
63 System.out.println ("-----");
64 rs.close();
65
66 String sentenciaUPDATE = null;
67 sentenciaUPDATE = "update VISITANTE_TEMPORAL\n ";
68 sentenciaUPDATE += " set DIRECCION = 'Calle General Ricardos, 15'\n ";
69 sentenciaUPDATE += " where DNI in ('12345678A', '23456789B')";
70 System.out.println ("-----");
71 System.out.printf ("Sentencia DML: \n %s \n", sentenciaUPDATE);
72 stmt.execute(sentenciaUPDATE);
73 System.out.println ("-----");
74 System.out.println ("UPDATE OK");
75 System.out.println ("-----");
76
77 System.out.printf ("Consulta: \n %s \n %s \n %s \n", consultal, consulta2, consulta3);
78 System.out.println ("-----DESPUES DE LA SENTENCIA UPDATE-----");
79 rs = stmt.executeQuery(consultal+consulta2+consulta3);
80 while (rs.next()) {
81     System.out.print(rs.getString(1) + " | ");
82     System.out.print(rs.getString(2) + " | ");
83     System.out.print(rs.getString(3) + " | ");
84     System.out.print(rs.getString(4) + " | ");
85     System.out.print(rs.getString(5) + " | ");
86     System.out.println(rs.getString(6));
87 }
88 System.out.println ("-----");
89
90 // Cerramos el Statement --> cierra automaticamente el ResultSet
91 stmt.close();
92 } catch (SQLException e) {
93     Conector_JDBC.muestraErrorSQL(e);
94 } catch (Exception e) {
95     System.err.println(e);
96 }
97 }
98 }

```

Y obtenemos el resultado esperado:

```
<terminated> JDBC_update [Java Application] C:\Users\Fernando\.p2\pool\plugins\org.eclipse.justj.openjdk.
```

```
DNI          char(9)      not null,  
NOMBRE       varchar(20)  not null,  
AP1          varchar(20)  not null,  
AP2          varchar(20)  null,  
DIRECCION    varchar(30),  
TFNO         numeric(9),  
PRIMARY KEY (DNI)                )
```

```
-----  
CREATE TABLE OK  
-----
```

Sentencia DDL:

```
truncate table VISITANTE_TEMPORAL
```

```
-----  
TRUNCATE OK  
-----
```

Sentencia DML:

```
insert into VISITANTE_TEMPORAL values  
( '12345678A', 'Mario', 'Rodrigues', null, null, 917458722),  
( '23456789B', 'Laura', 'Perez', 'Jimenez', null, null),  
( '34567890C', 'John', 'Smith', null, 'Desconocida', null)
```

```
-----  
INSERT OK  
-----
```

Consulta:

```
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO  
from VISITANTE_TEMPORAL  
order by DNI
```

```
-----ANTES DE LA SENTENCIA UPDATE-----  
12345678A | Mario | Rodrigues | null | null | 917458722  
23456789B | Laura | Perez | Jimenez | null | null  
34567890C | John | Smith | null | Desconocida | null  
-----
```

Sentencia DML:

```
update VISITANTE_TEMPORAL  
set DIRECCION = 'Calle General Ricardos, 15'  
where DNI in ('12345678A', '23456789B')
```

```
-----  
UPDATE OK  
-----
```

Consulta:

```
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO  
from VISITANTE_TEMPORAL  
order by DNI
```

