M6.UF2.A2
CONEXIÓN JDBC Y METADATA

Eduard Lara

INDICE

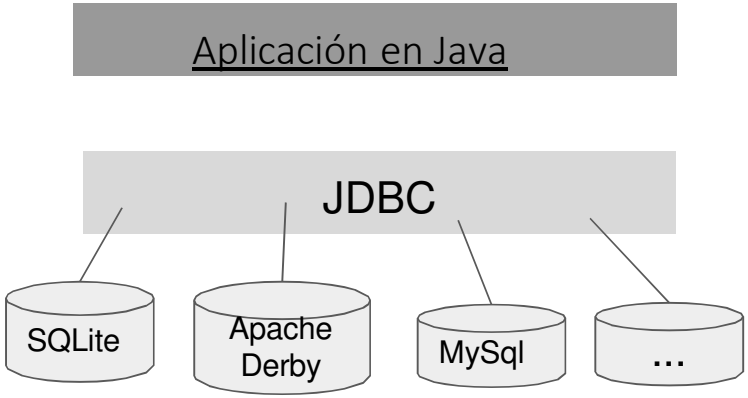
1. INTRODUCCION

- En tecnologías de base de datos podemos encontrarnos con dos normas de conexión a una base de datos:
  - **ODBC**: Open Database Connectivity. Define una API que pueden usar las aplicaciones para abrir una conexión con una base de datos, enviar consultas, actualizaciones y obtener resultados. Las aplicaciones pueden usar esta API siempre y cuando el servidor de base de datos sea compatible con ODBC
  - **JDBC.** Java Database Connectivity. Define una API que pueden usar los programas Java para conectarse a los servidores de bases de datos relacionales
  - **OLE-DB**. Object Linking and Embedding for Databases. De Microsoft. Es una API de C++ con objetivos parecidos a los de ODBC pero para orígenes de datos que no son bases de datos. En ODBC los comandos siempre están en SQL, en OLE-DB pueden estar en cualquier lenguaje soportado por el origen de datos.

1.1. ACCESO A DATOS MEDIANTE JDBC

- JDBC proporciona una librería estándar para acceder a fuentes de datos principalmente orientados a bases de datos relacionales que usan SQL.
- No solo provee una interfaz sino que también define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones Java el acceso a los datos.
- JDBC dispone de una interfaz distinta para cada base de datos, es lo que llamamos **driver** (controlador o conector). Esto permite que las llamadas a los métodos Java de las clases JDBC se correspondan con el API de la base de datos.

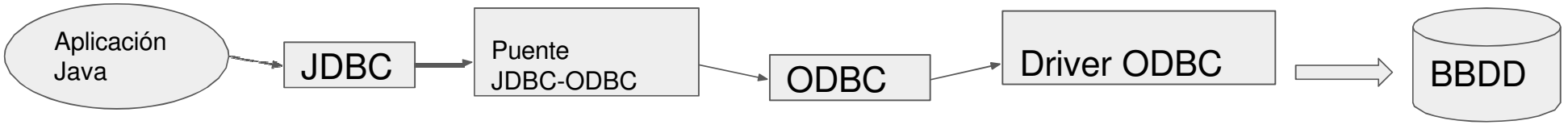
1.1. ACCESO A DATOS MEDIANTE JDBC



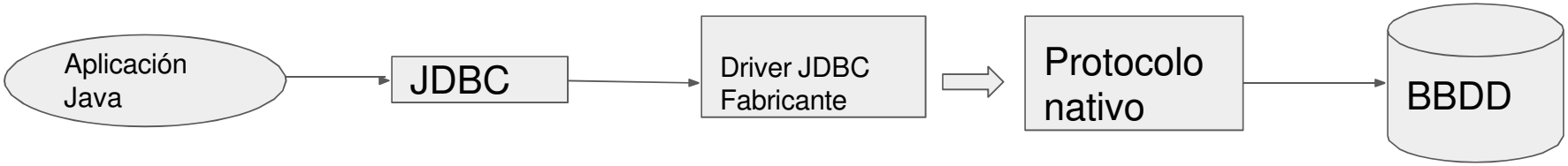
- JDBC consta de un conjunto de clases e interfaces que nos permiten escribir aplicaciones Java para gestionar las siguientes tareas con una bbdd relacional:
  - Conectarse a la base de datos
  - Enviar consultas e instrucciones DML a la bbdd
  - Recuperar y procesar los resultados recibidos

1.2. TIPOS DE DRIVERS JDBC

- **Tipo 1 JDBC-ODBC Bridge**: permite el acceso a bases de datos JDBC mediante un driver ODBC. Convierte las llamadas al API de JDBC en ODBC. Exige la instalación y configuración de ODBC en la máquina cliente

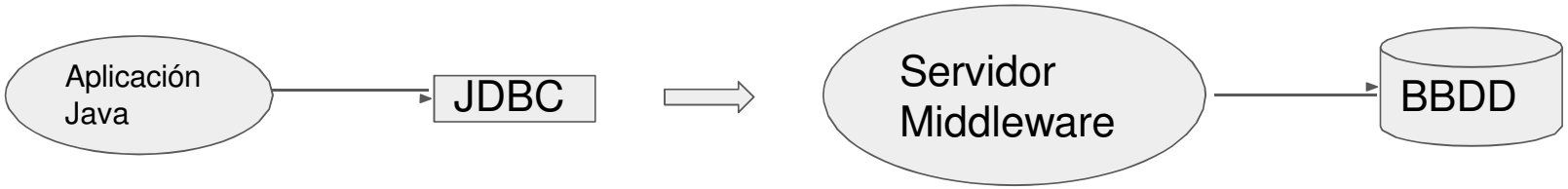


- **Tipo 2 Native**: controlador escrito parcialmente en Java y en código nativo de la base de datos. Traduce las llamadas al API de JDBC Java en llamadas propias del motor de base de datos. Exige instalar en la máquina cliente código binario propio del cliente de base de datos y del sistema operativo.



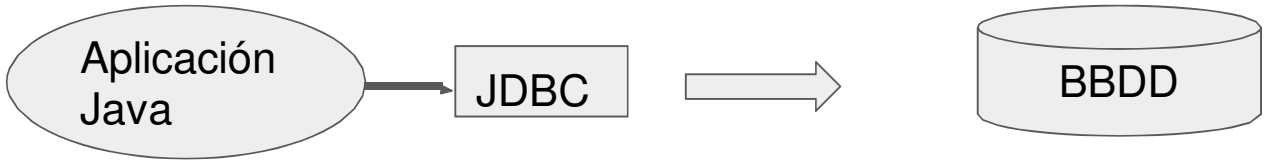
1.2. TIPOS DE DRIVERS JDBC

- **Tipo 3 Network:** controlador de Java puro que utiliza un protocolo de red (por ejemplo HTTP) para comunicarse con el servidor de base de datos. Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red y a continuación son traducidas por un software intermedio (Middleware) al protocolo usado por la bbdd. No exige instalación en cliente.



1.2. TIPOS DE DRIVERS JDBC

- **Tipo 4 Thin:** controlador de Java puro con protocolo nativo. Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red usado por el motor de base de datos. No exige instalación en cliente.



Los tipos 3 y 4 son la mejor forma de acceder. Los tipos 1 y 2 se usan normalmente cuando no queda otro remedio. En la mayoría de los casos la opción más adecuada será el tipo 4.

1.3. COMO FUNCIONA JDBC

- JDBC define varias interfaces que permiten realizar operaciones con bases de datos; a partir de ellas se derivan las clases correspondientes.
- Estas clases están definidas en el paquete **java.sql**
- El funcionamiento de un programa con JDBC requiere los siguientes pasos:
  - Importar las clases necesarias
  - Cargar el **driver** JDBC
  - Identificar el origen de datos
  - Crear un objeto **Connection**
  - Crear un objeto **Statement**
  - Ejecutar una consulta con el objeto **Statement**
  - Recuperar los datos del objeto **ResultSet**
  - Liberar sucesivamente **ResultSet, Statement, Connection**

1.3. COMO FUNCIONA JDBC

```
try {
    //Para el Driver mysql-connector-java-5.1.47.jar
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
    String conex = "jdbc:mysql://localhost:3306/futbol_mysql";
    Connection conexion=DriverManager.getConnection(conex,"root","");

    Statement sentencia = conexion.createStatement();
    String sql = "select `CodEq`,`Equipo`,`Ciudad`,`Estadio` from equipos";
    ResultSet result = sentencia.executeQuery(sql);
    while(result.next()) {
        System.out.println(result.getString(1) + " : " + result.getString(2) + " : " +
                result.getString(3) + " : " + result.getString(4));
    }
    result.close();
    sentencia.close();
    conexion.close();
}
catch(Exception ex)
{       System.out.print(ex.getMessage());  }
```

