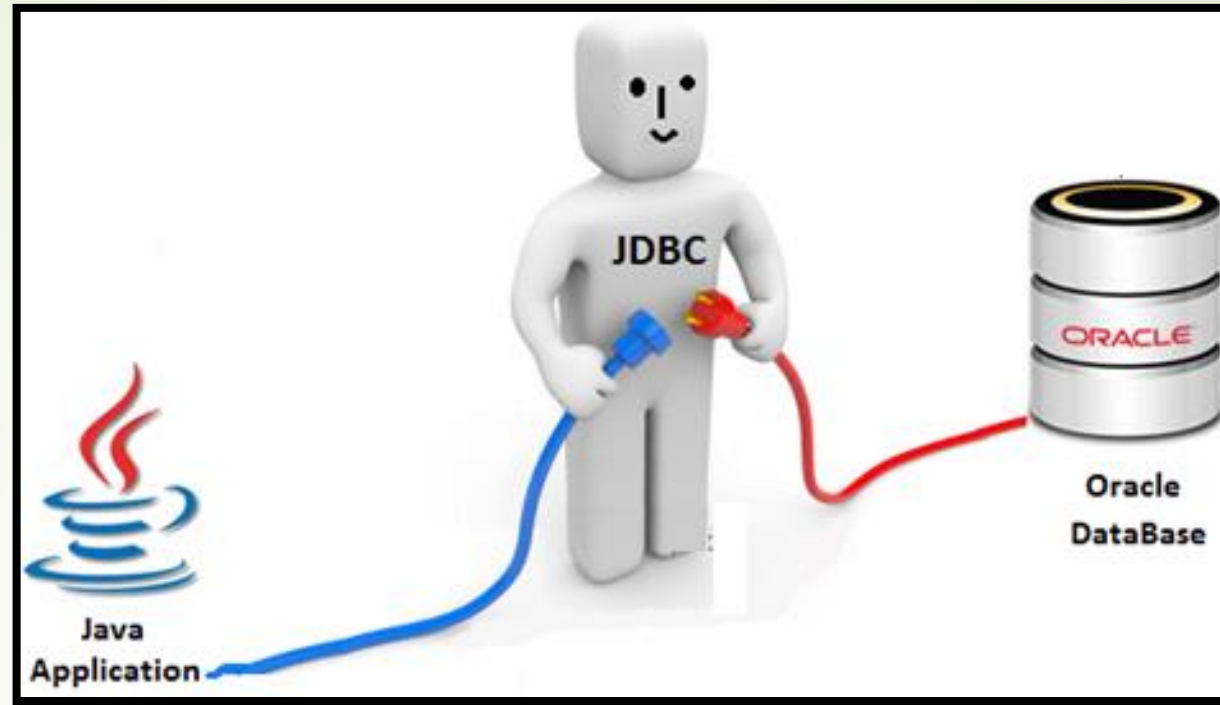




Desarrollo de Aplicaciones Multiplataforma

DOCENTE: Daniel López Lozano



Tema 2.

