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.

5. Insertar, eliminar y modificar filas.

6. Patrón de diseño Singleton para conectar con una base de datos.

7. La clase PreparedStatement.

    7.1 Inserción, eliminación y actualización de filas.

    7.2 Consultar datos.

8. Metadatos.

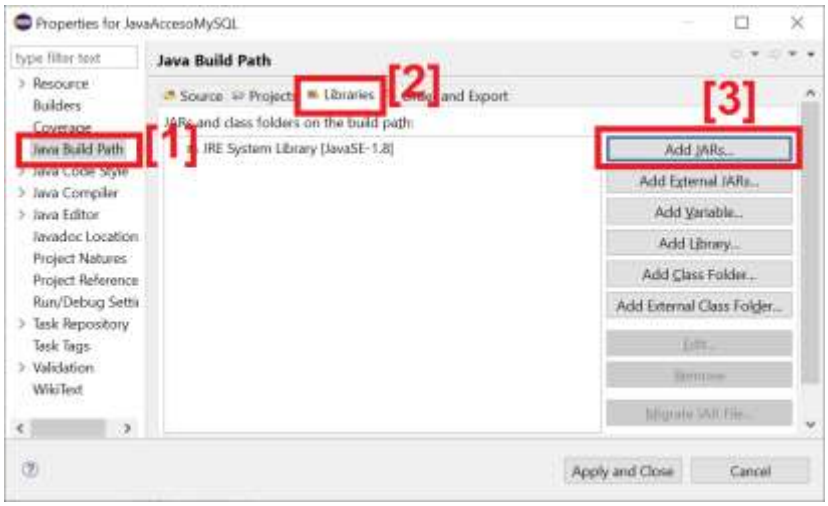
1. Introducción.

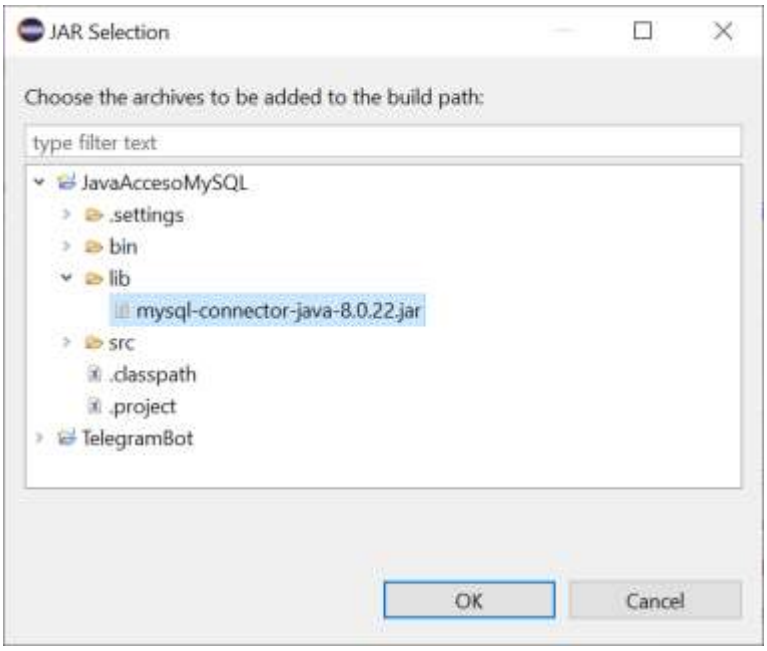
En esta unidad estudiaremos la forma de conectar nuestras aplicaciones Java con un servidor de base de datos.

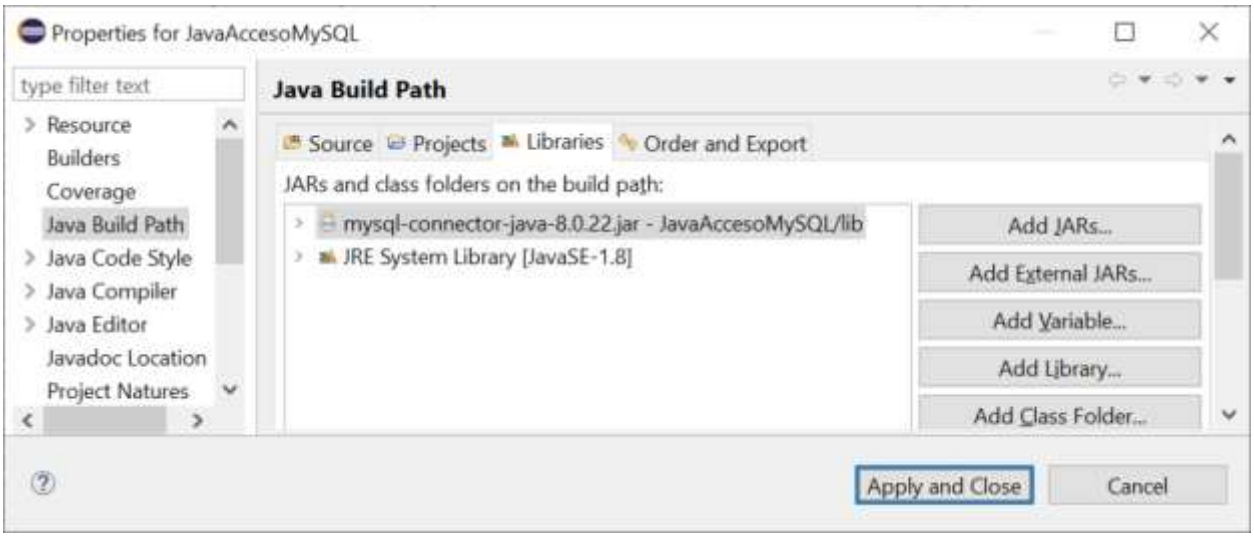
Para llevar a cabo esta conexión necesitamos lo siguiente:

- 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 Conectivity), 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/

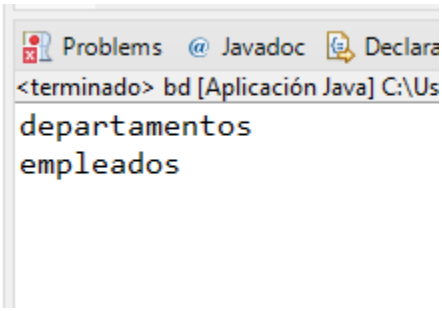
  Elige plataforma independiente y descarga el archivo, descomprime el archivo en una carpeta y copia solo el archivo .jar que es el que necesitas. Lo tienes que copiar en la carpeta lib de tu proyecto, sino existe la creas. Después en las propiedades del proyecto, pincha derecha ratón y propiedades, y 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 getConexion(), 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 getConexion() 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 recorremos 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(). Situarse 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.getConexion();

    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.

6. Patrón de diseño Singleton para conectar con una base de datos.

Antes de nada, empezaremos definiendo que es un patrón de diseño. En el mundo de la programación, existen multitud de problemas que han sido estudiados por los programadores a lo largo de los años. Un patrón de diseño es una solución dada a un determinado problema de diseño. Esa solución está probada, es efectiva y puede ser reutilizada en diferentes situaciones.

Dicho esto, el caso que nos ocupa es el estudio del patrón de diseño Singleton. El objetivo de este patrón es que una clase sólo pueda tener una instancia.

Imagina que estamos desarrollando una aplicación que trabaja con bases de datos, y que en determinadas situaciones debemos conectar con la base de datos para llevar a cabo alguna tarea. Si creamos una conexión con el servidor cada vez que quisiéramos conectar con la base de datos esto podría ocasionar la sobrecarga del servidor. Para evitar este problema utilizamos el “Patrón Singleton”. Su función será la de devolver siempre el mismo objeto Connection y de esta forma evitar crear una conexión nueva cada vez que se solicite.

¿Cómo sería la clase que se ajuste a este patrón?

- En nuestro caso debería tener un atributo de la clase Connection, estático e inicializado a null
- Debe contener un método (getConexion o similar) que devuelva un objeto de clase Connection. Debe ser static, para que pueda ser utilizado desde la propia clase, sin necesidad de instanciar objetos, ya que esto es lo que tratamos de evitar.
  Este método comprueba si ya existe una conexión creada, en caso afirmativo la devuelve, y si no existe la crea.
- No debe tener constructor.
- Para acceder al objeto Connection, basta con hacer:
  Connection conexion = NombreClase.getConexion().

En definitiva, para la base de datos “classicModels” tendremos la clase siguiente:

```
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/ClassicModels",
                    "root", "");
        }
        return con;
    }
}
```

7. La clase PreparedStatement.

7.1. Inserción, eliminación y actualización de filas.

- Mediante esta interfaz el trabajo de insertar, eliminar y actualizar datos se agiliza un poco. Evitando las molestas comillas simples para encerrar literales de tipo texto.
- De la misma forma que antes, debemos partir de un objeto **Connection**.
- Necesitamos obtener también un objeto **PreparedStatement**, a partir de dicha conexión.
- Por supuesto, será imprescindible una cadena (String) donde guardar la sentencia a ejecutar. En esta sentencia, en la parte correspondiente a VALUES, entre paréntesis, en lugar de los valores a insertar, escribimos un símbolo de interrogación por cada columna.
- Más tarde, se asignan los valores a cada columna mediante los métodos set (setString, setDouble...) del objeto PreparedStatement.
- Por último, se ejecuta la sentencia con el método **execute() o executeUpdate()** del objeto PreparedStatement. El segundo devuelve el número de filas afectadas por la operación realizada.

```java
Connection con = null;
try {
    // Obtenemos la conexión
    con = ConectarBd.getConexion();
    //Creamos la sentencia SQL
    String sql = "Insert into categoriasproductos (categoria, "
            + "descripcion) values (?,?)";

    //Obtenemos el PreparedStatement
    PreparedStatement pS = con.prepareStatement(sql);
    //Asignamos valores
    pS.setString(1, "Subm");
    pS.setString(2, "Submarino militar 2º GM");
    int filasInsertadas = pS.executeUpdate();

    System.out.println("Filas insertadas: " + filasInsertadas);

    con.close();
} catch (SQLException ex) {
    System.out.println(ex.getMessage());
}
```