```
-----DESPUES DE LA SENTENCIA UPDATE-----  
12345678A | Mario | Rodrigues | null | Calle General Ricardos, 15 | 917458722  
23456789B | Laura | Perez | Jimenez | Calle General Ricardos, 15 | null  
34567890C | John | Smith | null | Desconocida | null  
-----
```

# Ejecución de procedimientos almacenados en la base de datos

## Procedimientos y funciones almacenados

Muchos SGBD relacionales poseen lenguajes de programación que extienden el lenguaje SQL y permiten crear bloques de código que pueden ser invocados por otros bloques o desde una aplicación externa.

Esos bloques que pueden ser invocados (recibiendo y devolviendo parámetros) pueden ser de dos tipos:

- **Procedimientos:**
  - no devuelven ningún valor (aunque pueden tener parámetros de salida o de entrada-salida)
  - no pueden usarse en consultas
- **Funciones:**
  - devuelven siempre algún valor de un tipo conocido
  - sólo pueden tener parámetros de entrada
  - pueden usarse en consultas

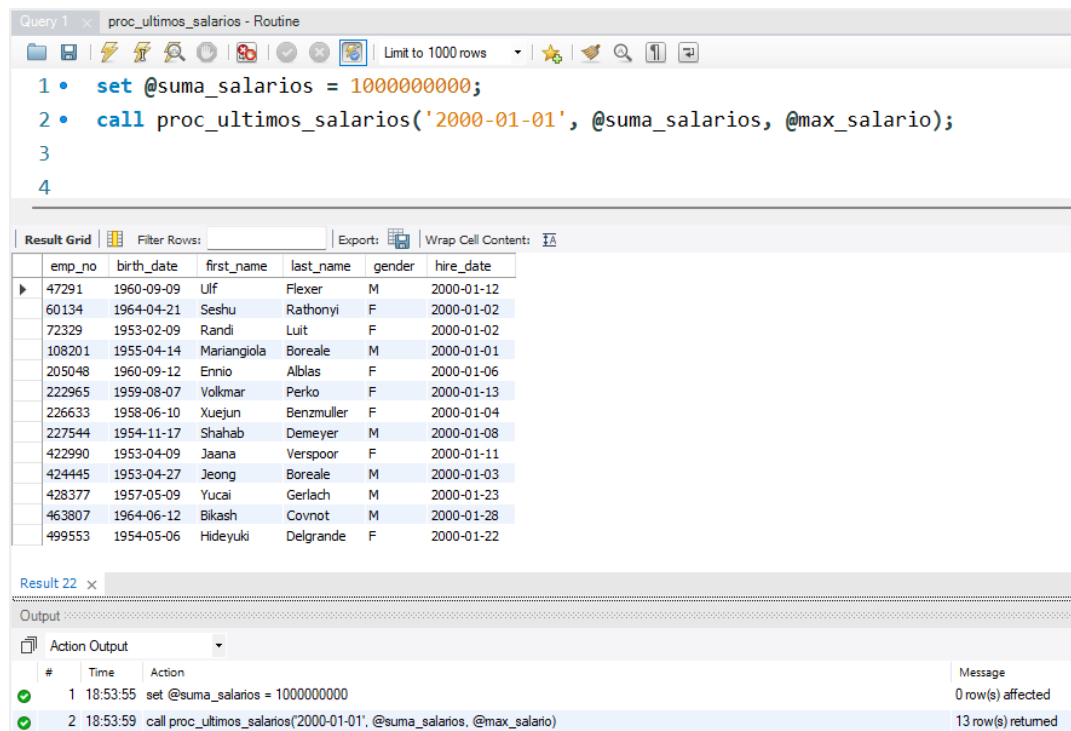
### Ejemplo 1: procedimiento **proc\_ultimos\_salarios** en la BDD **employees** en un servidor MySQL

1. añade al valor parámetro de entrada/salida **p\_suma\_sueldos** la suma de los sueldos posteriores al parámetro de entrada **p\_fecha\_desde**
2. devuelve el salario máximo en el parámetro de salida **p\_sueldo\_maximo**
3. selecciona el número y apellido de todos los empleados posteriores a la fecha indicada en el parámetro de entrada **p\_fecha\_desde**

```
1 • CREATE PROCEDURE proc_ultimos_salarios (in p_fecha_desde date,  
2                                         inout p_suma_sueldos bigint,  
3                                         out p_sueldo_maximo int)  
4 BEGIN  
5     select sum(salary) into p_suma_sueldos  
6         from salaries  
7         where from_date >= p_fecha_desde;  
8     select max(salary) into p_sueldo_maximo  
9         from salaries;  
10    select *  
11        from employees  
12        where hire_date >= p_fecha_desde;  
13 END
```

Para ejecutar el procedimiento es necesario pasarle 3 parámetros, uno de entrada, otro de entrada/salida y otro de salida, por ejemplo:

- la llamada al procedimiento con el comando **CALL** mostrará los empleados contratados a partir del parámetro de entrada



Query 1 x proc\_ultimos\_salarios - Routine

```
1 • set @suma_salarios = 1000000000;  
2 • call proc_ultimos_salarios('2000-01-01', @suma_salarios, @max_salario);  
3  
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

emp_no	birth_date	first_name	last_name	gender	hire_date
47291	1960-09-09	Ulf	Flexer	M	2000-01-12
60134	1964-04-21	Seshu	Rathonyi	F	2000-01-02
72329	1953-02-09	Randi	Luit	F	2000-01-02
108201	1955-04-14	Mariangiola	Boreale	M	2000-01-01
205048	1960-09-12	Ennio	Alblas	F	2000-01-06
222965	1959-08-07	Volkmar	Perko	F	2000-01-13
226633	1958-06-10	Xuejun	Benzmuller	F	2000-01-04
227544	1954-11-17	Shahab	Demeyer	M	2000-01-08
422990	1953-04-09	Jaana	Verspoor	F	2000-01-11
424445	1953-04-27	Jeong	Boreale	M	2000-01-03
428377	1957-05-09	Yucui	Gerlach	M	2000-01-23
463807	1964-06-12	Bikash	Covnot	M	2000-01-28
499553	1954-05-06	Hideyuki	Delgrande	F	2000-01-22

