1. PostgreSQL

1.a (2p) Create a PostgreSQL instance

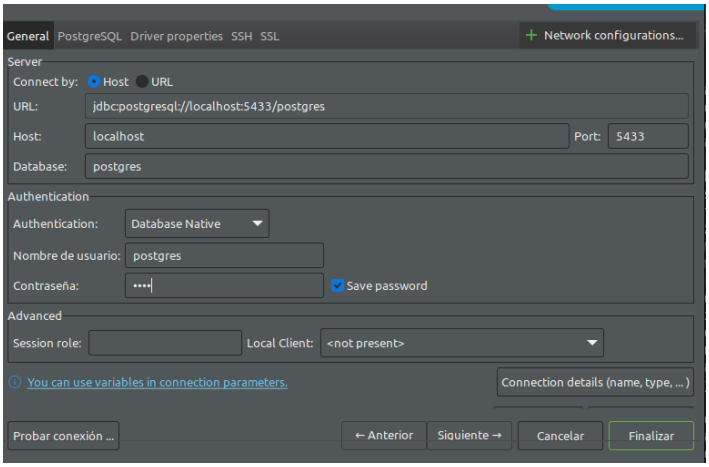
In this exercise you are invited to create and run a new server on your machine, running PostgreSQL. Is highly recommended to use Docker.

1. (0,5p) Write the sentence to create it.

```
sergio  sergio  ~  docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
af107e978371: Pull complete
4dab593eebe3: Pull complete
4998fa695fba: Pull complete
68722367c502: Pull complete
f94fde538ad8: Pull complete
083cda9930f9: Pull complete
d17e28f1e487: Pull complete
60abce37aea7: Pull complete
dc71bc844158: Pull complete
8af67c1d8689: Pull complete
a3a37d60b464: Pull complete
a28cd92dbadf: Pull complete
ebba832273a7: Pull complete
ca09208e18c7: Pull complete
Digest: sha256:b09f2562ab14fcae750cfc5ae457cd97e90c37679f520bc4a84180913de90261
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
```

```
sergio  sergio  ~  docker run --name examPostgres2 -e POSTGRES_PASSWORD=root -p 5433:5432 -d postgres
dbe6ef6dcfaf072cb1ade60e73480ffa090ac52d1ecc97f462a6933109d58fc4
```

2. (0,5p) Run a client, such DBeaver to create a sample database.



Here is the connection to the port 5433

3. (1p) Create a java project and connect to such database.

```java
package connection;

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

public class PostgreSQLJDBCConnection {

    // Connection details
    private static final String JDBC_URL = "jdbc:postgresql://localhost:5433/postgres";
    private static final String JDBC_USER = "postgres";
    private static final String JDBC_PASSWORD = "root";

    public static Connection getConnection() {
        Connection connection = null;
        try {
            // Load PostgreSQL JDBC driver
            Class.forName("org.postgresql.Driver");

            // Establish the connection
            connection = DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);

            if (connection != null) {
                System.out.println("Successfully connected to the database.");
            } else {
                System.out.println("Failed to connect to the database.");
            }
        } catch (ClassNotFoundException e) {
            System.out.println("Error: JDBC driver class not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            System.out.println("Error: Connection failed.");
            e.printStackTrace();
        }
        return connection;
    }
```

```java
package app;

import java.sql.Connection;

import connection.PostgreSQLJDBCConnection;

public class Main {
    Run|Debug
    public static void main(String[] args) {

        Connection conn = PostgreSQLJDBCConnection.getConnection();

        if (conn != null) {
            PostgreSQLJDBCConnection.closeConnection(conn);
        }

    }

}
```

1.b (2p) Implementing an OR Database

We are dealing with a classic inheritance exercise that we are going to model with PostgreSQL. From an **Employee** we will store the ***name*** and ***address*** of it (*type of street, street and number*) and a *set of telephones*. The valid street types will be *Street*, *Avenue* or *Part*. Employees can be:

- **Sellers**, in which case we will store your ***commission on sales*** and their ***role*** (telephone, distributor or door)
- **Commercial**, in which case we will store its ***remuneration*** (extra salary part) and ***area of action*** (Levante, South, Central or North)

In addition, for each Employee it will be indicated who is his ***superior***, through a ***reference to another Employee***.

*Use all the concepts studied on the subject.

2. ObjectDB

2.a (2p) Prepare your environment

Create *2 Hello ObjectDB projects* to use ObjectDB with it: one with *gradle* and another with *maven*, and run this test program:

```
EntityManagerFactory emf;
EntityManager em;
emf = Persistence.createEntityManagerFactory("testBD.odb");

// The extension is not mandatory, but is a good option

try {
      em = emf.createEntityManager();
      System.out.println("DB was created");
      em.close();
} catch (PersistenceException ex) {
      System.out.println(ex.getMessage());
}
```

2.b (4p)

We are going to develop a complete case, from the design of the database to its implementation and use. It is the management of a small company, which we will describe:

We have some **Employees**, of whom we need to know their **name**, **date of hire** and if they have had any **bad performance** lately. The employee **ID** must be automatically assigned by the system.

The data of the bad action should not be stored in the DB, we will only use it at execution time. Each employee has assigned an **Address** (street, number and block).

- The **Departments** have a **name** and a **location**, apart from their **identifier** (also assigned by the system). The departments will have several employees. One Employee can only belong to one department.
- As on the **Projects** (of which we only keep their **id** and **description**) we have the employee who is the project manager and of the employees who are participating in the project. An employee can participate in several projects at the same time but only be the head of one.

You have to:

1. (1p) Create the classes to store these requirements.
2. (1,5p) Create a program that asks the user for some data
3. (1,5p) Finally, resolve the following queries:
   a) Projects without department.
   b) Employees without departments.