Tienes en Moodle un ejemplo completo con inserciones, eliminaciones y actualizaciones mediante PreparedStatement.

7.2. Consultar datos

- Partimos como siempre de un objeto Connection.
- A partir de él obtenemos un objeto PrepararedStatement, mediante conexion.prepareStatement(consulta). Donde consulta es una cadena que contiene la sentencia select a ejecutar. Esta sentencia también llevará un símbolo de interrogación por cada parámetro.
- Asignamos valores a los parámetros mediante los métodos set del objeto PreparedStatement.
- Ejecutar la sentencia mediante executeQuery().
- El conjunto de registros devueltos se obtiene en un objeto ResultSet.

Un ejemplo:

```
Connection con = null;
try {
    // Obtenemos la conexión
    con = ConectarBd.getConexion();
    //Creamos la sentencia SQL
    String sql = "Select * from clientes where ciudad = ?";
    //Obtenemos el PreparedStatement
    PreparedStatement pS = con.prepareStatement(sql);
    //Añadimos valores
    pS.setString(1, "Frankfurt");
    //Obtenemos las filas
    ResultSet rs = pS.executeQuery();
    //Recorremos el ResultSet
    while (rs.next()){
        //uso un printf, con formateo
        //puedo hacer refe. a las columnas con posición  o nombre
        System.out.printf("N° cliente: %d, Nombre: %s, Ciudad: %s \n",
                rs.getInt(1), rs.getString("nombreContacto"),
                rs.getString("ciudad"));
    }
    //cerrar
    rs.close();
    con.close();
```

8. Metadatos.

Los metadatos aportan información sobre los datos. Aplicando el concepto a las bases de datos, los metadatos de una base de datos aportan información sobre las tablas, columnas, índices, relaciones, etc. de esta base de datos.

En los apartados que siguen, partimos de un objeto conexión (Connection) con permisos de superusuario (root) o similar.

8.1. Conocer las bases de datos alojadas en el servidor.

- Obtener un objeto Connection.
- A partir de él obtener los metadatos mediante el método getMetaData(). Este método devuelve los metadatos en un objeto de la clase DatabaseMetaData.
- Algunos métodos para acceder a los metadatos.
    - getDatabaseProductName(). Nombre del servidor.
    - getDatabaseProductVersion(). Versión del servidor.

- getDriverName(). Nombre del driver.
- getDriverVersion(). Versión del driver.
- getCatalogs(). Devuelve un ResultSet con la colección de bases de datos del servidor o las bases de datos a las que tiene acceso la cuenta con la que se ejecuta la aplicación. Para obtener el nombre de la base de datos del ResultSet se utiliza el método getString("TABLE_CAT");

8.2. Conocer las tablas de una base de datos.

- Obtener un objeto Connection.
- Obtener los metadatos.
- Obtener la lista de tablas de la base de datos.
    - getTables(null, null, null, null). Devuelve un ResultSet con las tablas de la base de datos asociada a la conexión. Mediante getString(3) obtenemos el nombre de la tabla.
    - getTables("nombrebasedatos", null, null, null). En este caso obtenemos el listado de tablas de la base de datos indicada.
    - Para acceder al nombre de una tabla a partir de ese ResultSet se utiliza getString(3).

8.3. Conocer las columnas de una tabla.

- Obtener un objeto Connection.
- Obtener los metadatos.
- Obtener la lista de columnas de una tabla.
    - getColumns(null, null, "nombretabla", null). Devuelve un ResultSet con la lista de columnas de la tabla pasada como parámetro de la base de datos asociada a la conexión.
    - getColumns("nombrebasedatos", null, "nombretabla", null).
    - Para acceder al nombre de las columnas a partir de ese ResultSet se utiliza getString(4);

8.4. Ejemplo de cómo acceder a las columnas de una tabla/consulta sin conocer sus nombres.

- Partimos de un objeto ResultSet devuelto por el método executeQuery de un objeto PreparedStatement. La sentencia pasada debe ser una SELECT
- Recorrer la lista de columnas del objeto ResultSet para conocer sus nombres mediante: resultset.getMetadata().getColumnName(posicion). La primera posición es la 1.
- Recorrer el resultset para visualizar los datos.