1.3. COMO FUNCIONA JDBC

Cargar driver

En primer lugar se carga el driver con el método forName() de la clase Class (java.lang). Para ello se le pasa un objeto String con el nombre de la clase del driver como argumento. En el ejemplo, como se accede a una base de datos MySql, necesitamos cargar el driver com.mysql.cj.jdbc.Driver:

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

Establecer conexión

A continuación se establece la conexión con la base de datos. El servidor MySql debe estar rodando. Usamos la clase DriverManager con el método getConnection() de la siguiente manera:

```
Connection conexion=DriverManager.getConnection
("jdbc:mysql://localhost:3306/futbol_mysql","root","");
```

1.3. COMO FUNCIONA JDBC

Método getConnection()

public static Connection getConnection (String url, String user, String password) throws SQLException

El primer parámetro del método getConnection() representa la URL de conexión a la bbdd. Tiene el siguiente formato para conectarse a MySql:

jdbc:mysql://nombre_host:puerto/nombre_basedatos

- **jdbc:mysql** → indica que estamos utilizando un driver JDBC para MySql
- **nombre_host** → **indica** el nombre del servidor donde está la base de datos. Aquí puede ponerse una IP, el nombre de máquina, o localhost
- **puerto** → puerto donde está escuchando el servidor MySql. Por defecto es el 3306. Si no se indica se asume que es este valor.
- **nombre_basedatos** → nombre de la base de datos a la que queremos conectarnos. Debe existir previamente en MySql.

1.3. COMO FUNCIONA JDBC

Ejecutar sentencias SQL

- A continuación se realiza la consulta, para ello recurrimos a la interfaz **Statement** para crear una sentencia. Para obtener un objeto **Statement** se llama al método **createStatement()** de un objeto **Connection** válido. La sentencia obtenida (o objeto obtenido) tiene el método **executeQuery()** que sirve para realizar una consulta en la base de datos, se le pasa un String en que está la consulta SQL:

```
Statement sta =conn.createStatement();
String sql = "select `CodEq`,`Equipo`,`Ciudad`,`Estadio` from equipos";
ResultSet result = sentencia.executeQuery(sql);
```

- El resultado nos lo devuelve como un **ResultSet,** que es un objeto similar a una lista en la que está el resultado de la consulta. Cada elemento de la lista es uno de los registros de la tabla “departamentos”.

1.3. COMO FUNCIONA JDBC

Recorrer la lista mediante el método next()

- Los métodos **getInt()** y **getString()** nos van devolviendo los valores de los campos de cada registro. Entre paréntesis se pone la posición de la columna en la tabla. También se puede poner una cadena que indica el nombre de la columna:

```
while (result.next()){
        System.out.printf("%d, %s, %s, %n",  result.getInt(1),
        result.getString(2),  result.getString(3));
}
 while (result.next()){
                System.out.printf("%d, %s, %s, %n",  result.getInt("codeq"),
                result.getString("equipo"),  result.getString("ciudad"));
 }
```

1.4. COMO FUNCIONA JDBC

Clase ResultSet

Dispone de varios métodos para mover el puntero que apunta a cada uno de los registros devueltos por la consulta:

- **boolean next()** → Mueve el puntero una fila hacia adelante a partir de la posición actual. Devuelve **true** si el puntero se posiciona correctamente y **false** si no hay registros en **ResultSet.**
- **boolean first()** → Mueve el puntero al primer registro de la lista.
- **boolean last()** → Mueve el puntero al último registro de la lista.
- **boolean previous()** → Mueve el puntero al registro anterior de la posición actual.
- **void beforeFirst()** → Mueve el puntero justo antes del primer registro.
- **int getRow()** → Devuelve el número de registro actual. Para el primer registro devuelve 1, para el segundo 2 y así sucesivamente.

Por último, se liberan todos los recursos y se cierra la conexión

- **result.close(); sentencia.close(); conexion.close();**

1.4. COMO FUNCIONA JDBC

Ejecución DML y DDL

Un objeto **Statement** crea un espacio de trabajo para realizar consultas SQL, ejecutarlas y recibir los resultados. Se pueden usar los siguientes métodos:

- **ResultSet executeQuery(String)** → para sentencias SELECT
- **int executeUpdate(String)** → se utiliza para sentencias que no devuelvan un **ResultSet** como son las sentencias DML: INSERT, UPDATE Y DELETE; y las sentencias DDL: CREATE, DROP y ALTER. El método devuelve un entero indicando el número de filas que se vieron afectadas y, en caso de las sentencias DDL, devuelve 0.
- **boolean execute(String)** → se puede utilizar para ejecutar cualquier sentencia SQL. Tanto las que devuelven un **ResultSet** (SELECT), como para las que devuelven el número de filas afectadas (INSERT, UPDATE, DELETE) y para las de definición de datos (CREATE). El método devuelve *true* si devuelve un **ResultSet** (para recuperar las filas será necesario llamar al método **getResultSet()**) y *false* si se trata de actualizaciones o no hay resultados; en este caso se usará el método **getUpdateCount()** para recuperar el valor devuelto de filas afectadas.

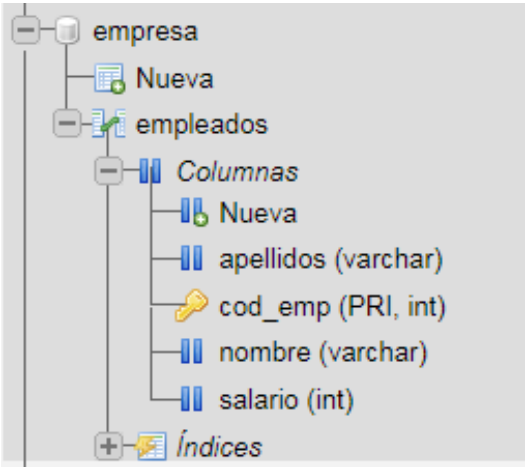
1.4. COMO FUNCIONA JDBC

```
try {
    //Para el Driver mysql-connector-java-5.1.47.jar
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
    String conex = "jdbc:mysql://localhost:3306/futbol_mysql";
    Connection conexion=DriverManager.getConnection(conex,"root","");

    String sql = "INSERT INTO EQUIPOS (`codeq`, `equipo`, `ciudad`, `estadio`) VALUES (?, ?, ?, ?);";
    PreparedStatement sentencia= conexion.prepareStatement(sql);
    sentencia.setInt(1,70 );
    sentencia.setString(2, "Manresa FC");
    sentencia.setString(3, "Igualada");
    sentencia.setString(4, "Congost");
    System.out.print(sentencia);

    int filas = sentencia.executeUpdate(sql);
    System.out.print("Filas afectadas: "+ filas);
    sentencia.close();
    conexion.close();
}
catch(Exception ex)
{       System.out.print(ex.getMessage());   }
```

2. CRUD CON JDBC-MYSQL

1. Realiza una aplicación en modo no grafico que realice un mantenimiento CRUD sobre la tabla empleados de la base de datos “empresa” de mysql.

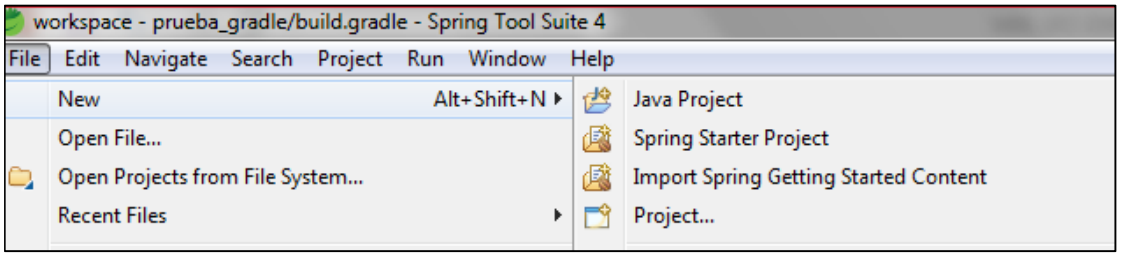
```
create table empleados(
    cod_emp int primary key auto_increment,
    nombre varchar(20),
    apellidos varchar(20),
    salario double
);
```

