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.

PR1: Creating and working with a relational database

We would like to have a database to store part of the relevant information about a wine distribution company with a designation of origin (DOP), which we will describe in terms of a set of relations.

The relations we work with are the following (underlined primary keys, *cursive foreign keys*) and the attributes cannot have *NULL* value unless specified. In each relation, you also have textual information about the primary and foreign keys.

This is the set of relations that describe the database (new relations have been added compared to CAA1):

WINE (wine_id, wine_name, vintage, alcohol_content, category, color, *winery_id*, *zone_id*, stock, price)

This relation contains information about the wines distributed by the company. For each wine, its identifier (*wine_id*), which is the primary key, is stored, along with the name (*wine_name*), vintage (*vintage*), alcohol content (*alcohol_content*), category (*category*), color (*color*), the winery identifier (*winery_id*), the identifier for the wine's designation of origin zone (*zone_id*), the available stock (*stock*), and the price (*price*) sold to customers (stores, restaurants, etc.). Stock is measured in boxes, and we assume each box always contains (regardless of the wine) 6 units. The price refers to the price per box.

The winery identifier (*winery_id*) is a foreign key to *WINERY*, and *zone_id* is a foreign key to *ZONE*.

Wine color (*color*) can take the values *red*, *white*, or *rosé*. The wine category can only take the values *young*, *aging*, *reserve*, and *grand reserve*. Alcohol content (*alcohol_content*) and price (*price*) must be positive values, and stock must be an integer greater than or equal to zero.

WINERY (winery_id, winery_name, town, established_year, winery_phone, sales_representative)

This relation contains information about the wineries that produce the wines. For each winery, its identifier (*winery_id*), which is the primary key, is stored, along with the name (*winery_name*), the town where it is located (*town*), the year the winery was established (*established_year*), as well as the winery's phone number (*winery_phone*) and the name of the person the distributor company typically communicates with (*sales_representative*).

The winery name (*winery_name*) is an alternate key. Sometimes the establishment year of the winery is unknown, so the value can be null.

ZONE (zone_id, zone_name, capital_town, climate, region, surface)

This relation contains information about the wine's designation of origin zones. Specifically, the zone identifier (*zone_id*), which is the primary key, is stored, along with the zone name (*zone_name*), the capital of the zone (*capital_town*), the climate (*climate*), the region (*region*) where the zone is located and the zone's surface area in hectares (*surface*).

The zone name (*zone_name*) is an alternate key.

GRAPE_VARIETY (grape_id, grape_name)

This relation contains information about the grape varieties that can be used for wine production. Specifically, the grape identifier (*grape_id*) and the name of the grape variety (*grape_name*) are stored.

The grape name (*grape_name*) is an alternate key.

WINE_GRAPE (*grape_id*, *wine_id*)

This relation contains information about which grape varieties (*grape_id*) were used in the production of different wines (*wine_id*).

The attributes (*grape_id*, *wine_id*) form the primary key. Additionally, *grape_id* is a foreign key to *GRAPE_VARIETY* and *wine_id* is a foreign key to *WINE*.

There may be wines for which this information is not available.

CUSTOMER_ORDER (order_id, *customer_id*, order_date, order_status)

This relation contains information about orders placed by the distributor company's customers. Specifically, the order identifier (*order_id*), which is the primary key, is stored, along with the customer identifier (*customer_id*), the date of the order (*order_date*), and its status (*order_status*). The order date (*order_date*) cannot be later than the current date. The order status (*order_status*) can only take the values *pending*, *shipped*, and *delivered*.

The *customer_id* attribute is a foreign key to *CUSTOMER*.

ORDER_LINE (*order_id*, order_line_id, *wine_id*, quantity)

This relation contains information about the wines requested in each order, i.e., the different items (or lines) that each order includes. Specifically, the order identifier (*order_id*), the order line identifier (*order_line_id*), the identifier of the wine (*wine_id*) being purchased, and the quantity of boxes ordered (*quantity*), which must be an integer greater than zero, is stored.

The attributes (*order_id*, *order_line_id*) form the primary key.

