MySQL y JDBC (I)

Bases de datos y Java

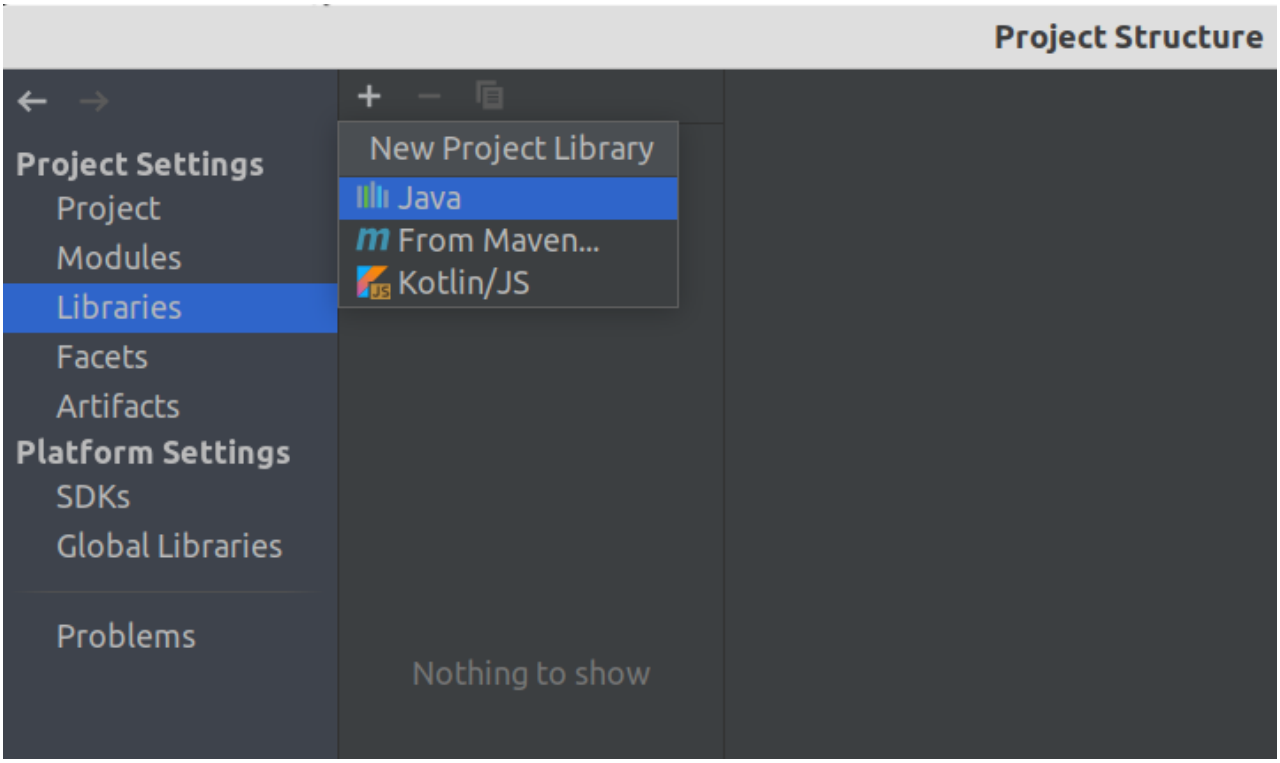
Para conectar a una base de datos en Java, necesitamos contar con JDBC (Java DataBase Connectivity). Esta librería se puede descargar desde dev.mysql.com o bien desde El repositorio central de Maven.

La importación de la librería puede hacerse de varias maneras:

