# Introduction to Databases

## Use of AI

The use of artificial intelligence tools **is not allowed in this activity**. The course plan and the UOC's academic integrity and plagiarism have information on what is considered misconduct in assessment activities and the consequences this may have.

## PR3: Building river bridges: accessing a relational database from application programs

In this third practice, we will use the same database as in the previous practices. Specifically, we will use the schema from practice 2 with some modifications. We will access this database from Java using JDBC.

We provide you with an initial code that you must complete to solve the proposed exercises. The code compiles correctly but is incomplete, as it does not yet perform the intended functions. Your task is to complete the code in the sections we have prepared for you. Within the code, you will find comments prefixed with TODO ("to do"), a common convention in programming environments to indicate tasks that need to be done. You do not need to delete these comments, as they mark where you should add your code.

The complexity of the code to develop is not high, and you can use any Java development environment (Eclipse, NetBeans, IntelliJ IDEA, etc.) or a text editor (such as Notepad, UltraEdit, Notepad++, etc.) to complete it. The goal of this practice is not to learn a specific programming environment, so we recommend using the one you are familiar with to avoid unnecessary delays. Additionally, as indicated in the installation manual, it is recommended to use version 21 of the Java JDK.

Regarding the organization of the practice code, all Java code is located in the */src/main/java* directory, while additional files are located in the */src/main/resources* directory.

Within the supplied Java code, there is a package *edu.uoc.practica.bd.util* that contains several utility classes for handling list presentation, file reading, and array processing. Among these classes, only the *DBAccessor* class needs to be modified, as it is responsible for obtaining

connections to the database. The parameters used to establish these connections (server, port, user, etc.) are located in the */src/main/resources/db.properties* file.

In all cases, you must consider the best type of JDBC statement to use (*Statement*, *PreparedStatement*, *CallableStatement*, etc.), as well as how to handle transactions (*commit* and *rollback*) and exceptions. Always ensure that all resources are properly closed.

For the correct execution of the third part of the practice, it is necessary to recreate the database by executing the scripts provided in the */sql* folder:

1. *create_db.sql*
2. *inserts_db.sql*

The database structure created by *create_db.sql* differs from the rest of the practices. Specifically, the *wine_id* attribute of the *WINE* table is changed to the type *SERIAL*. This change allows the primary key of the table to be generated automatically. You can find more information about the *SERIAL* numeric type at the following link:

PostgreSQL: Documentation: 16: 8.1. Numeric Types

**It is very important that, before each exercise, you delete the contents of the database and reinsert the records from the *inserts_db.sql* file to avoid interference between exercises.**

**Important note:** Given that different versions of PostgreSQL accept different syntax variants of their statements, only responses that strictly adhere to the SQL statements explained in the theoretical modules of the subject will be considered valid.

Similarly, the latest versions of the Java language provide improvements for resource management, reducing the need for programmers to manually handle various database operations. As with the SQL statements, the use of resources that do not correspond to the theoretical modules of the subject will not be evaluated.

# Exercise 1 (16% weight)

You are asked to create a program that lists the information from the view *BEST_SELLING_ZONES*, which you will need to build with the script *create_view.sql* found in the */sql* folder of the *zip* file attached with the statement.

The goal is to create a list of the five zones or designations of origin from which we have sold the most wines. The header of the list will be:

```
Zone         Capital      Climate         Region              Last selling Total
============ ============ ============== ================== ============ =====
```

Proper error handling must be implemented, and in case of errors, a generic error message *"ERROR: List not available"* should be returned.

Upon executing the *create_db.sql* and *inserts_db.sql* scripts provided with the statement, the database will be set up.

As part of this exercise, the implementation of the *DBAccessor* class you perform will also be evaluated.

## Assessment criteria

- We recommend that you compile and execute the file that you will submit prior to uploading the final solution, in case you have done a last-minute modification (reorganizing code, extra comments, etc.)
- We will not evaluate any file submitted if it generates compilation errors.
- Weight of the programming *DBAccessor* class would be 30% of this exercise. We'll value that:
    - Connection to database works.
    - Path is used correctly.
    - Parameters from the file *db.properties* are used correctly.
    - No execution errors when code is invoked.
- For the rest of this exercise (weight of 70%), we'll value that:
    - JDBC classes are correctly used.
    - The list is correct and that the code handles situations where the table or view does not exist.
    - Close all resources, as indicated in the theoretical modules of the subject (using *try*, *catch* and *finally* blocks).

    - The code submitted is correctly commented, allowing us to clearly understand how it works.
    - Grades for this exercise will be penalized if you leave “spies” along the code.
- To obtain maximum grades we will value the clarity and the efficiency of the code.
- Equally, we require that you provide examples showing how the code works in all scenarios requested in a separate document.

# Exercise 2 (17% weight)

**Read from a file and perform an *INSERT* or *UPDATE* operation.**

In the folder `src/main/resources`, you can find a file called `exercise2.data` to be used for updating information in the database. This file contains new data that we would like to insert or update into the database.

The file can be loaded into the system using the following instructions:

- The first line of the file is a comment (which our code already ignores) and contains the description of each element in the file, separated by commas.

- The following lines in the file represent the data for updating the *WINERY*, *ZONE*, and *WINE* tables. The intention is to load or update the system data regarding new wines, their wineries, and their designations of origin, which will become part of our distributor catalog. Specifically, you must create a program that, for each of these lines, does the following:

    1. Create a new winery if it does not exist. If it exists, update the phone number (*winery_phone*) and the sales representative's name (*sales_representative*).
    2. Create a new designation of origin if it does not exist. If it exists, there is no need to update the data, as it does not change.
    3. Create a new wine with the *stock* value set to 0 and display the new wine code in the standard output.

- You need to handle any possible execution errors, including exceptions that may be caused by constraint violations, such as:

    - Incorrect values provided for a given column.

    In these cases, you need to abort the file load and roll back any changes made up to that point.

## Assessment criteria

- We recommend that you compile and execute the file that you will submit prior to uploading the final solution, in case you have done a last-minute modification (reorganizing code, extra comments, etc.)
- We will not evaluate any file submitted if it generates compilation errors.
- We will value:
    - JDBC classes are correctly used.

    - Operations of INSERT and UPDATE perform what's expected in this exercise.
    - Rollback is correctly performed.
    - Close all resources, as indicated in the theoretical modules of the subject (using *try*, *catch* and *finally* blocks).
    - The code submitted is correctly commented, allowing us to clearly understand how it works.
    - Grades for this exercise will be penalized if you leave "spies" along the code.
- To obtain maximum grade we will value the clarity and the efficiency of the code.
- Equally, we require that you provide examples showing how the code works in all scenarios requested in a separate document.
- Every operation performed to the database must be informed through the standard output.