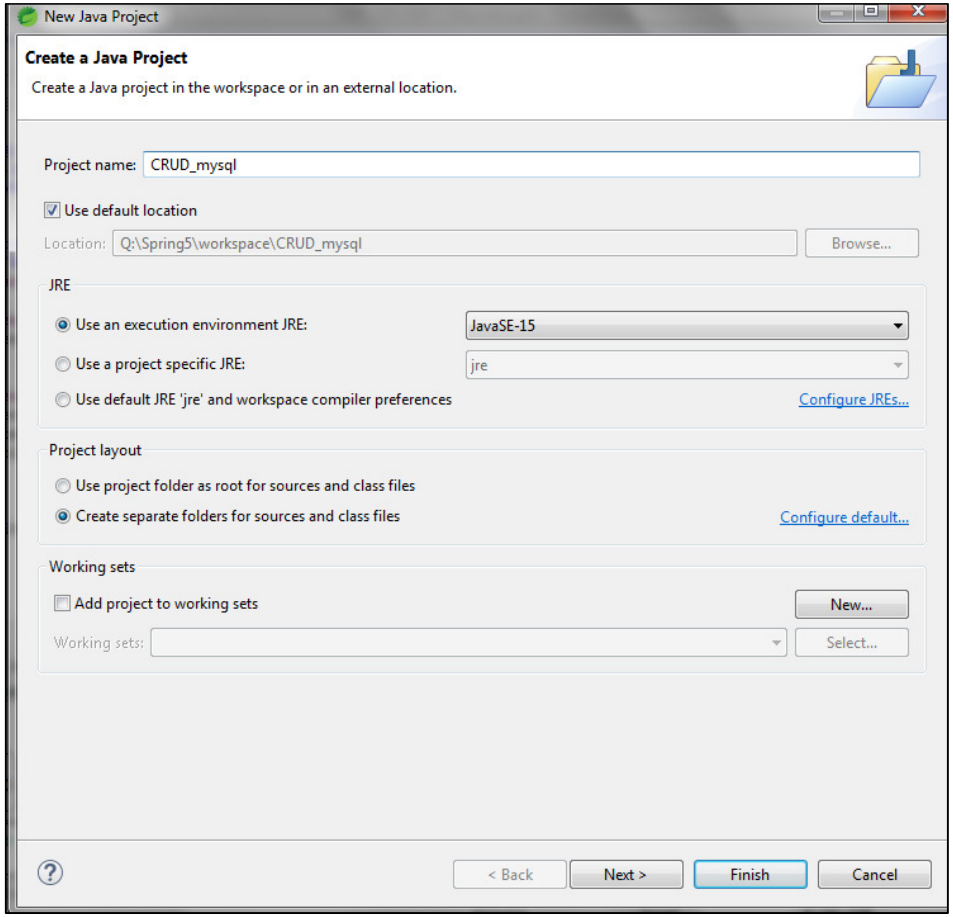


2. Debe ofrecer un menú con las siguientes opciones:
   a) Visualizar la lista de empleados
   b) Actualizar el salario de un empleado
   c) Insertar un nuevo empleado.
   d) Borrar un empleado.
   e) Salir

2. CRUD CON JDBC-MYSQL

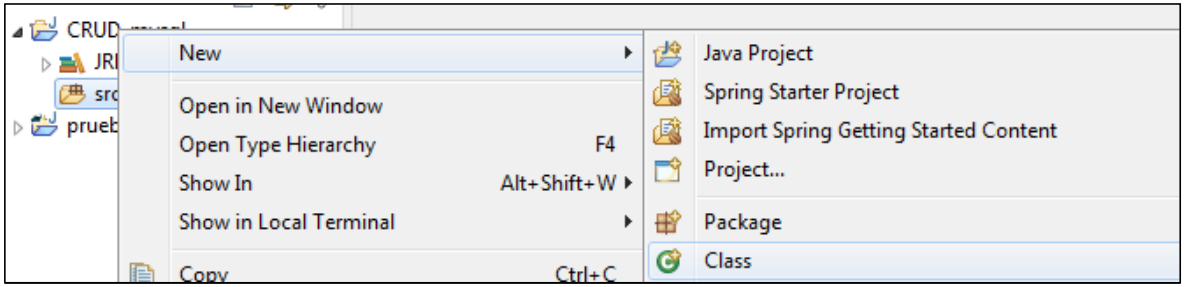
Paso 1. Primero de todo creamos un proyecto Java, de nombre por ejemplo CRUD_mysql:

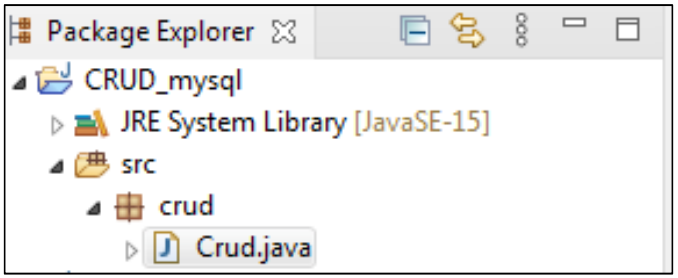


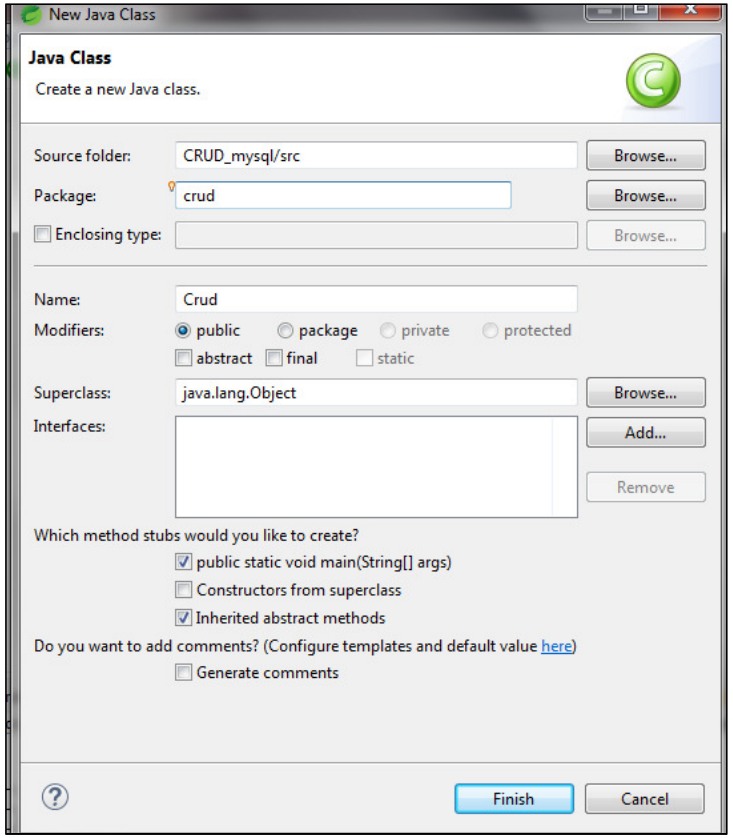


2. CRUD CON JDBC-MYSQL

Paso 2. Nos situamos sobre la carpeta src, y hacemos click botón derecho y seleccionamos New/Class. Pondremos nombre Crud, por ejemplo. Debemos activar la casilla de public static void main:







2. CRUD CON JDBC-MYSQL

Paso 3. Escribimos el siguiente código en el main del programa, en 3 fases diferenciadas:

- Carga del driver Mysql Java
- Instanciacion e inicialización de la variable Connection de conexión con la base de datos
- Creación de un CRUD mediante un simple while con las diferentes acciones que queremos implementar en la base de datos

```
public class Crud {

    static Scanner reader;
    static Connection conexion;

    public static void main(String[] args) throws Exception  {

        //CARGA DEL DRIVER MYSQL JAVA
        Class.forName("com.mysql.cj.jdbc.Driver");
        String conex="jdbc:mysql://localhost:3306/empresa";

        //INCIALIZAMOS LA CONEXIÓN A LA BASE DE DATOS
        conexion = DriverManager.getConnection (conex,"root","");

        reader = new Scanner(System.in);
        //CRUD--Create, Read, Update, Delete
        while (true) {
            System.out.println("1. Visualizar la lista de empleados");
            System.out.println("2. Incrementar salario de empleado");
            System.out.println("3. Insertar un nuevo empleado");
            System.out.println("4. Borrar un nuevo empleado");
            System.out.println("5. Ejecuta Procedimiento usuarios");
            System.out.println("6. Ejecuta Function inversa");
            System.out.println("7. Salir");
            System.out.print("Introduce que opcion quieres?");
            int opcion = reader.nextInt();
            switch(opcion) {
                case 1:
                    listar();
                    break;
                case 2:
                    listar();
```

2. CRUD CON JDBC-MYSQL

Paso 3 (Opcional). El nuevo driver versión 8 en determinadas circunstancias podría necesitar de algunas modificaciones en el string de conexión de la llamada desde Java:

```
public static void main(String[] args) {
    try {
        //Para el Driver mysql-connector-java-8.0.15.jar
        Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
        String conex = "jdbc:mysql://localhost/futbol_mysql?"
                + "useUnicode=true&"
                + "useJDBCCompliantTimezoneShift=true&"
                + "useLegacyDatetimeCode=false&"
                + "serverTimezone=UTC";
        Connection conn=DriverManager.getConnection(conex,"root","");
        DatabaseMetaData pepe=conn.getMetaData();
        Statement s = conn.createStatement();
        String selTable = "select * from equipos";
        s.execute(selTable);
        ResultSet rs = s.getResultSet();
        while((rs!=null) && (rs.next()))
            System.out.println(rs.getString(1) + " : " + rs.getString(2) +
                    " : " + rs.getString(3) + " : " + rs.getString(4));
    }
    catch(Exception ex)
    {
        System.out.print(ex.getMessage());
    }
}
```