Desarrollo de Aplicaciones con Bases de Datos Relacionales

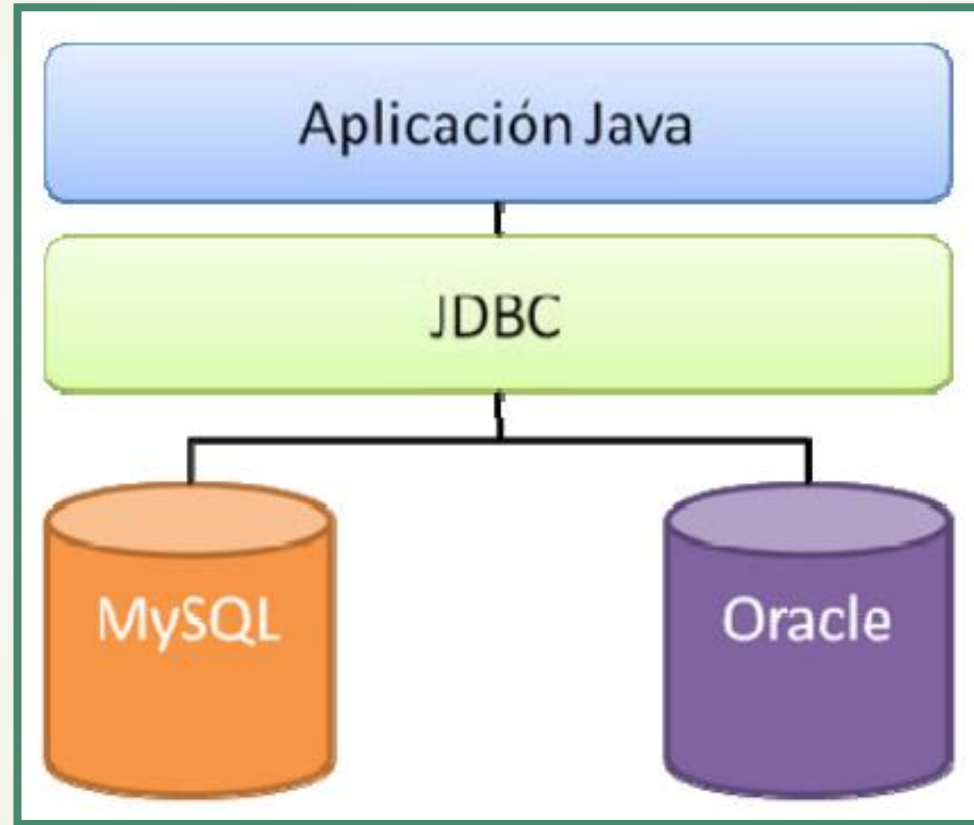
Índice de contenidos

- ❑ **Desfase Objeto-Relacional.**
- ❑ **ODBC y JDBC.**
- ❑ **Ejecución de código SQL desde Java usando MySQL.**

- ❑ El desfase **objeto-relacional** consisten en la **diferencia de aspectos** que existen entre la programación orientada a objetos y las bases de datos relacionales.
- ❑ El modelo relacional usa **relaciones y conjuntos** con una base matemática.
- ❑ La programación orientada a objetos se basa en **clases, objetos y las asociaciones** entre ellos (herencia).

- ❑ La diferencia entre objetos y las tablas con filas (tuplas) implica desarrollar un **software adicional** que compenetre ambos esquemas.
- ❑ Unos de los puntos a tratar de este curso es ver **soluciones** para solventar el problema del desfase objeto-relacional.
- ❑ En este tema veremos como usar **los conectores** que se comunican con el sistema gestor de bases de datos.
- ❑ El conector **envía la consulta SQL** a la base de datos.