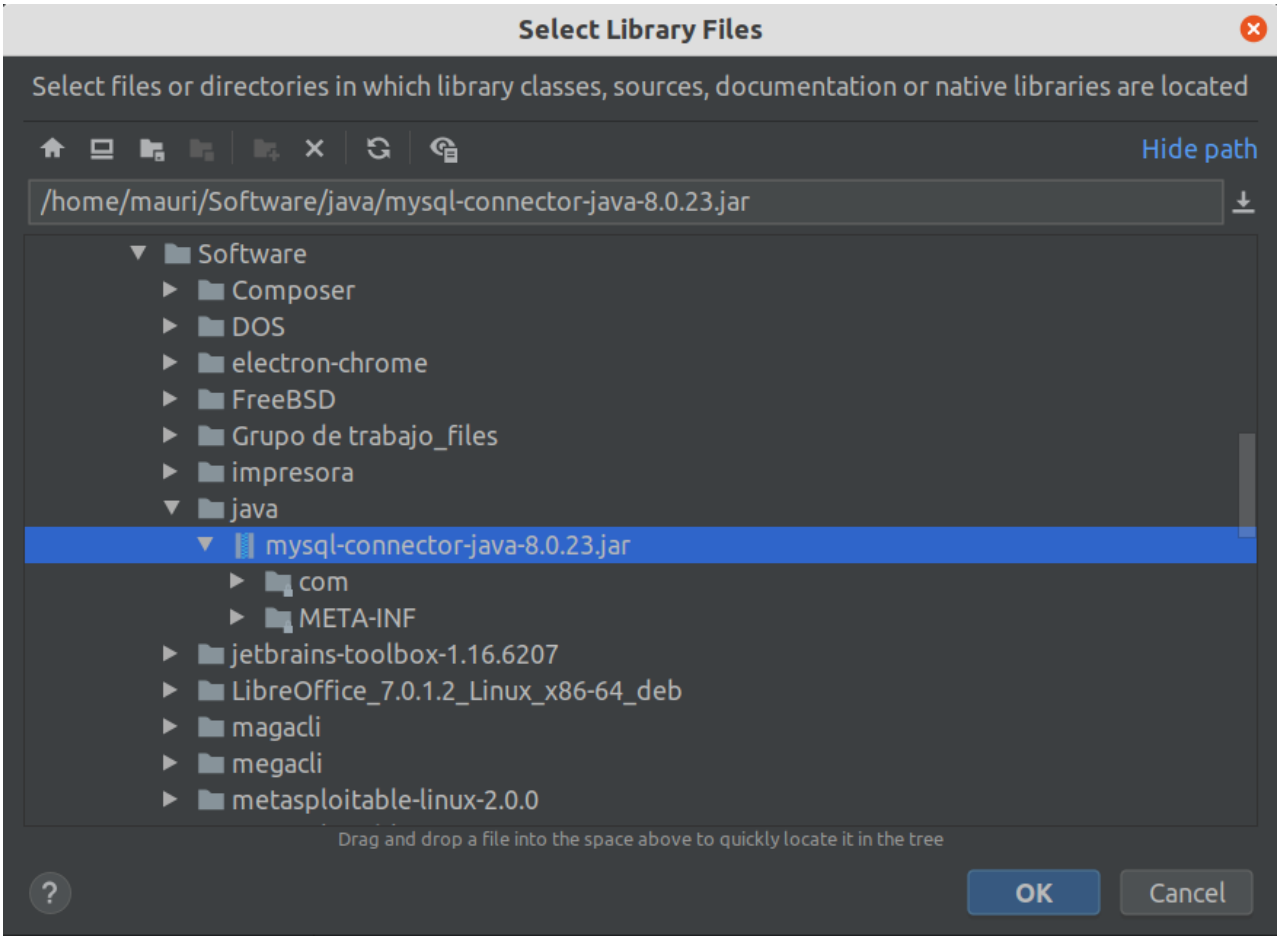
1. Utilizando un gestor de dependencias como Maven o Gradle. El procedimiento es el mismo que el aplicado con OpenCSV.
2. Importando manualmente la dependencia. El procedimiento manual es válido, aunque no es viable cuando existen muchas depedencias.