Result 22 x

Output

Action Output

#	Time	Action	Message
1	18:53:55	set @suma_salarios = 1000000000	0 row(s) affected
2	18:53:59	call proc_ultimos_salarios('2000-01-01', @suma_salarios, @max_salario)	13 row(s) returned

- obtenemos el valor del parámetro de salida y vemos que el de entrada/salida se ha modificado

Query 1 x proc\_ultimos\_salarios - Routine

```

1 • set @suma_salarios = 1000000000;
2 • call proc_ultimos_salarios('2000-01-01', @suma_salarios, @max_salario);
3 • select @suma_salarios, @max_salario;
4

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: I A

@suma_salarios	@max_salario
45286752527	158220

## Ejemplo 2: función **apellido\_empleado** en la BDD **employees** en un servidor **MySQL**

- devuelve el apellido del número de empleado que se le pasa como parámetro

```

1 • CREATE FUNCTION apellido_empleado(p_num_empleado int)
2 RETURNS varchar(15)
3 DETERMINISTIC
4 BEGIN
5     declare resultado VARCHAR(15);
6     select last_name
7     into resultado
8     from employees
9     where emp_no = p_num_empleado;
10 RETURN resultado;
11 END

```

Para probar la función basta con ejecutar una consulta:

Query 1 x

```

1 • select apellido_empleado(30005) from dual;
2
3

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: I A

apellido_empleado(30005)
Nourani

## Método `prepareCall()`

El método `prepareCall(String sql)` de **Connection** devuelve un objeto que implemente la interfaz **CallableStatement** la cual deriva de la interfaz **PreparedStatement** la cual a su vez deriva de **Statement**

- Algunos métodos importantes de **CallableStatement** son:

<b>void</b> <code>registerOutParameter(int</code> posicion, <b>int</b> tipoSQL)	Asigna al parámetro de un procedimiento en la posición indicada el tipo de dato SQL que se le indique: tipos codificados en <b>java.sql.Types</b> → arroja <b>SQLException</b>
<b>boolean</b> <code>execute(String</code> llamada)	Ejecuta la llamada al procedimiento o función almacenado → arroja <b>SQLException</b>
<b>ResultSet</b> <code>getResultSet()</code>	Devuelve el resultado de una consulta → arroja <b>SQLException</b>
<b>Date</b> <code>getDate()</code>	Devuelve una fecha del paquete java.SQL → arroja <b>SQLException</b>
<b>int</b> <code>getInt()</code>	Devuelve un valor entero → arroja <b>SQLException</b>
<b>String</b> <code>getString()</code>	Devuelve una cadena → arroja <b>SQLException</b>
<b>void</b> <code>setDate(int</code> posicion, <b>Date</b> fecha)	Le da al parámetro de entrada en la posición indicada el valor de la fecha → arroja <b>SQLException</b>
<b>void</b> <code>setInt (int</code> posicion, <b>int</b> numero)	Le da al parámetro de entrada en la posición indicada el valor del entero → arroja <b>SQLException</b>
<b>void</b> <code>setString(int</code> posicion, <b>String</b> cadena)	Le da al parámetro de entrada en la posición indicada el valor de la cadena → arroja <b>SQLException</b>

**Nota:** las fechas implementadas en la clase **java.sql.Date** son diferentes a las fechas comunes de Java contenidas en la clase **java.util.Date**



- La sintaxis para llamar a un **procedimiento almacenado** es

```
prepareCall("{call nombre_procedimiento(?,...?,?)}")
```

- cada **signo de interrogación** corresponde a un **parámetro**
- inicialización de **parámetros de entrada y entrada-salida**
  - con los métodos **setDate()**, **setInt()**, **setString()**, ...
- designación de los tipos de los **parámetros de entrada-salida y de salida**
  - con el método **registerOutParameter(int posicion, int tipoSQL)**
- recogida de los **parámetros de entrada-salida y de salida**
  - con los métodos **getDate()**, **getInt()**, **getString()**, ...