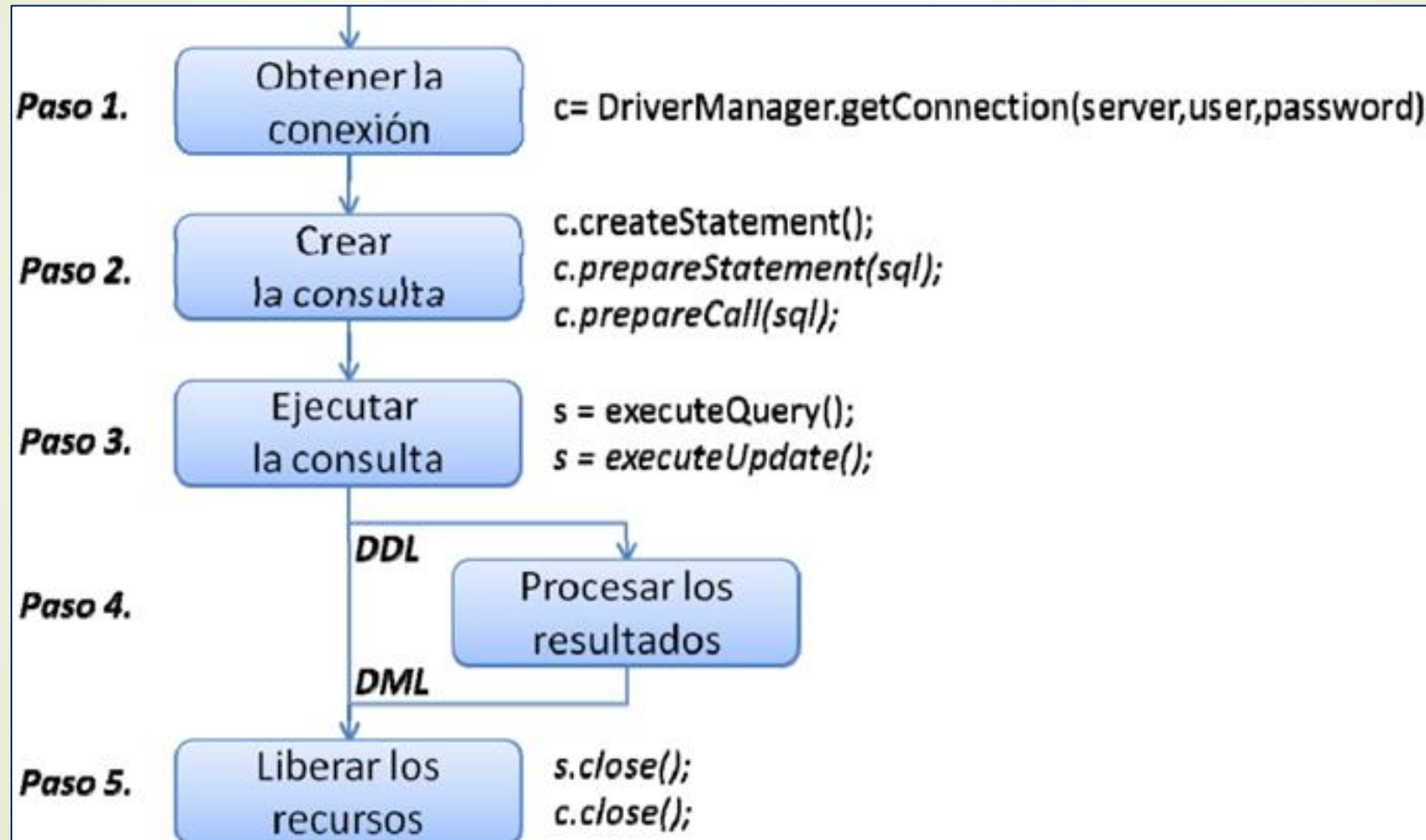
- ❑ Las normas más comunes que implementan dicha interfaz son **ODBC** y **JDBC**.



- ❑ **ODBC (Open Database Connectivity)** define una API que pueden usar las aplicaciones para abrir una conexión con una base de datos.
- ❑ Permitiendo realizar consultas, actualizaciones y obtener los resultados **en el propio lenguaje**.
- ❑ Se puede usar dicha API para conectar con cualquier **servidor de bases de datos compatible con ODBC**.
- ❑ ODBC fue desarrollado por **Microsoft**.

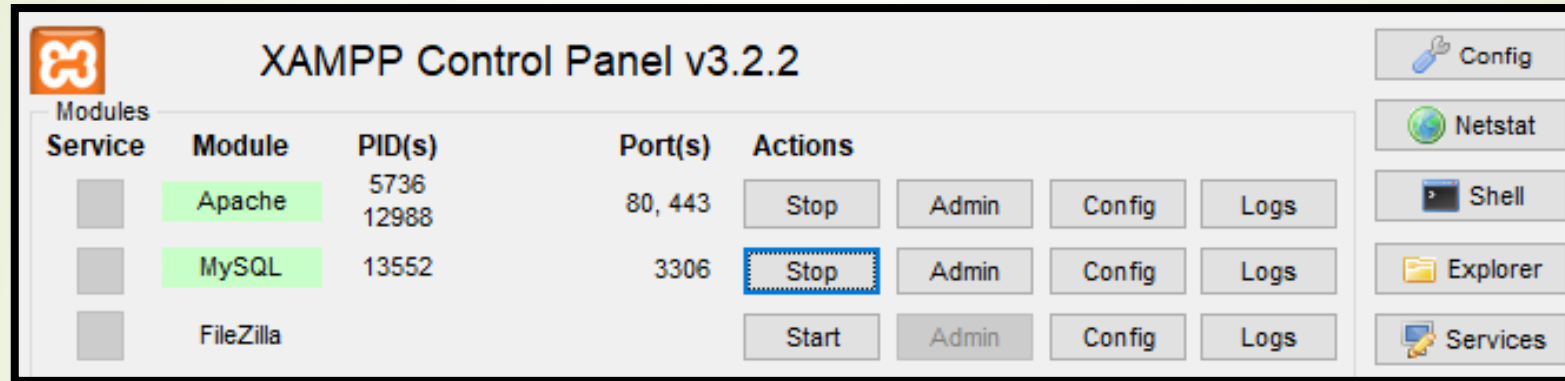
- ❑ **JDBC (Java Database Connectivity)** define una API similar a ODBC pero para el lenguaje Java.
- ❑ JDBC además de proveer un interfaz también define una **arquitectura estándar** para que los fabricantes puedan desarrollar la compatibilidad de conexión con Java.
- ❑ Dentro del modelo existen alternativas para la compatibilidad donde el conector puede estar escrito **totalmente en Java**, usar un **conector ODBC** como puente, etc.