2. CRUD CON JDBC-MYSQL

Paso 4. Implementamos la función listar. Básicamente lo que hace es un: "SELECT * FROM EMPLEADOS"

```
public static void listar() throws SQLException {
    System.out.println("----------------------------");
            String sql= "select * from empleados;";
    Statement sta= conexion.createStatement();
    ResultSet rs = (ResultSet)sta.executeQuery(sql);
    while (rs.next()) {
        System.out.println(rs.getInt(1) +"-"+ rs.getString(2) + "-"+
                rs.getString(3) +"-" +rs.getString(4));
    }
    System.out.println("----------------------------");

}
```

2. CRUD CON JDBC-MYSQL

Paso 5. Implementamos la función actualizar. Básicamente hace un : "UPDATE EMPLEADOS SET COLUMNA=VALOR WHERE "

```
public static int actualizar() throws SQLException {
    String sql= "update `empleados` set `salario` = `salario` + 10000 "
              + " where `cod_emp`=?;";
    System.out.print("Introduce id de empleado: ");

    int id = reader.nextInt();
    PreparedStatement sta = conexion.prepareStatement(sql);
    sta.setInt(1, id);
    return sta.executeUpdate();

}
```

2. CRUD CON JDBC-MYSQL

Paso 6. Implementamos la función insertar. Básicamente hace un : "INSERT INTO EMPLEADOS VALUES () "

```
public static int insertar() throws SQLException {
    System.out.print("Introduce nombre empleado: ");
    String nombre=reader.next();
    System.out.print("Introduce apellido empleado: ");
    String apellidos=reader.next();
    System.out.print("Introduce salario: ");
    int salario=reader.nextInt();
    String sql= "insert into empleados(nombre, apellidos, salario) "
                + " values (?,?,?);";
    PreparedStatement sta = conexion.prepareStatement(sql);
    sta.setString(1, nombre);
    sta.setString(2, apellidos);
    sta.setInt(3, salario);
    return sta.executeUpdate();
}
```

2. CRUD CON JDBC-MYSQL

Paso 7. Implementamos la función borrar. Básicamente hace un : "DELETE FROM EMPLEADOS WHERE () "

```
public static int borrar() throws SQLException {
    String sql= "delete from empleados where cod_emp =?;";
    System.out.print("Introduce id de empleado: ");
    int id=reader.nextInt();
    PreparedStatement sta = conexion.prepareStatement(sql);
    sta.setInt(1, id);
    return sta.executeUpdate();
}
```

2. CRUD CON JDBC-MYSQL

Paso 8. Crea un procedimiento almacenado en Mysql y realiza las llamadas desde Java:

```
CREATE PROCEDURE usuarios()
    select * from mysql.user;
```

```
public static void executeProcedure() throws SQLException {
    // String query = "{ call usuarios(?) }";
    String query = "{ call usuarios() }";
    CallableStatement stmt = conexion.prepareCall(query);
    //stmt.setInt(1, candidateId);

    ResultSet rs = stmt.executeQuery();
    while (rs.next()) {
                System.out.println(rs.getString(1) +"-"+ rs.getString(2) + "-"+
                        rs.getString(3) +"-" +rs.getString(4));
    }
    System.out.println();
}
```

2. CRUD CON JDBC-MYSQL

Paso 9. Crea una función en Mysql y realiza la llamada desde Java:

```
CREATE FUNCTION inversa (PALABRA VARCHAR(20))
            RETURNS VARCHAR(20)
    RETURN REVERSE(PALABRA);
```

```
public static void executeFunction() throws SQLException {
    //Preparing a CallableStatement to call a function
    CallableStatement cstmt = conexion.prepareCall("{? = call inversa(?)}");
    //Registering the out parameter of the function (return type)
    cstmt.registerOutParameter(1, Types.VARCHAR);
    //Setting the input parameters of the function
    cstmt.setString(2, "Pedro");
    //Executing the statement
    cstmt.execute();
    System.out.println("Pedro invertida es : "+cstmt.getString(1));
}
```

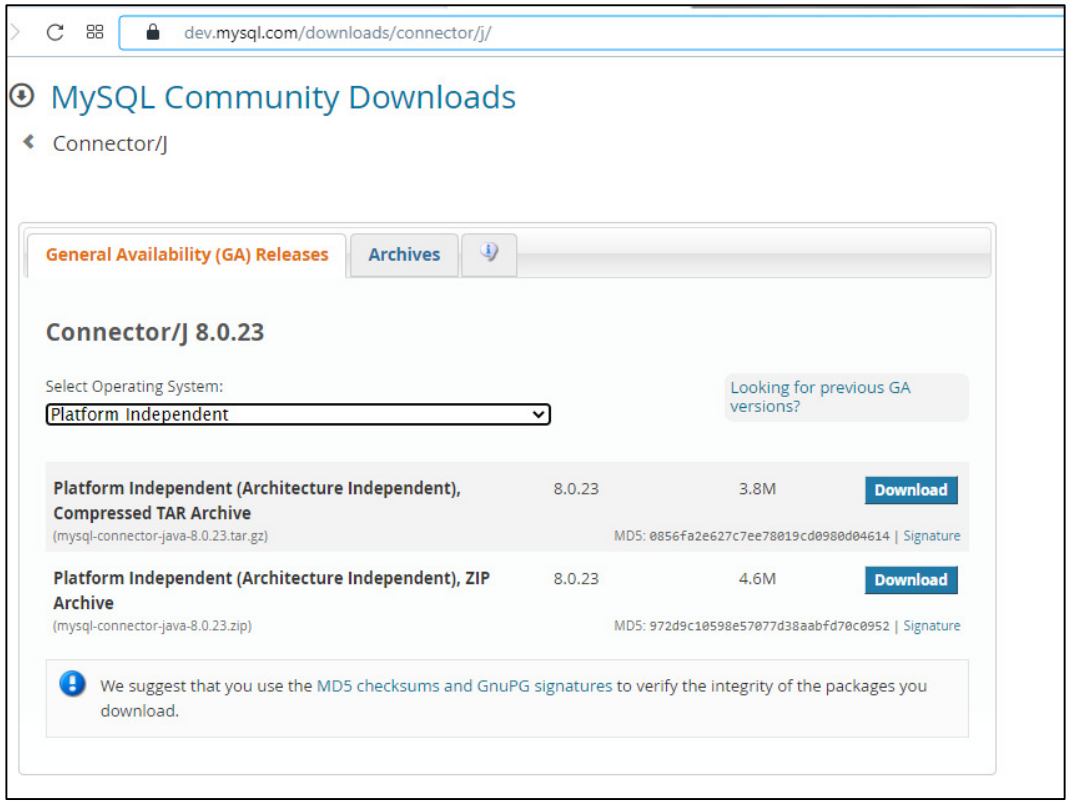
2. CRUD CON JDBC-MYSQL

Paso 10. Tenemos 3 opciones para agregar el driver de conexión Mysql Connector para java en nuestro proyecto:

- Manualmente, descargando el driver jar de la pagina web Mysql Connector
- Mediante el repositorio Maven
- Mediante el repositorio Gradle