**Ejemplo:** invocamos al procedimiento **proc\_ultimos\_salarios** con

- Un parámetro de entrada **fecha\_desde**
- Un parámetro de entrada-salida **sueudos\_acumulados**
- Un parámetro de salida **max\_salario**

```

1 package callablestatement;
2
3 import java.sql.Connection;
4 import java.sql.CallableStatement;
5 import jdbc_connection.Conector_JDBC;
6 import java.sql.SQLException;
7 import java.sql.Types;
8 import java.sql.Date;
9 import java.sql.ResultSet;
10
11 public class JDBC_procedimiento_almacenado {
12
13     public static void main(String[] args) {
14         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
15             //Preparamos una sentencia para ejecutar el procedimiento con 3 parámetros
16             CallableStatement cstmt = con.prepareCall("{call proc_ultimos_salarios(?,?,?)");
17             //El 1º parámetro es de ENTRADA --> de tipo java.sql.Date (!= java.util.Date)
18             //Primero convertimos el String a java.sql.Date
19             Date fecha_desde = Date.valueOf("2000-01-01");
20             cstmt.setDate(1, fecha_desde);
21
22             //El 2º parámetro es de ENTRADA/SALIDA: BIGINT en MySQL --> long en Java
23             //Inicializamos el parámetro de E/S --> este valor se modificará tras la ejecución
24             long sueldos_acumulados = 300000000;
25             cstmt.setLong(2, sueldos_acumulados); // este valor se modificará tras la ejecución
26             //registramos el parámetro como de salida y le decimos su tipo java.sql.Types
27             cstmt.registerOutParameter(2, Types.BIGINT);
28             System.out.println ("Parametro E/S antes: " + sueldos_acumulados);
29
30             //El 3º parámetro es de SALIDA: int en MySQL --> int en Java
31             int max_salario;
32             //registramos el parámetro como de salida y le decimos su tipo java.sql.Types
33             cstmt.registerOutParameter(3, Types.INTEGER);
34

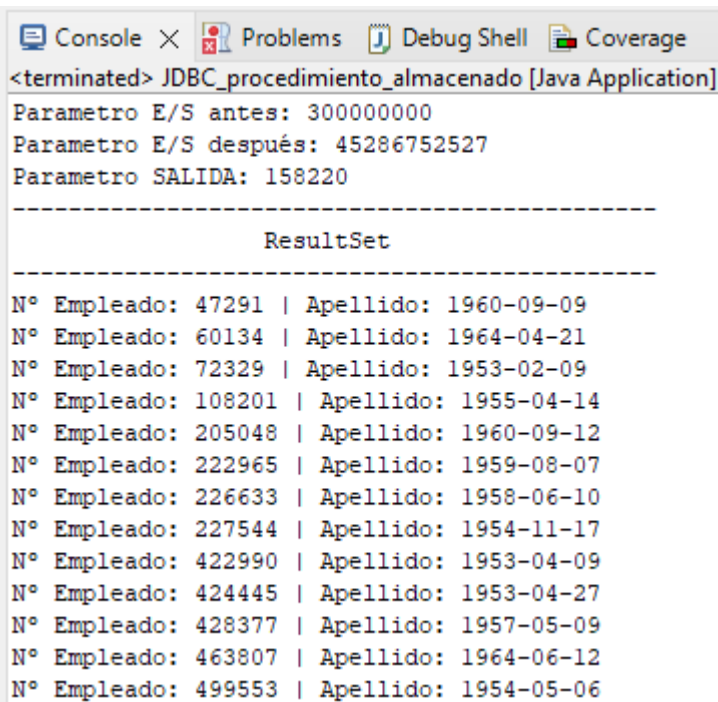
```

```

35         //Ejecutamos el procedimiento
36         pstmt.execute();
37         //Leemos y mostramos el parámetro de E/S
38         sueldos_acumulados = pstmt.getLong(2);
39         System.out.println ("Parametro E/S después: " + sueldos_acumulados);
40         //Leemos y mostramos el parámetro de SALIDA
41         max_salario = pstmt.getInt(3);
42         System.out.println ("Parametro SALIDA: " + max_salario);
43
44         //Leemos el ResultSet y lo mostramos
45         System.out.println ("-----");
46         System.out.println ("          ResultSet          ");
47         System.out.println ("-----");
48         ResultSet rs = pstmt.getResultSet();
49         while (rs.next()) {
50             System.out.print("Nº Empleado: " + rs.getInt(1));
51             System.out.println(" | Apellido: " + rs.getString(2));
52         }
53         // Cerramos el CallableStatement --> cierra automaticamente el ResultSet
54         pstmt.close();
55     } catch (SQLException e) {
56         Conector_JDBC.muestraErrorSQL(e);
57     } catch (Exception e) {
58         System.err.println(e);
59     }
60 }