- ❑ Para acceder a bases de datos usando JDBC se debe seguir el siguiente esquema y uso de objetos.

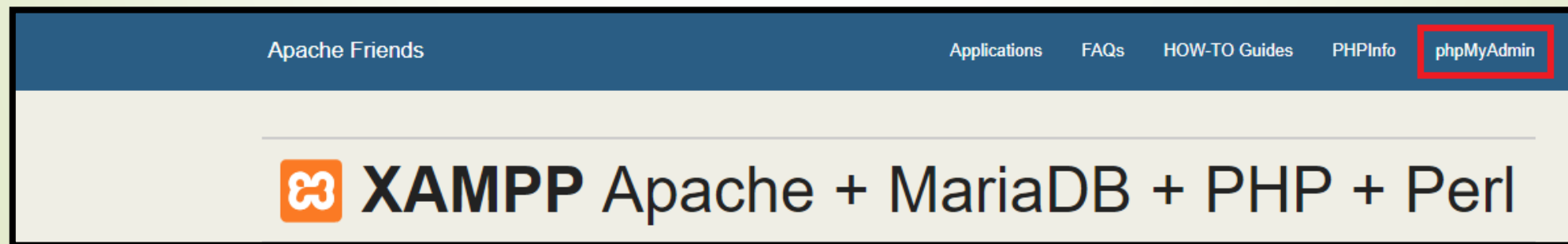


Cargar código SQL en XAMPP

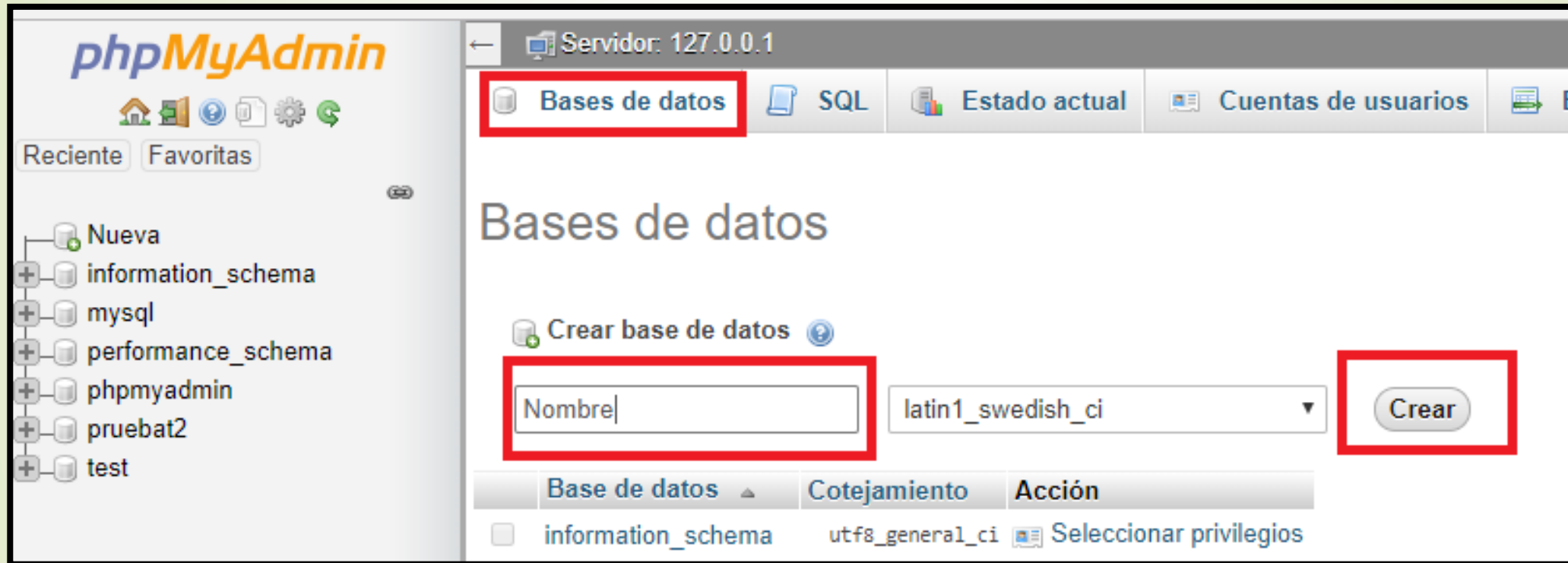
Lanzar módulos con XAMPP Control Panel



En el navegador web meternos en localhost



Crear la base de datos



Introducir el código SQL

Servidor: 127.0.0.1 » Base de datos: Nombre

Estructura **SQL** Buscar Generar una consulta Exportar Importar Operaciones Privilegios Rutinas Más

Ejecutar la(s) consulta(s) SQL en la base de datos Nombre: ?

```
1 DROP TABLE IF EXISTS EMPLEADOS;
2 DROP TABLE IF EXISTS DEPARTAMENTOS;
3
4
5 CREATE TABLE DEPARTAMENTOS
6 (
7   id INT(2),
8   nombre VARCHAR(20),
9   localizacion VARCHAR(50),
10  PRIMARY KEY(id)
11 );
12
13 INSERT INTO DEPARTAMENTOS VALUES (10, 'CONTABILIDAD', 'NEW YORK');
14 INSERT INTO DEPARTAMENTOS VALUES (20, 'INVESTIGACION', 'DALLAS');
15 INSERT INTO DEPARTAMENTOS VALUES (30, 'VENTAS', 'CHICAGO');
```

☐ Enlazar parámetros ?

Guardar esta consulta en favoritos:

[Delimitador] ☒ Mostrar esta consulta otra vez ☐ Mantener la caja de texto con la consulta ☐ Deshacer («rollback») al finalizar ☒ Habilite la revisión de las claves foráneas

- ❑ A continuación presentamos un ejemplo sencillo para MySQL.

```
CREATE TABLE DEPARTAMENTOS
(
  id INT(2),
  nombre VARCHAR(20),
  localizacion VARCHAR(50),
  PRIMARY KEY(id)
);
```

```
CREATE TABLE EMPLEADOS
(
  id INTEGER(4) AUTO_INCREMENT,
  apellido VARCHAR(20) UNIQUE,
  cargo VARCHAR(9),