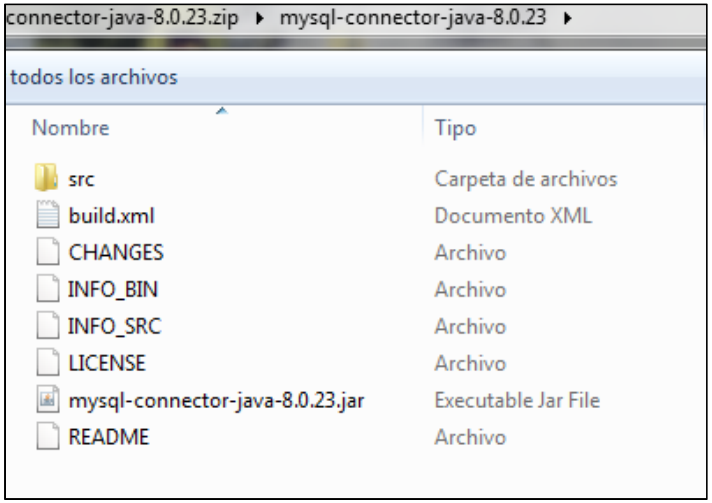
2. CRUD CON JDBC-MYSQL

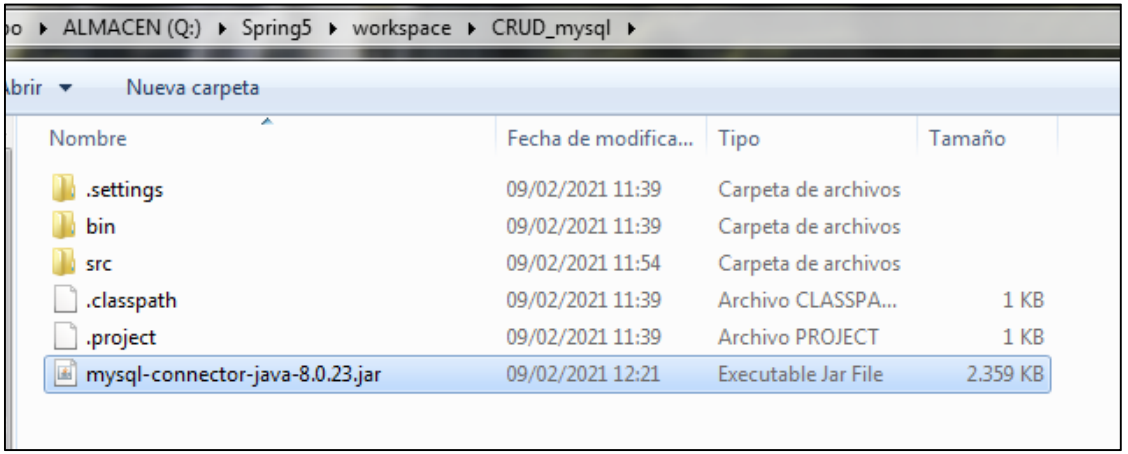
Paso 11 (Manual). Si lo hacemos manualmente, debemos ir a la pagina Mysql Connector https://dev.mysql.com/downloads/connector/j/ y descargar la ultima versión del driver:



2. CRUD CON JDBC-MYSQL

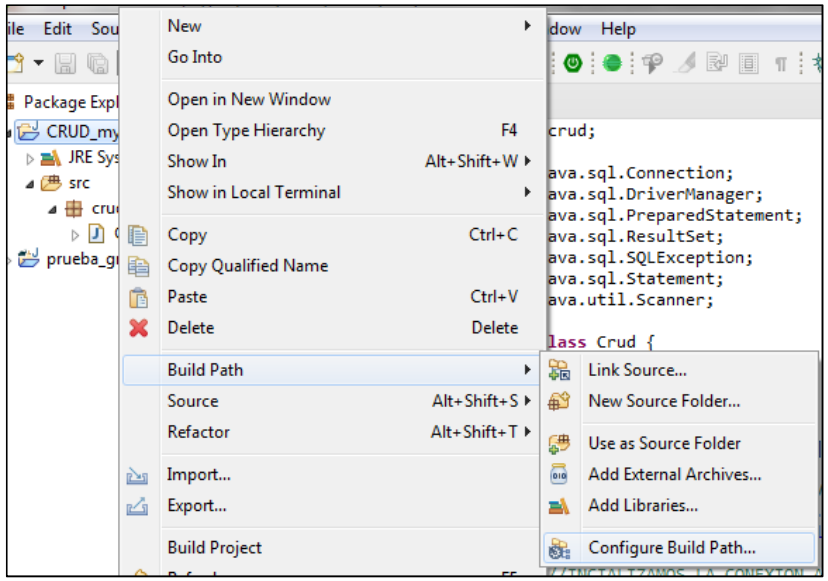
Paso 12 (Manual). Descomprimimos el fichero zip, extraemos el driver jar, y lo agregamos a nuestro proyecto. Dejamos el jar dentro de nuestro proyecto, por si tenemos que hacer una copiar, el jar venga ya incluido

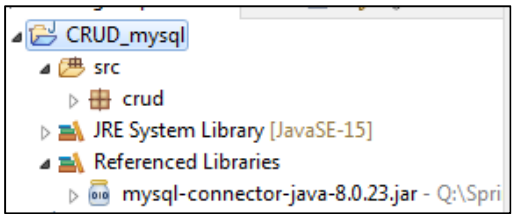


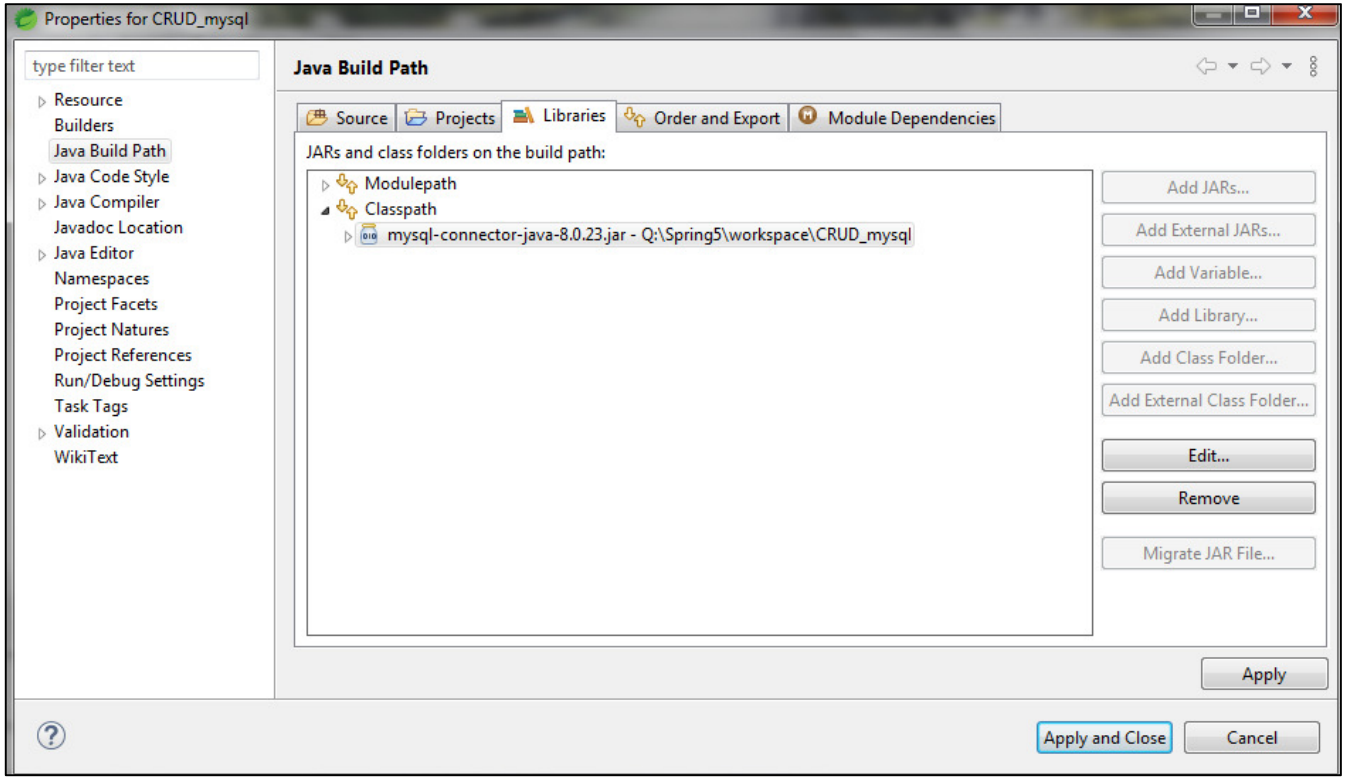


2. CRUD CON JDBC-MYSQL

Paso 13 (Manual). Vamos a la opción Java Build Path y agregamos esta librería a nuestro proyecto:

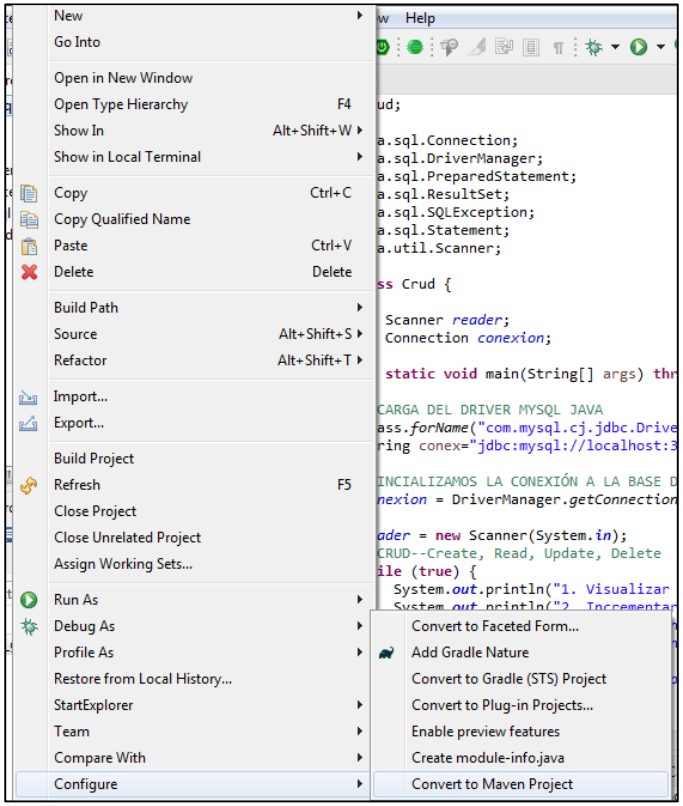


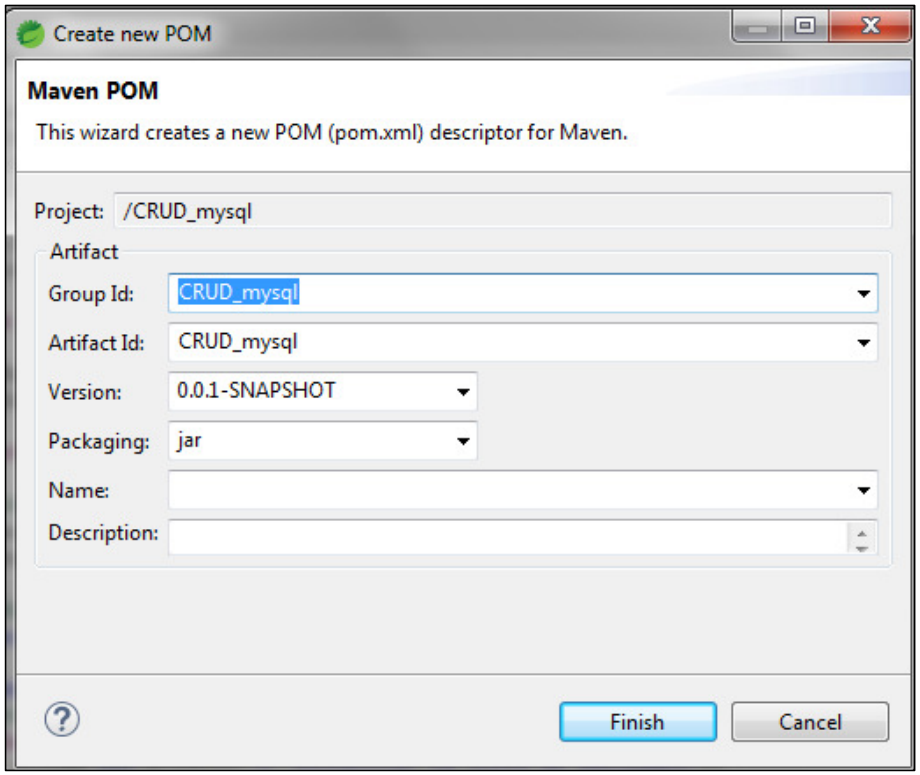




2. CRUD CON JDBC-MYSQL

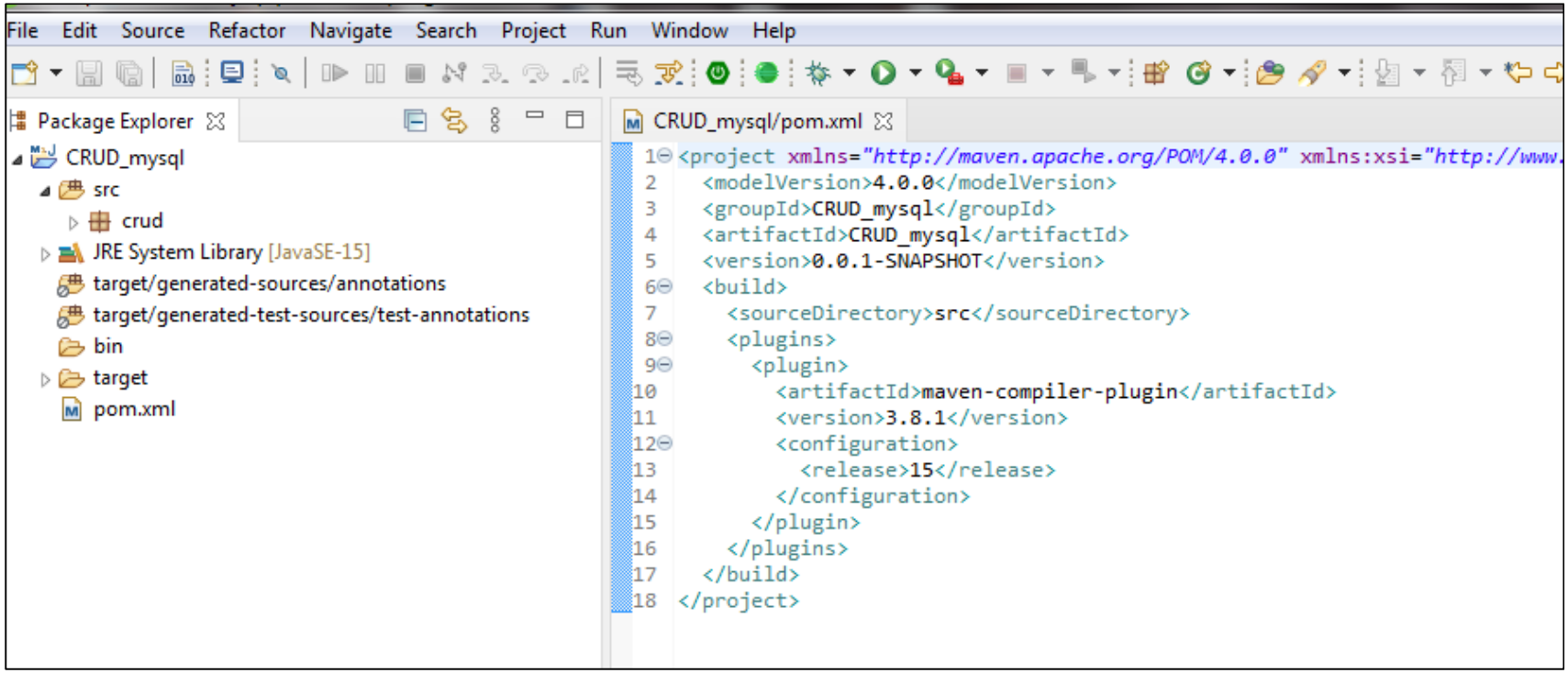
Paso 14 (Maven). Para poder importar la librería mysql connector a nuestro proyecto via Maven, primero debe de convertirse nuestro proyecto en un proyecto Maven:





2. CRUD CON JDBC-MYSQL

Paso 15 (Maven). Una vez convertido nuestro proyecto en proyecto maven, podemos observar el fichero pom.xml, lugar donde se guardan las dependencias que debemos agregar para obtener las librerías del repositorio:

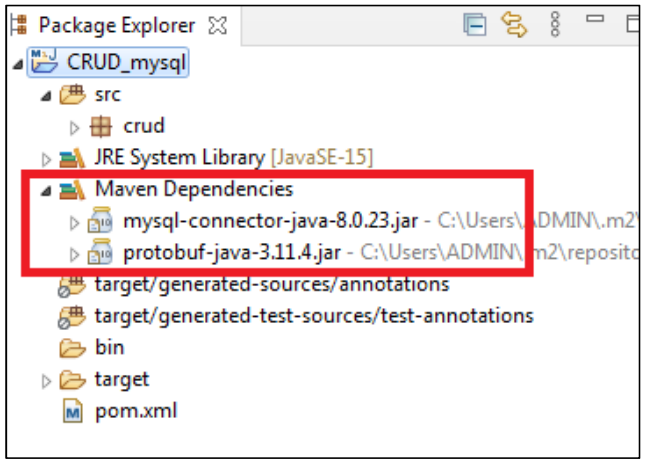


2. CRUD CON JDBC-MYSQL

Paso 16 (Maven). Agregamos la dependencia del mysql connector en el fichero pom.xml

