

## BASE DE DATOS CON JAVA

### Índice:

1. Introducción.
2. Interfaces utilizadas.
3. Conexión con el servidor de base de datos.
4. Seleccionar datos de tablas.
6. Insertar, eliminar y modificar filas.

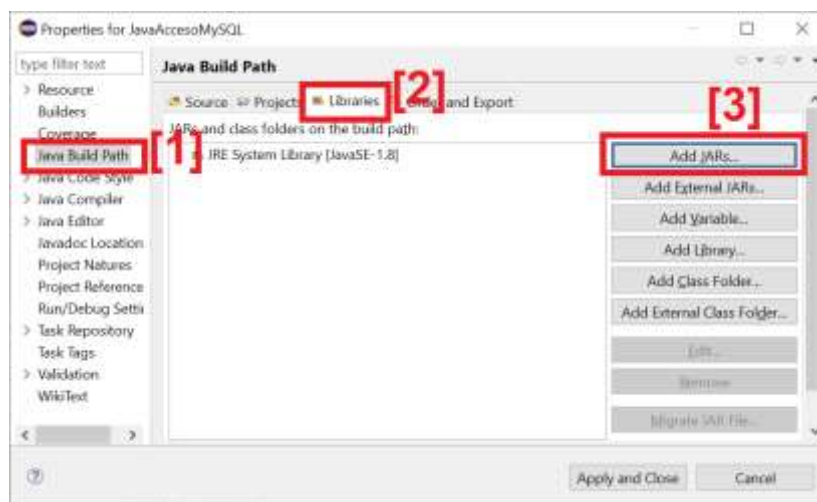
## 1. Introducción.

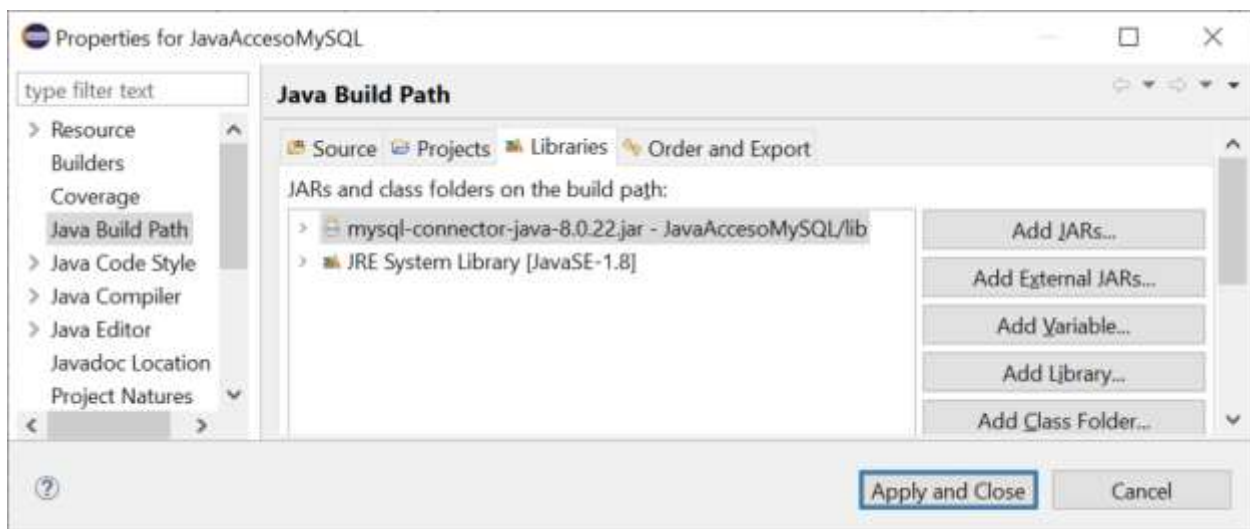
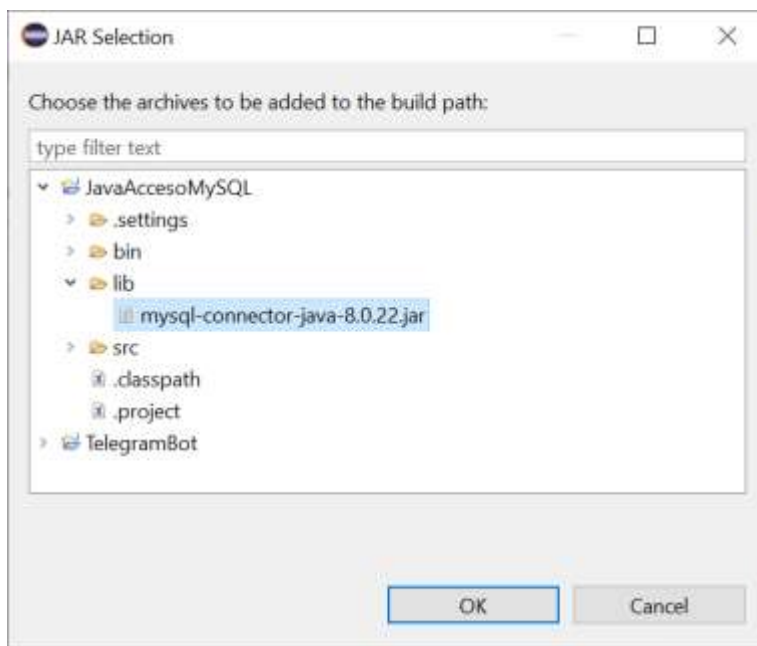
En esta unidad estudiaremos la forma de conectar nuestras aplicaciones Java con un servidor de base de datos.