  jefe INTEGER(4),
  fecha_alta DATE,
  salario DOUBLE,
  comision DOUBLE,
  departamento INTEGER(4),
  PRIMARY KEY(id)
);
```


POM.xml

```
<dependencies>

  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java
  - Reemplazar driver MYSQL por el driver de la base de datos
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.25</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```


- ❑ Una vez que tenemos la base de datos cargada y el proyecto preparado para acceso a datos, vamos a introducir el código de la aplicación.
- ❑ En primer lugar es necesario objeto Connection para realizar operaciones.

```
// Establecemos la conexión con la BD
Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/empresaBD",
"root", //usuario de la BD
""); //contraseña
```

- ❑ Posteriormente preparamos la consulta y la ejecutamos.

```
// Preparamos la consulta
Statement sentencia = conexion.createStatement();
String sql = "SELECT * FROM departamentos";
ResultSet resul = sentencia.executeQuery(sql);
```

- ❑ Eso produce un objeto de la clase ResultSet que contiene los datos pedidos. Dichos datos los recorreremos mediante un bucle usando el objeto ResultSet.

```
while (resul.next()) {
    salida+=resul.getInt(1)+"\n"+
        resul.getString(2)+"\n"+
        resul.getString(3);
}
```

- ❑ El método `next` nos da la siguiente fila y `getTipo` nos devuelve una columna de la tabla.
- ❑ Podemos acceder a los campos de la fila poniendo los nombres directamente en lugar de las posiciones.