Instalar manualmente JDBC

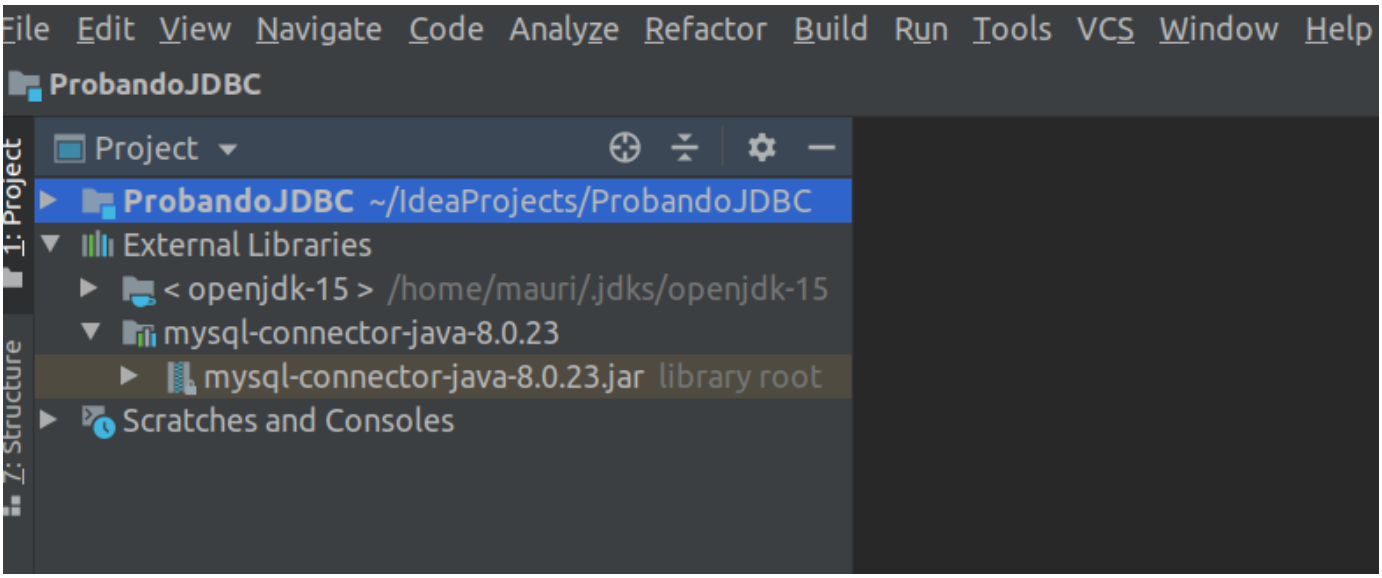
1. Ir hasta https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.23
2. Descargar el archivo *jar*
3. Importar la librería desde el proyecto:
    1. Abrir *File\Project Structure*
    2. Elegir la sección *Libraries* y hacer clic sobre "+"



3. Seleccionar el archivo



Después, podremos que la librería está importada en el proyecto:



A partir de este momento tenemos el código necesario para conectarnos a una base de datos *MySQL*

Dar permiso a un usuario

Dependiendo del SGBD que estemos usando, variará la sintaxis del DDL. Si estamos usando MySQL 5.x:

```
> GRANT ALL PRIVILEGES ON NombreBD.* TO 'usuario'@'localhost' IDENTIFIED BY
'P9$$w0rd';
```

Con esto creamos usuario y damos privilegios.

Si estamos usando MySQL 8.x:

```
> CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'P9$$w0rd';

> GRANT ALL ON NombreBD.* TO 'usuario'@'localhost';
```

Servidor remoto

En el ejemplo anterior, la base de datos está en *localhost*. Pero no siempre será así. El servidor puede estar en un servidor remoto. Para poder conectarnos en remoto, el usuario creado para la base de datos debe utilizar como host la IP de origen, o bien un comodín (%):

```
> CREATE USER 'usuario'@'%' IDENTIFIED BY 'P9$$w0rd';

> GRANT ALL ON NombreBD.* TO 'usuario'@'%';
```

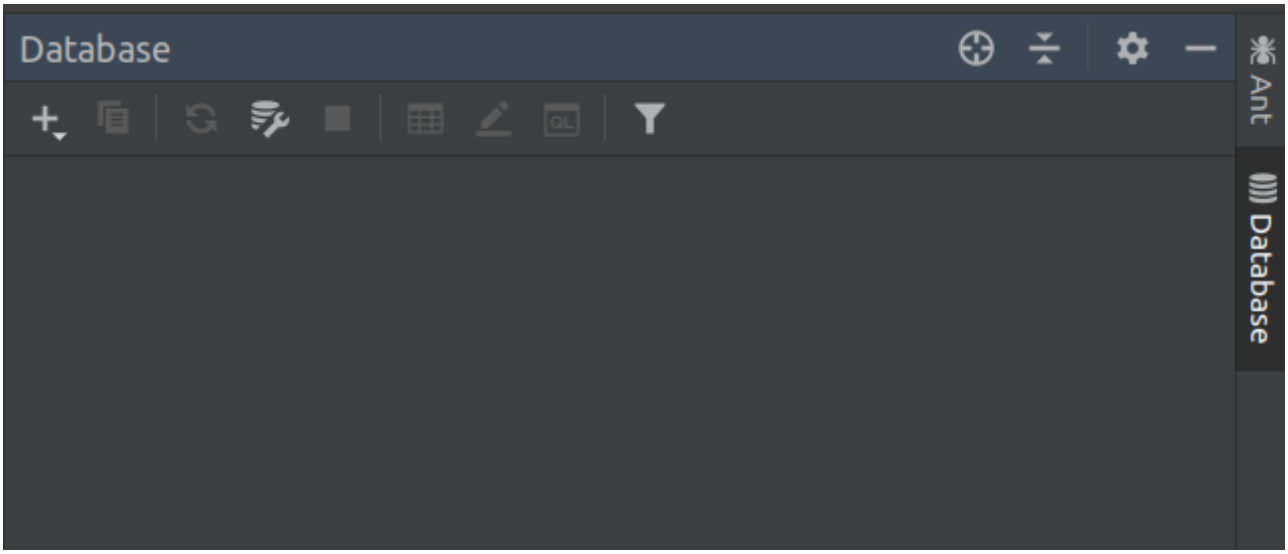
Por defecto *MySQL* está configurado para aceptar conexiones locales. Si queremos realizar conexiones remotas, tenemos que modificar el archivo de configuración de MySQL (*/etc/mysql/mysql.conf.d/mysqld.cnf*). Los parámetros a tener en cuenta son:

port: por defecto es 3306, pero se puede modificar.
bind-address: para permitir conexiones desde direcciones externas, podemos especificar como dirección *0.0.0.0*.

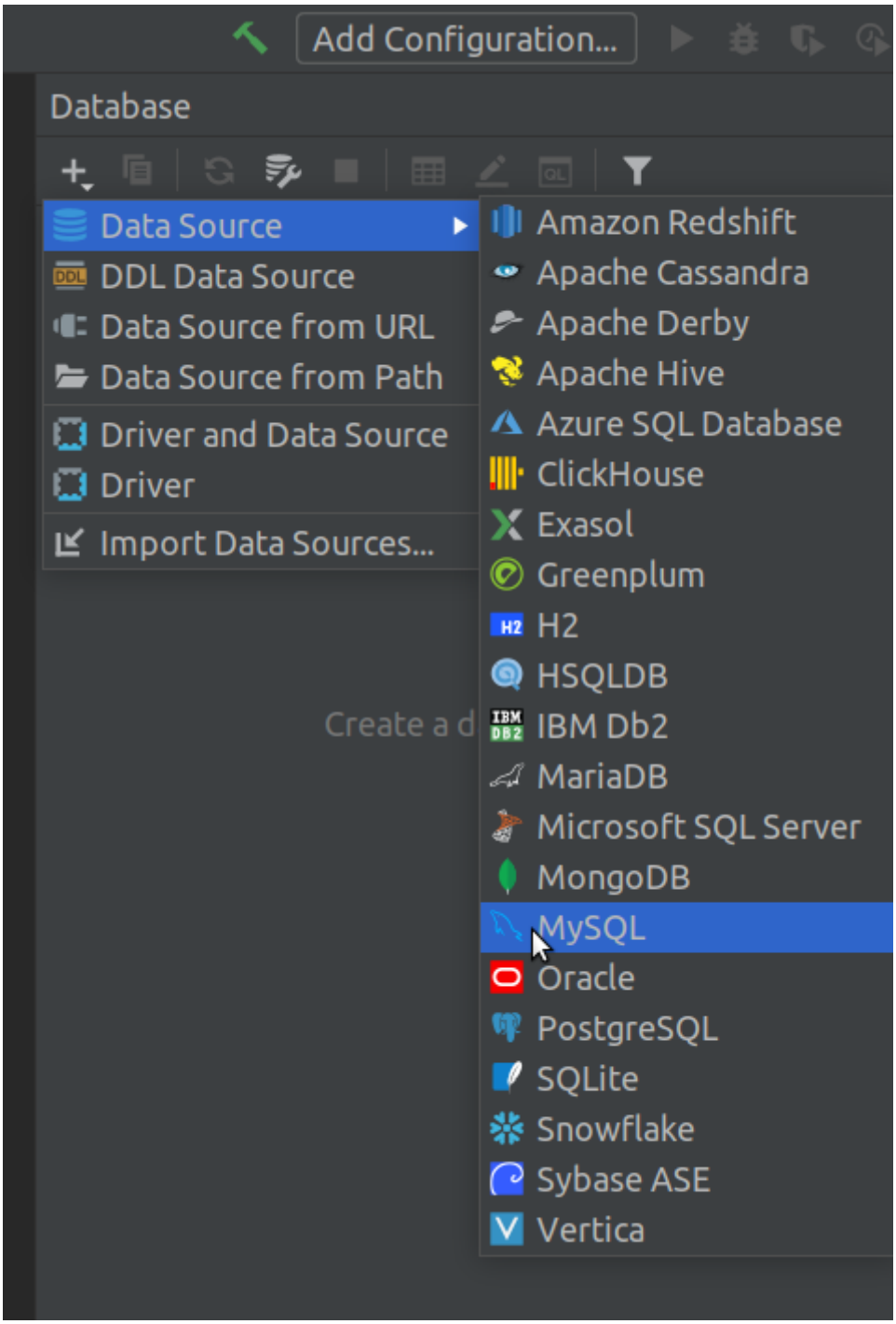
> Nota: esta configuración no es segura, y debe combinarse con medidas de seguridad adicionales en un servidor en producción.