```

Que nos devolverá



The screenshot shows a Java IDE console window with the following tabs: Console, Problems, Debug Shell, and Coverage. The console output is as follows:

```

<terminated> JDBC_procedimiento_almacenado [Java Application]
Parametro E/S antes: 3000000000
Parametro E/S después: 45286752527
Parametro SALIDA: 158220
-----
          ResultSet
-----
Nº Empleado: 47291 | Apellido: 1960-09-09
Nº Empleado: 60134 | Apellido: 1964-04-21
Nº Empleado: 72329 | Apellido: 1953-02-09
Nº Empleado: 108201 | Apellido: 1955-04-14
Nº Empleado: 205048 | Apellido: 1960-09-12
Nº Empleado: 222965 | Apellido: 1959-08-07
Nº Empleado: 226633 | Apellido: 1958-06-10
Nº Empleado: 227544 | Apellido: 1954-11-17
Nº Empleado: 422990 | Apellido: 1953-04-09
Nº Empleado: 424445 | Apellido: 1953-04-27
Nº Empleado: 428377 | Apellido: 1957-05-09
Nº Empleado: 463807 | Apellido: 1964-06-12
Nº Empleado: 499553 | Apellido: 1954-05-06

```

- La sintaxis para llamar a una **función almacenada** es

**prepareCall("{? = call nombre\_funcion(?,...,?)}")**

- el **primer signo de interrogación** corresponde al **resultado de la función**
- los demás signos de interrogación corresponden a los **parámetros de entrada**
- inicialización de **parámetros**
  - con los métodos **setDate()**, **setInt()**, **setString()**, ...
- designación del **tipo del resultado**
  - con el método **registerOutParameter(int posicion, int tipoSQL)**
- recogida del **resultado**
  - con los métodos **getDate()**, **getInt()**, **getString()**, ...

**Ejemplo:** invocamos a la función **apellido\_Empleado**

- Un **parámetro** (de entrada) **numEmpleado**
- Un **resultado** de tipo **VARCHAR**

```
JDBC_funcion_almacenada.java X
1 package callablestatement;
2
3 import java.sql.Connection;
4
5 public class JDBC_funcion_almacenada {
6
7     public static void main(String[] args) {
8         try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
9             CallableStatement cstmt = con.prepareCall("{? = call apellido_Empleado(?)}")
10
11             //Designamos el tipo del resultado: VARCHAR en SQL --> String en Java
12             cstmt.registerOutParameter(1, Types.VARCHAR);
13
14             //Inicializamos el parámetro (de entrada)
15             int numEmpleado = 20001;
16             cstmt.setInt(2, numEmpleado);
17
18             //Ejecutamos el CallableStatement
19             cstmt.execute();
20             String apellido = cstmt.getString(1);
21             System.out.printf ("Apellido del empleado n° %d: %s\n", numEmpleado, apellido.toString());
22
23             // Cerramos el CallableStatement
24             cstmt.close();
25         } catch (SQLException e) {
26             Conector_JDBC.muestraErrorSQL(e);
27         } catch (Exception e) {
28             System.err.println(e);
29         }
30     }
31 }
32
33
34
35
```

Console X Problems Debug Shell Coverage

<terminated> JDBC\_funcion\_almacenada [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_18.0.

Apellido del empleado n° 20001: Eppinger

# Gestión de transacciones

Recordamos que el lenguaje SQL permite gestionar transacciones mediante dos sentencias:

1. **ROLLBACK** → **anula** todas las operaciones DML (INSERT, DELETE o UPDATE) desde la sentencia **COMMIT explícito** anterior, desde la anterior sentencia **COMMIT implícito** (sentencia DDL o DCL) o desde la sentencia ROLLBACK anterior
2. **COMMIT** → **valida** todas las operaciones DML (INSERT, DELETE o UPDATE) desde la sentencia **COMMIT explícito** anterior, desde la anterior sentencia **COMMIT implícito** (sentencia DDL o DCL) o desde la sentencia ROLLBACK anterior