```
while (resul.next()) {  
    salida+=resul.getInt("id")+"\n"+  
           resul.getString("nombre")+"\n"+  
           resul.getString("localizacion");  
}
```

- ❑ Por ultimo mostramos los resultados y liberamos recursos.

```
System.out.println("TABLA DEPARTAMENTOS: \n"+salida);  
resul.close(); // Cerrar ResultSet  
sentencia.close(); // Cerrar Statement  
conexion.close(); // Cerrar conexión
```

- ❑ Tener en cuenta que hay excepciones que tratar. En este caso para no ser dependiente el código de cuando se liberan recursos usamos finally.

```
Connection conexion = null;
Statement statement = null;
ResultSet resultSet = null;

try {

    // 1. crear conexion
    conexion = DriverManager.getConnection(
        "jdbc:mysql://127.0.0.1:3306/taller",
        "root", "admin");

    // 2. Sentencia
    statement = conexion.createStatement();
    resultSet = statement.executeQuery(
        "SELECT * FROM coches;");

}
```

```
} catch (SQLException exception) {
    System.out.println("Error de SQL\n"+exception.getMessage());
    exception.printStackTrace();
} finally {
    try{
        if (resultSet != null) resultSet.close();
        if (statement != null) statement.close();
        if (conexion != null) conexion.close();
    }catch (Exception e) { e.printStackTrace();}
}
```

- ❑ Obviamente lo interesante de todo esto es crear consultas SQL parametrizadas.

```
//La variable loc contiene una entrada de datos de usuario  
String selectStatement = "SELECT * FROM departamentos WHERE localizacion = "+ loc;
```

- ❑ También se pueden usar para esto consultas parametrizadas que se basan en usar una clase PreparedStatement en lugar de clase Statement.

```
String selectStatement = "SELECT * FROM departamentos WHERE localizacion =? ";  
PreparedStatement sentencia = con.prepareStatement(selectStatement);  
sentencia.setString(1, localizacion);  
ResultSet resul = sentencia.executeQuery();
```


- ❑ Su principal beneficio es que ignoran los metacaracteres añadiendo seguridad frente a la inyección de código. Otro ejemplo con más parámetros.

```
String selectStatement = "SELECT * FROM empleados WHERE ocupacion =? AND fecha_alta=?";
PreparedStatement sentencia = con.prepareStatement(selectStatement);
sentencia.setString(1, ocupacion);
sentencia.setString(2, fecha_alta);
ResultSet resul = sentencia.executeQuery();
```

- ❑ Algunos ejemplos de inyección de código para obtener acceso y/o información sin credenciales.
 - ❑ <https://www.mclibre.org/consultar/php/lecciones/php-db-inyeccion-sql.html#inyeccion-1>
 - ❑ <https://www.securityartwork.es/2013/11/21/evasion-de-autenticacion-con-inyeccion-sql/>

❑ Más métodos de la clase **ResultSet**:

- ✓ `previous()`: Retrocede el cursor a la siguiente fila de `ResultSet`.
- ✓ `first()`: Apunta a la primera fila del `ResultSet`.
- ✓ `last()`: Apunta a la última fila del `ResultSet`.
- ✓ `getRow()`: Devuelve el índice(int) de la fila actual.
- ✓ `getDouble(int)` / `getDouble(String)`: Devuelve una columna decimal
- ✓ `getBoolean(int)` / `getBoolean(String)`: Devuelve una columna de booleanos
- ✓ `getDate(int)` / `getDate(String)`: Devuelve una columna de fechas
- ✓ `getMetaData`: Devuelve un objeto `ResultSetMetaData` con información extra

❑ Métodos de **ResultSetMetaData**:

- ✓ `getColumnCount()`: Devuelve el número de columnas de la consulta.
- ✓ `getColumnName(int)`: Devuelve el nombre de una columna según la posición.
- ✓ `getColumnType(int)`: Devuelve el tipo de datos de una columna según la posición.

- ❑ Para operaciones de manipulación de datos y de definición se usa el método `executeUpdate`.

Ejemplo de INSERT INTO

```
String dep = 50; // num. departamento
String dnombre = "MARKETING"; // nombre
String loc = "GRANADA"; // localidad