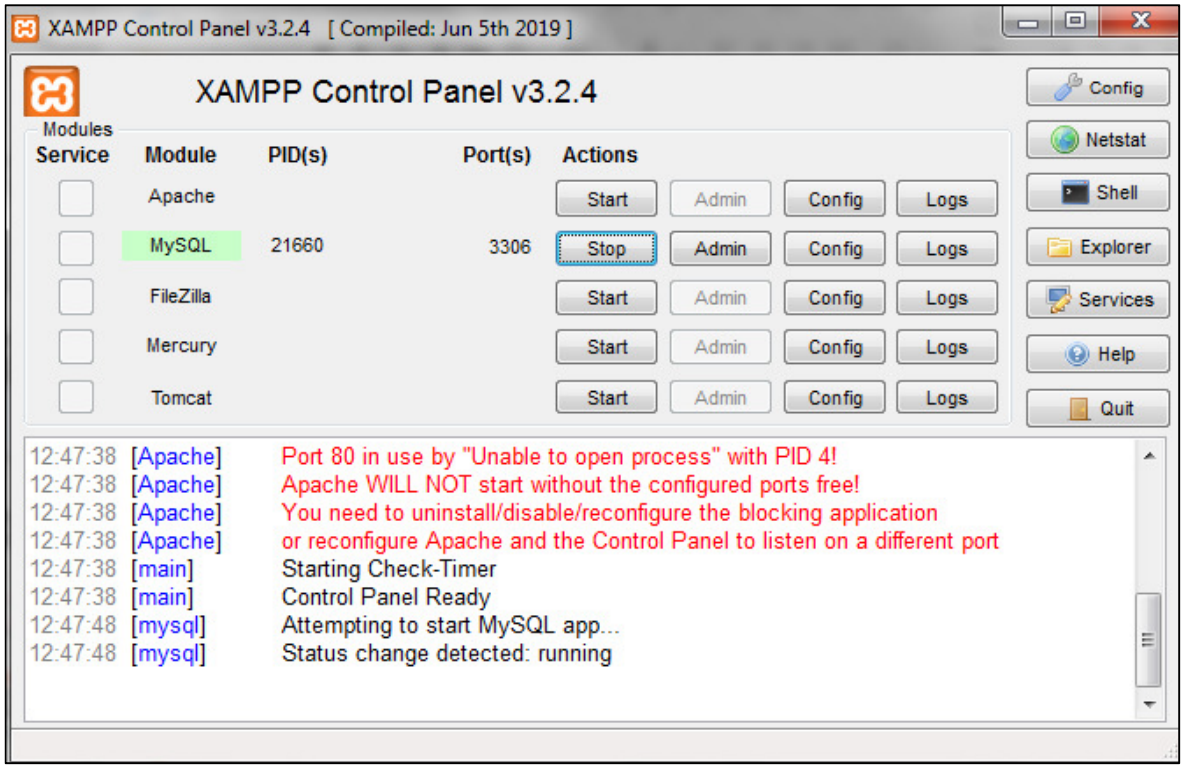
CRUD_mysql/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
  <modelVersion>4.0.0</modelVersion>
  <groupId>CRUD_mysql</groupId>
  <artifactId>CRUD_mysql</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <release>15</release>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
  <dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.23</version>
    </dependency>
  </dependencies>

</project>

```



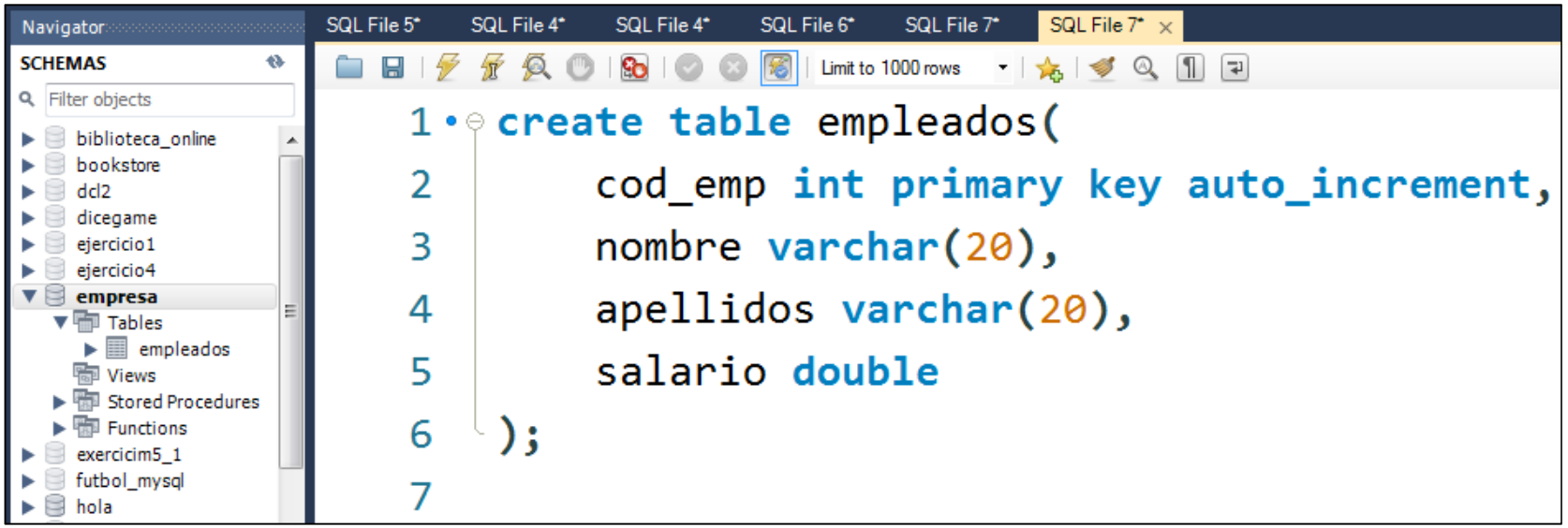
2. CRUD CON JDBC-MYSQL

Paso 17. En el caso de la base de datos mysql, debemos activar previamente el servicio desde el control panel de xampp. Se levanta en el puerto 3306.



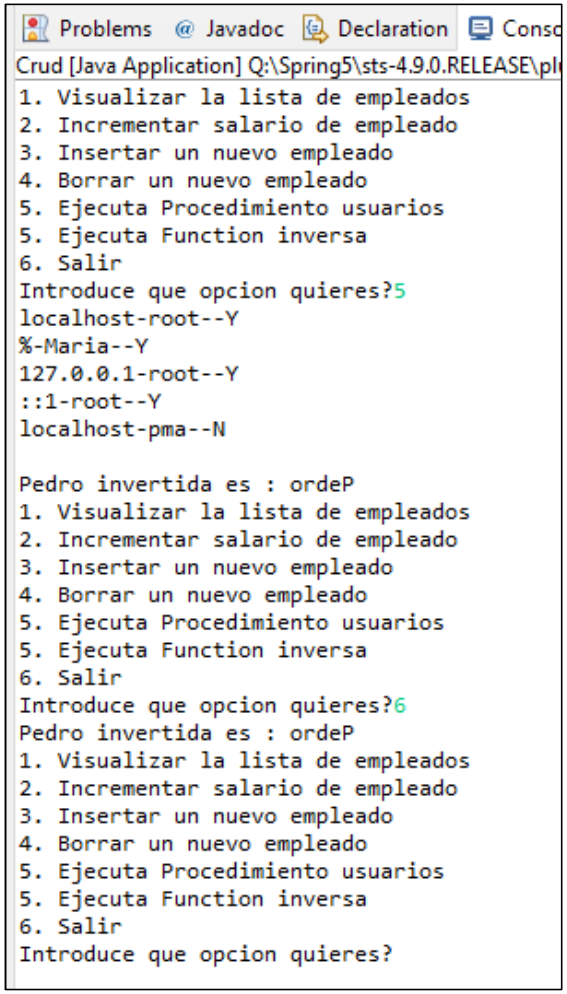
2. CRUD CON JDBC-MYSQL

Paso 18. Desde Phpmyadmin o MySql Workbench podemos crear la base de datos empresa y la tabla empleados si no las tenemos creadas:



2. CRUD CON JDBC-MYSQL

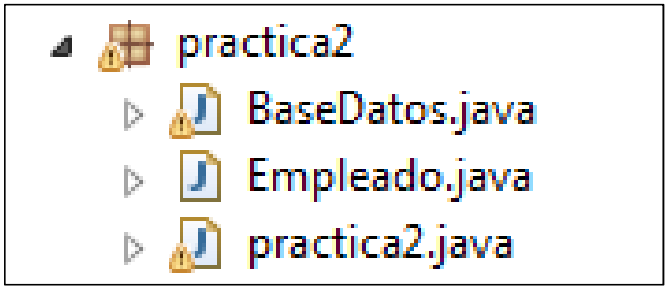
Paso 19. Resultado de la ejecución:



3. EVOLUCION HACIA JPA-HIBERNATE

Dividiremos el único fichero anterior crud.java en otros 3 ficheros:

- BaseDatos.java: Agrupa las funcionalidades del acceso a base de datos en la una clase java. Debe tener un atributo Connection y establecer la conexión en el constructor. Debe contener el resto de funciones de base de datos: insertar, listar, etc
- Empleados.java: La clase pojo Entity que utilizaran las funciones insertar, borrar y actualizar como parámetro de entrada.
- Main.java : Contendrá el programa principal con el bucle y las opciones del menu



3. EVOLUCION HACIA JPA-HIBERNATE

BaseDatos.java