## Ejemplo:

1. fijamos **AUTOCOMMIT** a **OFF**
2. insertamos tres filas y vemos que desaparecen después de hacer **rollback**
3. insertamos de nuevo las tres filas y vemos que después de hacer **commit** se mantienen

```
1 package JDBC_transacciones;
2
3 import java.sql.Connection;
4 import java.sql.ResultSet;
5 import java.sql.Statement;
6 import java.sql.SQLException;
7 import jdbc_connection.Conector_JDBC;
8
9 public class JDBC_transacciones {
10     public static void sentenciasDDL(Connection con) throws SQLException {
11         Statement stmt = con.createStatement();
12         String sentenciaDDL = null;
13         sentenciaDDL = "create table if not exists VISITANTE_TEMPORAL (      \n";
14         sentenciaDDL += "     DNI          char(9)      not null,          \n";
15         sentenciaDDL += "     NOMBRE      varchar(20) not null,          \n";
16         sentenciaDDL += "     AP1        varchar(20) not null,          \n";
17         sentenciaDDL += "     AP2        varchar(20) null,             \n";
18         sentenciaDDL += "     DIRECCION  varchar(30),                  \n";
19         sentenciaDDL += "     TFNO       NUMERIC(9),                   \n";
20         sentenciaDDL += "     PRIMARY KEY (DNI)                       )      \n";
21         System.out.println ("-----");
22         System.out.printf ("Sentencia DDL: \n %s \n", sentenciaDDL);
23         stmt.execute(sentenciaDDL);
24         System.out.println ("-----");
25         System.out.println ("CREATE TABLE OK");
26         System.out.println ("-----");
27         sentenciaDDL = "truncate table VISITANTE_TEMPORAL";
28         stmt.execute(sentenciaDDL);
29         System.out.printf ("Sentencia DDL: \n %s \n", sentenciaDDL);
30         System.out.println ("-----");
31         System.out.println ("TRUNCATE OK");
32         stmt.close();
33     }
34     public static void sentenciasINSERT(Connection con) throws SQLException {
35         Statement stmt = con.createStatement();
36         String sentenciaINSERT = null;
37         sentenciaINSERT = "insert into VISITANTE_TEMPORAL values          \n";
38         sentenciaINSERT += "     ('12345678A', 'Mario', 'Rodriguez', null, null, 917458722), \n";
39         sentenciaINSERT += "     ('23456789B', 'Laura', 'Perez', 'Jimenez', null, null),    \n";
40         sentenciaINSERT += "     ('34567890C', 'John', 'Smith', null, 'Desconocida', null)   \n";
41         System.out.println ("-----");
42         System.out.printf ("Sentencia DML: \n %s \n", sentenciaINSERT);
43         stmt.execute(sentenciaINSERT);
44         System.out.println ("-----");
45         System.out.println ("INSERT OK");
46         System.out.println ("-----");
47         stmt.close();
48     }
49 }
```

```

49 public static void consultas(Connection con, String antesDespues, String commitRollback) throws SQLException {
50     Statement stmt = con.createStatement();
51     String consulta1 = "select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO ";
52     String consulta2 = "from VISITANTE_TEMPORAL ";
53     String consulta3 = "order by DNI";
54     System.out.printf ("Consulta: \n %s \n %s \n %s \n", consulta1, consulta2, consulta3);
55     System.out.println ("-----" + antesDespues + " DE LA SENTENCIA " + commitRollback + "-----");
56     ResultSet rs = stmt.executeQuery(consulta1+consulta2+consulta3);
57     while (rs.next()) {
58         System.out.print(rs.getString(1) + " | ");
59         System.out.print(rs.getString(2) + " | ");
60         System.out.print(rs.getString(3) + " | ");
61         System.out.print(rs.getString(4) + " | ");
62         System.out.print(rs.getString(5) + " | ");
63         System.out.println(rs.getString(6));
64     }
65     System.out.println ("-----");
66     stmt.close(); // Cerramos el Statement --> cierra automaticamente el ResultSet
67 }
68
69 public static void main(String[] args) {
70     try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaalegre")) {
71         //Ponemos el modo AUTOCOMMIT A OFF
72         con.setAutoCommit(false);
73         sentenciasDDL(con); //creamos la tabla
74         sentenciasINSERT(con); //hacemos los insert
75         consultas(con, "--ANTES", "ROLLBACK");
76         Statement stmt = con.createStatement();
77         //hacemos ROLLBACK
78         stmt.execute("ROLLBACK");
79         consultas(con, "DESPUES", "ROLLBACK");
80         sentenciasINSERT(con);
81         consultas(con, "--ANTES", "COMMIT--");
82         //hacemos COMMIT
83         stmt.execute("COMMIT");
84         consultas(con, "DESPUES", "COMMIT--");
85         stmt.close();
86     } catch (SQLException e) {
87         Conector_JDBC.muestraErrorSQL(e);
88     } catch (Exception e) {
89         System.err.println(e);
90     }
91 }
92 }