Para llevar a cabo esta conexión necesitamos lo siguiente:

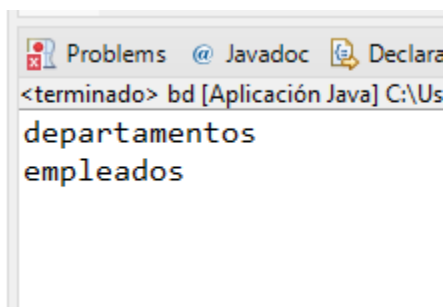
- Servidor de base de datos al que conectarnos, en nuestro caso MariaDb. Este servidor puedes encontrarlo en el paquete Xampp y descargarlo desde aquí: <https://www.apachefriends.org/es/download.html> o bien ir directamente a la web de MariaDB o MySQL.
- Una cuenta con permisos en ese servidor y una base de datos a la que conectar.
- Un driver JDBC (Java Database Connectivity), que tiene la función de comunicar nuestra aplicación con el servidor de base de datos. En nuestro caso necesitamos un driver que comunique con MariaDB/MySQL. Existen varias formas de incorporar este driver a nuestros proyectos: añadir la dependencia en el archivo POM.XML si se trata de un proyecto Maven, añadirlo como librería externa... Para añadirlo como librería externa, descarga desde este enlace: <https://dev.mysql.com/downloads/connector/j/>

Elige plataforma independiente y descarga el archivo, descomprime el archivo en una carpeta y copia solo el archivo .jar que es el que necesitas. Lo tienes que copiar en la carpeta lib de tu proyecto, sino existe la creas. Después en las propiedades del proyecto, pincha derecha ratón y propiedades, y sigue estos pasos:





Y ejecutar, en nuestro caso, ver las tablas de departamentos:



MI APP:

```
import java.sql.*;
public class App
{
    public static void main( String[] args )
    {
        Connection conexion=null;

        try {

            conexion = ConectarBd.getConexion();

            Statement st = conexion.createStatement();

            ResultSet rs = st.executeQuery("show tables");

            //avanzo al primer registro
            while(rs.next()) {
                System.out.println(rs.getString(1));
            }

            // rs.close();
        } catch (SQLException e) {
            System.out.println(e.getErrorCode() + " " +
e.getMessage());
        } finally {
            try {
                conexion.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

### MI ConectarBD

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConectarBd {
    private static Connection con = null;

    public static Connection getConexion() throws SQLException {
        if (con == null) {
            con =
DriverManager.getConnection("jdbc:mysql://localhost/departamentos",
                            "nieves", "nieves");
        }
        return con;
    }
}
```

## 2. Interfaces utilizadas

El paquete en que están estas interfaces es java.sql.

- Connection. Con ella se establece la conexión con el servidor. A través de esta conexión se realizan todas las demás operaciones.
- Statement. Ejecutar sentencias SQL
- PreparedStatement. Ejecutar sentencias SQL.
- ResultSet. Acceder al conjunto de resultados devueltos por una SELECT.

## 3. Conexión con el servidor de base de datos.

En este ejemplo conectaremos con el servidor MariaDB instalado en nuestro ordenador.

Para conectar usamos la clase DriverManager que utiliza una URL de JDBC. En esta URL se especifica el servidor al que vamos a conectar, la base de datos, usuario y contraseña. Su formato es: **jdbc:mysql://hostname/database**. Siendo hostname el equipo donde está el servidor (nombre, IP o localhost) y database, el nombre de la base de datos.

MariaDB/MySQL utilizan, por defecto, el puerto 3306. Si fuese otro, debería indicarse también en la cadena de conexión, con la forma: **hostname:puerto**.