```
11 public class BaseDatos {
12     private Connection conexion;
13 
14     public BaseDatos() {
15         try {
16             Class.forName("com.mysql.cj.jdbc.Driver");
17             String conex="jdbc:mysql://localhost:3306/empresa";
18             this.conexion = DriverManager.getConnection (conex,"root","");
19 
20         } catch (Exception e) {
21             e.printStackTrace();
22         }
23     }
24 
25     public void listar() {  
40     public int actualizar(Empleado emp) {
53     public int insertar(Empleado emp) {
68     public int borrar(Empleado emp) {
80 }
```

3. EVOLUCION HACIA JPA-HIBERNATE

Empleado.java

```java
public class Empleado {
    private int id;
    private String nombre;
    private String apellidos;
    private int salario;

    public Empleado() {
    }

    public Empleado(int id, String nombre, String apellidos, int salario) {
        this.id = id;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.salario = salario;
    }

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
```

3. EVOLUCION HACIA JPA-HIBERNATE

Main.java

```
public static void main(String[] args)  {

    Scanner reader = new Scanner(System.in);
    BaseDatos db = new BaseDatos();
    //CRUD Create - Read - Update - Delete
    while (true) {
        System.out.println("1. Visualizar la lista de empleados");
        System.out.println("2. Incrementar salario de empleado");
        System.out.println("3. Insertar un nuevo empleado");
        System.out.println("4. Borrar un nuevo empleado");
        System.out.println("5. Salir");
        System.out.print("Introduce que opcion quieres?");
        int opcion = reader.nextInt();
        switch(opcion) {
            case 1:
                db.listar();
                break;
            case 2:
                db.listar();
                System.out.print("Introduce id de empleado: ");
                int id = reader.nextInt();
                db.actualizar(new Empleado(id, "","",0));
                db.listar();
                break;
```

4. API METADATA CON JAVA

- ¿Qué pasa si desconocemos la estructura de las tablas de una base de datos? Es decir, ¿cuáles son los campos de una tabla?¿Cómo son las restricciones de integridad? Etc.
- La interfaz **DatabaseMetaData** proporciona dicha información, con los métodos:
  - **getCatalogs** → Información de las bases de datos
  - **getTables()** → Información sobre tablas y vistas
  - **getColumns()** → Información sobre columnas de una tabla
  - **getPrimaryKeys()** → Información sobre columnas que forman la clave primaria de una tabla
  - **getExportedKeys()** → Información sobre claves ajenas (foráneas) que utilizan la clave primaria de una tabla (que apuntan a la tabla)
  - **getImportedKeys()** → Información sobre claves ajenas de una tabla
  - **getProcedures()** → Información sobre procedimientos almacenados.

4. API METADATA CON JAVA

```
try{
    Class.forName("com.mysql.cj.jdbc.Driver");
    String conex="jdbc:mysql://localhost:3306/futbol_mysql";
    Connection conexion = DriverManager.getConnection (conex,"roor","");
    DatabaseMetaData dbmd = conexion.getMetaData();
    ResultSet rs = null;
    String nombre = dbmd.getDatabaseProductName();
    String driver = dbmd.getDriverName();
    String url = dbmd.getURL();
    String usuario = dbmd.getUserName();
    System.out.print("Nombre: "+ nombre +"\ndriver: " + driver +
            "\nURL: " + url +"\nusuario: " +usuario);
    rs = dbmd.getTables(null,"futbol_mysql",null,null);
    while (rs.next()){
        System.out.print("\nCatalogo: " + rs.getString(1) + "\nEsquema: " + rs.getString(2) +
         "\ntabla: "+ rs.getString(3) + "\ntipo: "+ rs.getString(4));
    }
    conexion.close();
}
catch (Exception e) {e.printStackTrace(); }
```

4. API METADATA CON JAVA

Método GetTables

- Devuelve un objeto ResultSet que proporciona información sobre las tablas y vistas de la base de datos. Su sintaxis es:

public abstract ResultSet getTables(String catalogo, String esquema, String patronDeTabla, String tipos[ ]) throws SQLException

- El significado de los parámetros es:
  - Primer parámetro: catálogo de la base de datos. Al poner null indicamos el catálogo actual.
  - Segundo parámetro: esquema de la base de datos. Al poner null indicamos esquema actual.
  - Tercer parámetro: es un patrón en el que se indica el nombre de las tablas que queremos que obtenga el método. Se puede utilizar el carácter guión bajo o porcentaje. Así, por ejemplo, "de%" obtendría todas las tablas que empiezan por "de".
  - Cuarto parámetro: un array de String en el que indicamos qué tipos de objetos queremos obtener. Por ejemplo: TABLE, VIEW, etc. Al poner null nos devolverá todos los tipos.

4. API METADATA CON JAVA

- Cada fila de ResultSet que devuelve **getTables()** tiene información sobre una tabla. La descripción de cada columna tiene las siguientes columnas:
  - TABLE_CAT → Columna1: el nombre del catálogo al que pertenece la tabla
  - TABLE_SCHEM → Columna2: el nombre del esquema al que pertenece la tabla
  - TABLE_NAME → Columna3: el nombre de la tabla o vista
  - TABLE_TYPE → Columna4: tipo TABLE o VIEW
  - REMARKS → Columna5: comentarios
  - Y más: TYPE_CAT, TYPE_SCHEM, TYPE_NAME, SELF_REFERENCING_COL_NAME, REF_GENERATION

4. API METADATA CON JAVA

Método GetColumns

- Devuelve un objeto ResultSet con información sobre las columnas de una tabla o tablas. La descripción de cada columna tiene las siguientes columnas: TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME, DATA_TYPE, TYPE_NAME, COLUMN_SIZE, BUFFER_LENGHT, DECIMAL_DIGITS, NUM_PREC_RADIX, NULLABLE, REMARKS, COLUMN_DEF, SQL_DATA_TYPE, SQL_DATETIME_SUB, CHAR_OCTET_LENGTH, ORDINAL_POSITION, IS_NULLABLE, IS_AUTOINCREMENT, etc.

```
public abstract ResultSet getColumns( String catalogo, String Esquema, String
patronNombreDeTabla, String patronNombreDeColumna) throws SQLException
```

- Para patrón de tabla y de columna se puede utilizar el porcentaje o el guión bajo.

4. API METADATA CON JAVA

Más métodos de DatabaseMetaData

- **getPrimaryKeys()** → devuelve la lista de columnas que forman la clave primaria de la tabla especificada
- **getExportedKeys()** → devuelve la lista de todas las claves ajenas que utilizan la clave primaria de la tabla especificada.
- **getImportedKeys()** → devuelve la lista de claves ajenas existentes en la tabla indicada
- **getProcedures()** → devuelve la lista de procedimientos y funciones almacenadas.

PRACTICA 4

Genera un programa en Java que tenga las siguientes opciones:

- **1. Visualiza datos de la Metadata: Nombre driver, URL, username, nombre base de datos, versión base de datos**
- **2. Muestra las bases de datos de mysql: getCatalogs()**
- **3. Ejecuta show databases: executeQuery()**
- **4. Muestra las tablas de una base de datos: getTables()**
- **5. Muestra las columnas de una tabla de una base de datos: getColumns(), obtener el nombre, el tipo y el tamaño de las columnas**
- **6. Muestra las claves primarias y foreign keys de una tabla: getPrimaryKeys() y getExportedKeys()**
- **7. Salir**

PRACTICA 4

```
while (true) {
    System.out.println("1. Visualiza datos de la Metadata");
    System.out.println("2. Muestra las bases de datos de mysql");
    System.out.println("3. Ejecuta show databases; ");
    System.out.println("4. Muestra las tablas de una base de datos");
    System.out.println("5. Muestra las columnas de una tabla de una base de datos");
    System.out.println("6. Muestra las claves primarias y foregin keys de una tabla");
    System.out.println("7. Salir");
    System.out.print(
    int opcion = rea
```

```
public static void show_primary_keys(DatabaseMetaData dbmd) throws SQLException {
    System.out.print("Introduce base de datos: ");
    String based=reader.next();
    System.out.print("Introduce tabla de la base de datos: ");
    String tabla=reader.next();
    ResultSet rs = dbmd.getPrimaryKeys(based, null, tabla);
    while (rs.next()){
        System.out.println("Table name: "+rs.getString("TABLE_NAME"));
        System.out.println("Column name: "+rs.getString("COLUMN_NAME"));
        System.out.println("Primary key sequence: "+rs.getString("KEY_SEQ"));
        System.out.println("Primary key name: "+rs.getString("PK_NAME"));
        System.out.println(" ");
    }
}
```