//construir orden INSERT
String sql = "INSERT INTO departamentos VALUES ('"+nombre+"', '"+localizacion+"')";
System.out.println(sql);
Statement sentencia = conexion.createStatement();
int filas=0;
//Ejecucion orden INSERT
filas = sentencia.executeUpdate(sql);
System.out.println("Filas afectadas: " + filas);
sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión
```

Distintos usos del INSERT INTO

//LAS FORMAS SON EQUIVALENTES

Si queremos usar el autoincrement

```
INSERT INTO USUARIOS (LOGIN,PASS,TIPO) VALUES ('PEPE','43242','NORMAL');  
INSERT INTO USUARIOS VALUES (NULL,'PEPE','43242','NORMAL');
```

Si queremos establecer la clave primaria directamente

```
INSERT INTO USUARIOS VALUES (100,'PEPE','43242','NORMAL');
```

Ejemplo de UPDATE

```
String dep = "10", subida = "100";  
String sql = "UPDATE empleados SET salario = salario + "+subida+"  
            WHERE dept_no = "+dep;  
Statement sentencia = conexion.createStatement();  
int filas = sentencia.executeUpdate(sql);  
System.out.println("Empleados modificados: "+filas);  
sentencia.close(); // Cerrar Statement  
conexion.close(); // Cerrar conexión
```

Ejemplo de DELETE

```
String localizacion = "GRANADA";  
String sql = "DELETE FROM departamentos  
            WHERE localizacion = "+localizacion;  
  
Statement sentencia = conexion.createStatement();  
int filas = sentencia.executeUpdate(sql);  
System.out.println("Departamentos borrados: "+filas);  
sentencia.close(); // Cerrar Statement  
conexion.close(); // Cerrar conexión
```

Ejemplo de INSERT INTO con PreparedStatement

```
String sql = "INSERT INTO departamentos VALUES ('?', '?')";  
PreparedStatement sentencia = con.prepareStatement(sql);  
sentencia.setString(1, nombre);  
sentencia.setString(2, localizacion);
```


Ejemplo de UPDATE con PreparedStatement

```
String sql = "UPDATE empleados SET salario = ? WHERE dept_no = ?";  
PreparedStatement sentencia = con.prepareStatement(sql);  
sentencia.setString(1, salario);  
sentencia.setString(2, departamento);
```

Ejemplo de DELETE con PreparedStatement

```
String sql = "DELETE FROM departamentos  
            WHERE localizacion = ?";  
PreparedStatement sentencia = con.prepareStatement(sql);  
sentencia.setString(1, localizacion);
```

- ❑ También podemos controlar transacciones para evitar inconsistencia en la bases de datos.

```
conexion.setAutoCommit(false);  
...  
try {  
    sentencia = conexion.createStatement();  
    sentencia.executeUpdate("UPDATE cuentascorrientes SET saldo=saldo + 200" +  
                            "WHERE numero_cuenta='11234343'");  
    sentencia.executeUpdate("UPDATE cuentascorrientes SET saldo=saldo - 200" +  
                            "WHERE numero_cuenta='34365211'");  
    conexion.commit();  
    ...  
} catch SQLException ex) {  
    System.out.println("ERROR:al hacer un Insert");  
    conexion.rollback();  
}
```

```
CREATE TABLE CUENTASCORRIENTES  
(  
    id INT(2),  
    numero_cuenta VARCHAR(20),  
    saldo DOUBLE,  
    PRIMARY KEY(id)  
);
```

Bibliografía

- ❑ **Ramos Martín, Alicia y Ramos Martín, M^aJesús:**
“Acceso a Datos”. Editorial Garceta. 2012
- ❑ **Córcoles Tendero, J.Ed. y Montero Simarro, Francisco:**
“Acceso a Datos. CFGS”. Editorial Ra-Ma. 2012
- ❑ **“Sistemas abiertos”. Contenidos de la asignatura.**
<http://deim.urv.cat/~pedro.garcia/SOB/>
Última visita: Septiembre 2017.
- ❑ **Ejercicios Acceso a Datos 2^a CFGS DAM**
<https://github.com/andresmr/AcessoDatos>
Última visita: Septiembre 2017
- ❑ **Master en desarrollo de aplicaciones Android. Universidad Politécnica de Valencia**
<http://www.androidcurso.com/index.php/recursos/42-unidad-9-almacenamiento-de-datos/299-preferencias>
Última visita: Septiembre 2017