Debemos incluir **import java.sql.\***, ya que es aquí donde se encuentran las clases e interfaces que usaremos.

```
System.out.print( "Conectando con MariaDB..." );

Properties propiedadesConexion = new Properties();

propiedadesConexion.put("user", "root");
propiedadesConexion.put("password", "");

try {
    Connection conexion =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/ClassicModels",
                                    propiedadesConexion);

    System.out.println(";;;Conectado!!!");

    Statement statement = conexion.createStatement();
    ResultSet rs = statement.executeQuery("Show tables");

    System.out.println("\nTablas de la base de datos ClassicModels\n");

    while(rs.next()) {
        System.out.println(rs.getString(1));
    }
} catch (SQLException e) {
    // TODO Auto-generated catch block
    System.out.println(e.getMessage());
}
```

El resultado debe ser este:

```
Conectando con MariaDB...;;;Conectado!!!

Tablas de la base de datos ClassicModels

categoriasproductos
clientes
detallespedidos
empleados
oficinas
pagos
pedidos
productos
```

#### 4. Seleccionar datos de tablas.

Pasos:

- Conectar con la base de datos y obtener un objeto Connection. En este ejemplo he utilizado la clase ConectarBd y su método getConnection(), que retorna un objeto sql.Connection. Ten en cuenta que el usuario y contraseña utilizados pueden no existir en tu servidor. Ejemplo:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConectarBd {
    private static Connection con = null;

    public static Connection getConnection() throws SQLException{
        if (con==null){
            con =
DriverManager.getConnection("jdbc:mysql://localhost/classicmodels",
"classicUser", "classicuser");
        }
        return con;
    }
}
```

- Crear un objeto Statement a partir de la conexión.
- Este objeto Statement tiene un método executeQuery al que pasamos la sentencia Select a ejecutar.
- Tras ejecutar la sentencia, un conjunto de registros es devuelto en un objeto ResultSet.
- Mediante un bucle while recorreremos el conjunto de registros. La pregunta para controlar el fin del bucle es **while (resultSet.next())**. Podemos interpretarla como "Mientras haya siguiente". Cada vez que se llama al método next() se avanza un registro.
- Para acceder a las columnas de la tabla del ResultSet se utilizan los métodos getString, getInt... de la forma getString("nombrecampo") o getString( numerocolumna). Las columnas se numeran a partir de 1.
- El bloque try-catch controla los posibles errores en la sentencia Select.

Ejemplo: Seleccionar datos de productos: código y nombre.

```
try {
    Connection conexion = ConectarBd.getConexion();
    System.out.println("!!!Conectado!!!");

    Statement statement = conexion.createStatement();

    ResultSet rs = statement.executeQuery("select * from productos");

    System.out.println("\nListado de productos\n");

    while(rs.next()) {
        System.out.println(rs.getString(1)+ "\t" + rs.getString(2));
    }

} catch (SQLException e) {
    // TODO Auto-generated catch block
    System.out.println(e.getMessage());
}
```

Resumiendo, hemos utilizado objetos Connection, Statement y ResultSet. El primero para establecer la conexión, el segundo para ejecutar la orden SQL y el tercero para acceder al conjunto de registros que ha devuelto la sentencia SELECT.

A continuación se muestran una serie de métodos de ResultSet.

- absolute(int numeroFila). Moverse a una posición absoluta del ResultSet. Moves the ResultSet to point at an absolute position. Esta posición es un número de fila pasado como parámetro.
- afterLast(). Moverse detrás del último registro.
- beforeFirst(). Moverse al principio del ResultSet, delante del primer registro.
- first(). Situar en el primer registro.
- last(). Moverse al último registro.
- next(). Devuelve true o false según el ResultSet tenga o no más registros. Mueve el puntero al siguiente registro del ResultSet.
- previous(). Moverse al registro anterior.
- relative(int posicion). Mueve el puntero del ResultSet a una posición relativa indicada como parámetro. Es relativa respecto a la posición actual. El valor



indicado puede ser positivo o negativo.`List<String> contenido = Files.readAllLines(rutaArchivo);`

- para que un `ResultSet` permita avanzar/retroceder en los registros debe ser `ResultSet.TYPE_SCROLL_INSENSITIVE`.

## 5. Insertar, actualizar y eliminar filas.

Para realizar estas operaciones sobre una base de datos es necesario:

- Obtener un objeto de tipo `Connection` que permita la comunicación con la base de datos.
- Un objeto de tipo `Statement` para ejecutar la orden SQL.
- Al método `executeUpdate` del objeto `Statement` le pasamos la sentencia SQL a ejecutar (`INSERT`, `UPDATE` o `DELETE`) y devuelve el número de filas afectadas por la operación.
- Deben controlarse los errores que puedan producirse.

Ejemplo de insert:

```
try {
    Connection conexion = ConectarBd.getConnection();

    System.out.println("¡¡¡Conectado!!!");

    Statement statement = conexion.createStatement();

    String sqlInsert = "Insert into categoriasproductos (categoria, descripcion) "
        + "values ('Dron', 'Vehículo areo...')";

    System.out.println("\nNuevo producto\n");

    int numero = statement.executeUpdate(sqlInsert);

    System.out.println("Filas insertadas: " + numero);
} catch (SQLException e) {
    System.out.println(e.getMessage());
}
```

Obviamente, si ejecutamos varias veces el mismo insert obtenemos una excepción SQL por clave duplicada.

```
Duplicate entry 'Dron' for key 'PRIMARY'
```

En moodle hay un ejemplo completo.