El plugin de IntelliJ

Desde IntelliJ se puede utilizar el plugin para bases de datos. Para utilizar el plugin, debemos desplegar el panel derecho *Database*.

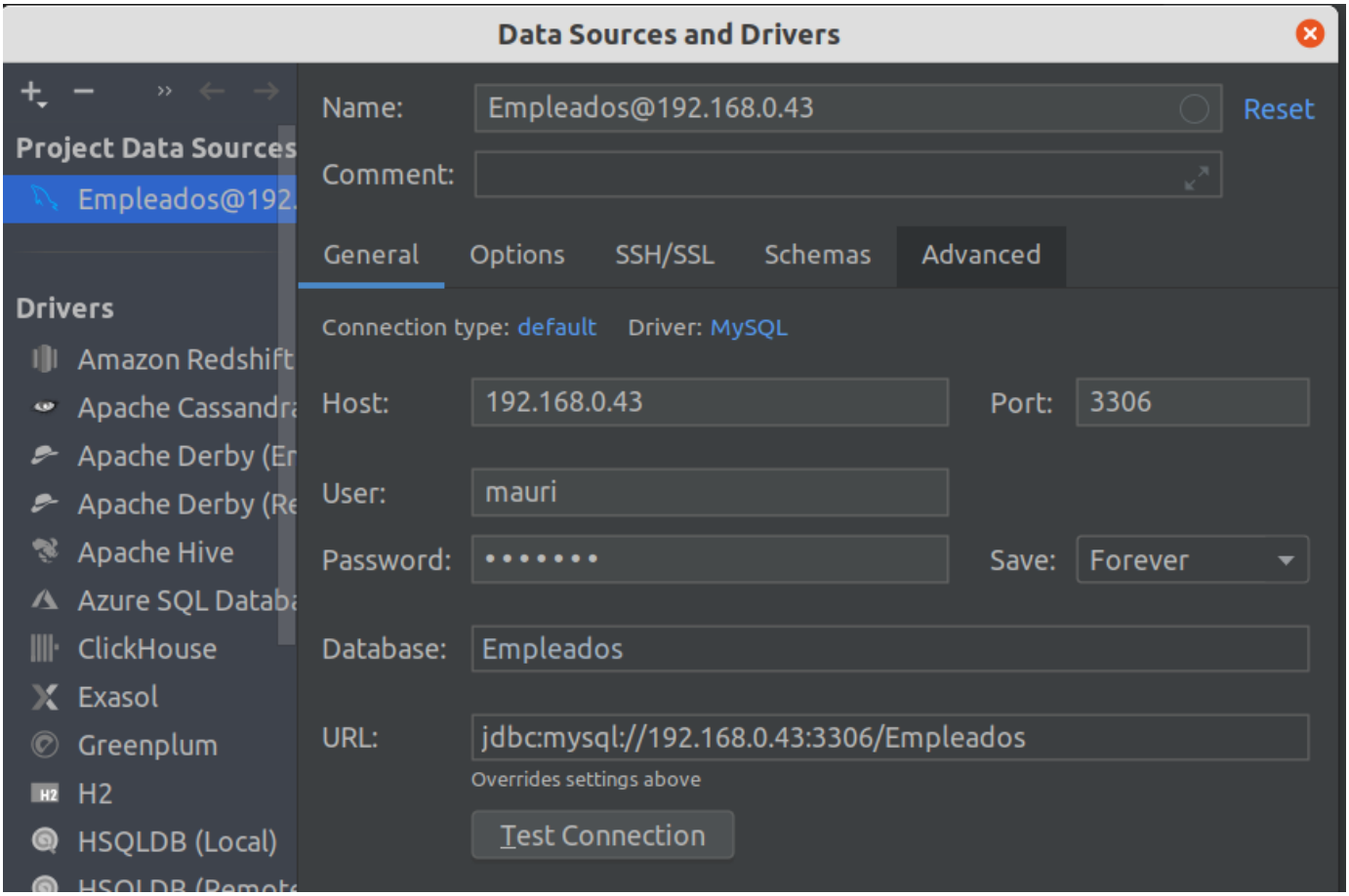


A continuación debemos añadir la fuente de