```

comprobamos que se se obtiene el resultado esperado:

```
Console × Problems Debug Shell Coverage
<terminated> JDBC_transacciones [Java Application] C:\Users\Fernando\p2\pool\plugins\org.eclipse.just

-----
Sentencia DDL:
create table if not exists VISITANTE_TEMPORAL (
  DNI          char(9)      not null,
  NOMBRE       varchar(20) not null,
  AP1          varchar(20) not null,
  AP2          varchar(20) null,
  DIRECCION    varchar(30),
  TFNO         NUMERIC(9),
  PRIMARY KEY (DNI)
)

-----
CREATE TABLE OK

-----
Sentencia DDL:
truncate table VISITANTE_TEMPORAL

-----
TRUNCATE OK

-----
Sentencia DML:
insert into VISITANTE_TEMPORAL values
('12345678A', 'Mario', 'Rodrigues', null, null, 917458722),
('23456789B', 'Laura', 'Perez', 'Jimenez', null, null),
('34567890C', 'John', 'Smith', null, 'Desconocida', null)

-----
INSERT OK

-----
Consulta:
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO
from VISITANTE_TEMPORAL
order by DNI

-----
--ANTES DE LA SENTENCIA ROLLBACK--
12345678A | Mario | Rodrigues | null | null | 917458722
23456789B | Laura | Perez | Jimenez | null | null
34567890C | John | Smith | null | Desconocida | null
-----
```

```
Consulta:
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO
from VISITANTE_TEMPORAL
order by DNI

-----
--DESPUES DE LA SENTENCIA ROLLBACK--
-----

Sentencia DML:
insert into VISITANTE_TEMPORAL values
('12345678A', 'Mario', 'Rodrigues', null, null, 917458722),
('23456789B', 'Laura', 'Perez', 'Jimenez', null, null),
('34567890C', 'John', 'Smith', null, 'Desconocida', null)

-----
INSERT OK

-----
Consulta:
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO
from VISITANTE_TEMPORAL
order by DNI

-----
--ANTES DE LA SENTENCIA COMMIT--
12345678A | Mario | Rodrigues | null | null | 917458722
23456789B | Laura | Perez | Jimenez | null | null
34567890C | John | Smith | null | Desconocida | null
-----

Consulta:
select DNI, NOMBRE, AP1, AP2, DIRECCION, TFNO
from VISITANTE_TEMPORAL
order by DNI

-----
--DESPUES DE LA SENTENCIA COMMIT--
12345678A | Mario | Rodrigues | null | null | 917458722
23456789B | Laura | Perez | Jimenez | null | null
34567890C | John | Smith | null | Desconocida | null
-----
```

# Para saber más

Hasta aquí has aprendido lo mínimo que debes saber sobre esta unidad. Vamos a aprender ahora aspectos más avanzados.

## Optimización de sentencias DML: sentencias preparadas

En muchas ocasiones una aplicación va a necesitar ejecutar una sentencia SQL que no se conoce completa hasta el momento de tener que ejecutarla y se construye en la aplicación.

Por ejemplo, si necesito encontrar los datos de un cliente cuyo DNI me llega el parámetro `p_dni` construiría la sentencia:

**String** consulta = "SELECT \* FROM CLIENTES WHERE DNI '=' + p\_dni + '";"

Hacer esto repetitivamente tiene dos graves problemas:

1. **Mal rendimiento de la BDD** → cada vez que se ejecute la sentencia anterior con un DNI distinto se tiene que
  - a. enviar la sentencia completa de nuevo
  - b. compilar la sentencia completa de nuevo
2. **Mala seguridad** → un hacker malicioso podría modificar el valor del parámetro `p_dni` (inyección de código SQL)

Para evitar estos problemas JDBC implementa las **sentencias preparadas**:

- cada **sentencia preparada** se envía a la BDD una sola vez
- el SGBD **precompila** la sentencia
- la aplicación solamente envía los valores de los parámetros para cada ejecución

## Método `prepareStatement()`

El método `prepareStatement(String sql)` de **Connection** devuelve un objeto que implemente la interfaz **PreparedStatement** la cual deriva de la interfaz **Statement**

La lista de **parámetros** que va a recibir una sentencia se marca con **signos de interrogación** (un signo para cada parámetro)

**Ejemplo:** la siguiente sentencia INSERT va a recibir tres parámetros

**PreparedStatement** s = con.**prepareStatement**("INSERT INTO TABLA VALUES (?, ?, ?);");

Algunos de métodos más importantes de *PreparedStatement* son:

<b>int executeUpdate()</b>	Ejecuta la sentencia DML (INSERT, DELETE o UPDATE) → arroja <b>SQLException</b> y <b>SQLTimeoutException</b>
<b>void setInt(int</b> posicion, <b>int</b> valor)	Fija el valor del entero en la posición indicada → arroja <b>SQLException</b>
<b>void setString(int</b> posicion, <b>String</b> valor)	Fija el valor de la cadena en la posición indicada → arroja <b>SQLException</b>
<b>void setNull(int</b> posicion, <b>int</b> tipoSQL)	Pone a NULL el campo en la posición indicada (es necesario indicar el tipo de dato SQL: tipos codificados en java.sql.Types) → arroja <b>SQLException</b>

**Ejemplo:** en el método **main()** usamos un *PreparedStatement* para insertar tres filas en una tabla

```

1 package jdbc_prepared_statement;
2
3 import java.sql.Connection;
4 import java.sql.Statement;
5 import java.sql.Types;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.sql.SQLException;
9 import jdbc_connection.Conector_JDBC;
10
11 public class JDBC_prepared_statement {
12
13     public static void CrearTabla(Connection con) throws SQLException {
14         Statement stmt = con.createStatement();
15         String sentencia = "drop table if exists ELIMINAME";
16         System.out.println (sentencia);
17         stmt.execute(sentencia);
18         sentencia = "create table ELIMINAME (          \n";
19         sentencia += " DNI          char(9) not null, \n";
20         sentencia += " APELLIDOS  varchar(32) not null, \n";
21         sentencia += " CP          int,          \n";
22         sentencia += " PRIMARY KEY (DNI)      )";
23         System.out.println (sentencia);
24         stmt.execute(sentencia);
25         stmt.close();
26         System.out.println ("Se ha creado OK");
27     }
28
29     public static void ConsultaTabla(Connection con) throws SQLException {
30         Statement stmt = con.createStatement();
31         ResultSet rs = stmt.executeQuery("select * from ELIMINAME order by 1");
32         while (rs.next()) {
33             System.out.print("DNI: " + rs.getString(1) + " | ");
34             System.out.print("APELLIDOS: " + rs.getString(2) + " | ");
35             String cp = ((Integer) rs.getInt(3)).toString(); //getInt devuelve 0 si el valor es NULL
36             if (cp.equals("0")) cp = "NULL";
37             System.out.println("CP: " + cp);
38         }
39         // Cerramos el Statement --> cierra automaticamente el ResultSet
40         stmt.close();
41     }
42

```



```

43 public static void main(String[] args) {
44     try (Connection con = Conector_JDBC.Conexion("employees", "localhost", "3306", "root", "vistaaalegre")) {
45         CrearTabla(con);
46         //Sentencia preparada
47         PreparedStatement sInsert = con.prepareStatement("INSERT INTO ELIMINAME (DNI, APELLIDOS, CP) VALUES (?, ?, ?)");
48         sInsert.setString(1, "78901234X");
49         sInsert.setString(2, "NADALES");
50         sInsert.setInt(3, 44126);
51         sInsert.executeUpdate();
52         System.out.println("DML ejecutada OK");
53         sInsert.setString(1, "89102345E");
54         sInsert.setString(2, "ROJAS");
55         sInsert.setNull(3, Types.INTEGER);
56         sInsert.executeUpdate();
57         System.out.println("DML ejecutada OK");
58         sInsert.setString(1, "56789012B");
59         sInsert.setString(2, "SAMPER");
60         sInsert.setInt(3, 29730);
61         sInsert.executeUpdate();
62         System.out.println("DML ejecutada OK");
63         // Cerramos el PreparedStatement
64         sInsert.close();
65         ConsultaTabla(con);
66     } catch (SQLException e) {
67         Conector_JDBC.muestraErrorSQL(e);
68     } catch (Exception e) {
69         System.err.println(e);
70     }
71 }
72
73 }

```

Comprobamos que se han insertado las tres filas correctamente:

```

Problems Tasks Console × Properties
<terminated> JDBC_prepared_statement [Java Application] C:\Users\Fernando\g
drop table if exists ELIMINAME
create table ELIMINAME (
  DNI          char(9) not null,
  APELLIDOS    varchar(32) not null,
  CP           int,
  PRIMARY KEY (DNI)
)
Se ha creado OK
DML ejecutada OK
DML ejecutada OK
DML ejecutada OK
DNI: 56789012B | APELLIDOS: SAMPER | CP: 29730
DNI: 78901234X | APELLIDOS: NADALES | CP: 44126
DNI: 89102345E | APELLIDOS: ROJAS | CP: NULL

```

## Acceso a una BDD con el conector ODBC

1. En una MV Ubuntu instalamos los paquetes necesarios para configurar el driver ODBC siguiendo esta guía  
<https://codersathi.com/install-mysql-odbc-driver-in-ubuntu/>
- 2.