datos deseada. En nuestro caso, *MySQL*, aunque ofrece soporte para diferentes SGBD (para poder usarlos habrá que importar el driver correspondiente).

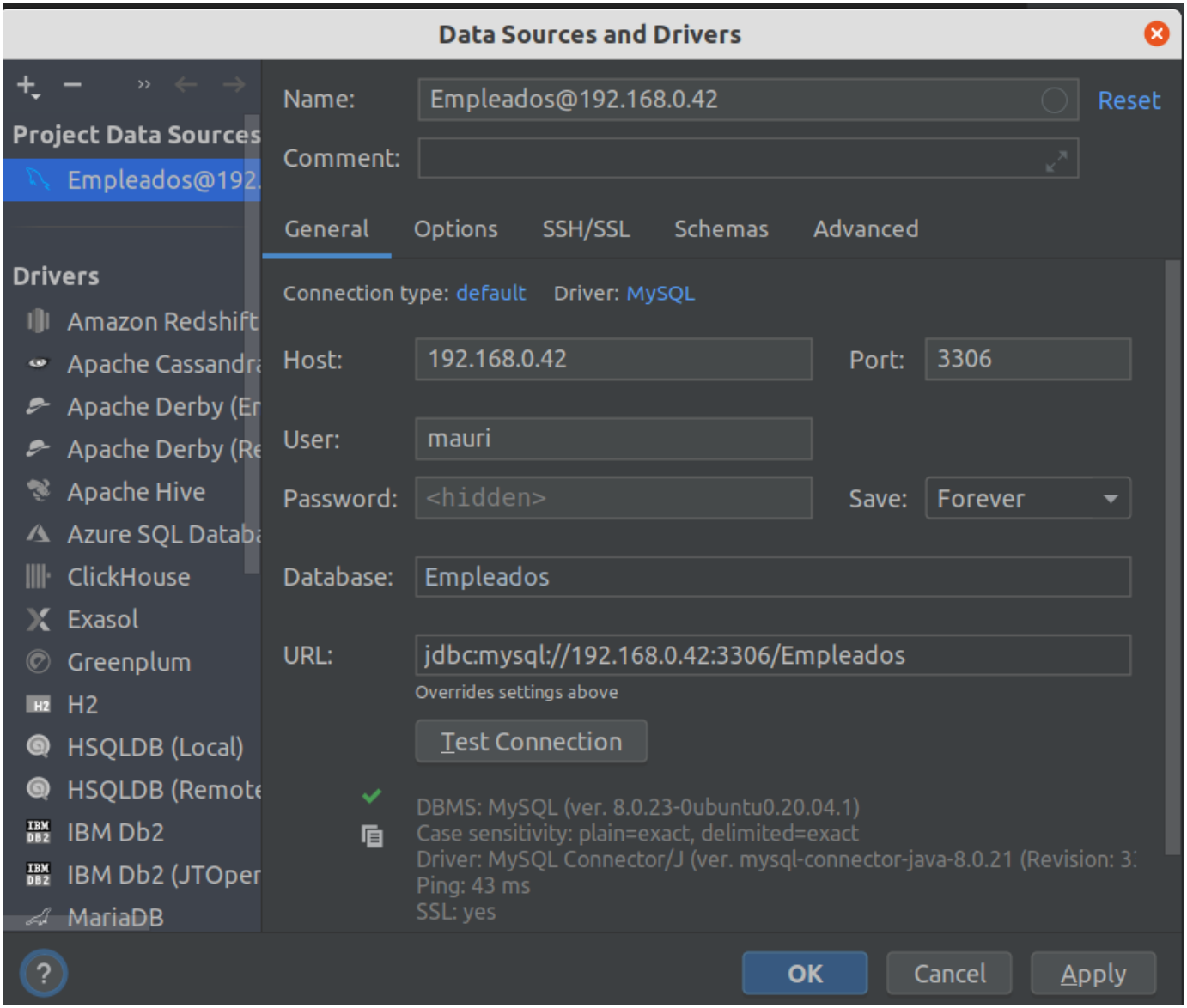


Finalmente, debemos definir los parámetros necesarios:

- *Host*: ip del servidor
- *User*: nombre el usuario con permisos sobre la base de datos.
- *Password*: contraseña asignada al usuario.
- *Database*: nombre de la base de datos.



Con los parámetros definidos, podemos hacer clic en el botón *Test Connection*. Si todo es correcto, podremos ver la confirmación de la conexión.



Base de datos local/remota

A continuación se muestra un ejemplo de uso, para una base de datos llamada pizzas. El SGBD se encuentra en la máquina local (localhost) y las credenciales de usuario son pizzauser/Perro20.

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

public class DBConnection {
    static String db = "pizzas";
    static String login = "pizzasuser";
    static String password = "Perro20";
    static String url = "jdbc:mysql://localhost:3306/" + db + "?
useSSL=false";
    Connection conn = null;
    public DBConnection(){
```

```
        try{
            conn = DriverManager.getConnection(url,login,password);

            if (conn!=null){
                System.out.println("Conexión ok: " + conn);

                ResultSet rs = conn.prepareStatement("SELECT nombre FROM
pizzas").executeQuery();
                while (rs.next()){
                    System.out.println(rs.getString("nombre"));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}
```

En caso de estar accediendo a una base de datos remota, el parámetro *url* debe ser modificado: *jdbc:mysql://ip_host:3306/nombreBD?useSSL=false*

Consultas

El siguiente ejemplo muestra cómo hacer una consulta:

```
    String query = "SELECT ...";
    ResultSet result = conn.prepareStatement(query).executeQuery();
    while (result.next()){
        result.getString("Campo1"); // devuelve una cadena de texto
correspondiente al campo Campo1
        result.getInt("Campo2"); // devuelve una cadena de texto
correspondiente al campo Campo2
    }
```

Borrados, inserciones y actualizaciones

Para insertar, utulizamos el método *executeUpdate* sobre un objeto de la clase *PreparedStatement*. En el siguiente ejemplo se hace directamente en una sola línea. El método *executeUpdate* devuelve el número de tuplas modificadas.

```
    int totalRows= conn.prepareStatement(updateQuery).executeUpdate(); //
devuelve el número de tuplas modificadas
```

En ocasiones necesitamos recuperar las claves de los elementos modificados. Para ello, debemos utilizar el método *executeUpdate* sobre un objeto de clase *PreparedStatement*, pero añadiendo como parámetro la constante *Stement.RETURN_GENERATED_KEYS*. Después de ejecutar la actualización, podemos llamar al método *getGeneratedKeys*, y obtendremos un objeto de tipo *ResultSet* conteniendo las claves.

```
    PreparedStatement statement = conn.prepareStatement(updateQuery,
Statement.RETURN_GENERATED_KEYS);
    int totalRows = statement.executeUpdate(); // devuelve el número de
tuplas modificadas
    ResultSet result = statement.getGeneratedKeys(); // devuelve una lista
con los id modificados.
    if (result.next()){
        System.out.println("La nueva clave es: " + result.getInt(1));
    }
```

Un primer ejemplo para practicar

Crear una base de datos donde registrar los empleados de una empresa. Se necesita registrar la información de cada empleado:

- DNI
- Nombre
- Apellidos
- Sueldo

La tabla de la base de datos incluirá un campo adicional como clave primaria ("id").

Mediante código Java, crea un método para completar cada una de las siguientes tareas:

- **boolean insertar(Empleado empleado)**: inserta un empleado en la base de datos y devuelve true si se consiguió insertar.
- **Empleado buscar(String dni)**: devuelve un empleado a partir de su DNI, buscándolo en la base de datos, o null si no lo encuentra.
- **boolean borrar(Empleado empleado)**: borra un empleado a partir de su DNI devolviendo true, o false si no es posible.
- **boolean actualizar(Empleado, double sueldo)**: modifica el sueldo de un empleado y devuelve true, o false si no es posible.