The *order_id* attribute is a foreign key to *CUSTOMER_ORDER*, and the *wine_id* attribute is a foreign key to *WINE*.

CUSTOMER (customer_id, customer_VAT, customer_name, customer_address, customer_town, customer_phone, customer_email, customer_status)

This relation contains information about customers (which can be stores, restaurants, supermarkets, etc.). Specifically, the customer identifier (*customer_id*), which is the primary key, is stored, along with their tax identification number (*customer_VAT*), which is an alternate key, the customer's name (*customer_name*), their address (*customer_address*), the town where the customer is located (*customer_town*), their phone number (*customer_phone*), a contact email address (*customer_email*), and the customer's status (*customer_status*), which can take the values *active* or *inactive*.

CLARIFICATIONS

The `create_db.sql` file is provided with the SQL sentences necessary to create the database and the `inserts_db.sql` file with the data insertion sentences that must be executed to answer all questions of this practical once exercise 1 has been resolved.

Remember that to work on tables that are part of a specific schema, you must use the schema name as a prefix, or you must set the `search_path` variable. For your convenience, at the beginning of each session execute the following statements:

```
SET search_path TO "name_of_schema";
```

Also remember to remove from the solution that you plan to submit the statements that you have used to verify your exercises, like *`DROP`*, any test *`INSERT`*, etc. that could alter the output of expected results.

Important note: SQL language implemented in PostgreSQL may accept different syntax statements that could vary depending on which version you have installed, and these may not be SQL standard. Please avoid (unless otherwise specified) utilizing those types of statements and focus on the ones that are explained in the topics. This is especially relevant for topic 4 where SQL standard is discussed. If you use SQL standard statements, your code will work in any RDBMS.

Exercise 1 (10 % weight)

In the `create_db.sql` file, we provide all the SQL sentences needed to create *WINE, ZONE, GRAPE_VARIETY, WINE_GRAPE, CUSTOMER_ORDER*, *ORDER_LINE* and *CUSTOMER* tables. Therefore, after one last analysis, it is clear that the following enhancements must be implemented:

1. What are the necessary SQL statements to **create** the *WINERY* table according to the definition provided? (2%)
2. You also need to provide the **alter** SQL statements to allow the following modifications to the database:
   - 2.1. Table *WINE* needs the following modifications (2%):
     - a. Add the *prizes* attribute to store the number of prizes won by the wine. It can take *NULL* as a value.
     - b. The *alchohol_content* attribute must be between 10º and 11º for white wines (*white*), between 11º and 15º for rosé wines (*rosé*) and, between 13º and 18º for red wines (*red*).
     - c. The winery identifier attribute (*winery_id*) is a foreign key to *WINERY*.
   - 2.2. Table *CUSTOMER_ORDER* needs the following modifications (2%):
     - a. Add the *order_amount* attribute to store the total amount of the order. This value must be equal or greater than 0.
     - b. Add the *order_reference* attribute to store an order identifier for the client reference. This attribute must be a character string and must have a format according to this pattern: *XXX-NN-XXXX*, where *X* must be capital letters and *N* must be numbers.
   - 2.3. Table *ORDER_LINE* needs the following modifications (2%):
     - a. Add the *discount* attribute to store a discount percentage for each order line. This attribute can be *NULL* in case that there is no discount.
   - 2.4. Table *ZONE* needs the following modifications (2%):
     - a. We cannot have capital towns with repeated names.
     - b. The *surface* attribute is not necessary and should be removed.

In the file `inserts_db.sql` you will find all the SQL *INSERT* statements that you need to execute **once all the tables have been modified**. Please, remember that the number of columns that tables require is specified in the file `inserts_db.sql`, therefore we will NOT accept as valid solutions those submissions with different number of columns or different types of columns not included in this file.

Important note: do **NOT** use domains to solve this exercise.

Note: in the following link you shall find information about regular expressions.

PostgreSQL: Documentation: 16: 9.7. Pattern Matching

Assessment criteria

- *Each question has a specific weight according to the number of sections and their difficulty.*
- *Questions not answered will not penalize the mark in this exercise.*
- *SQL statements that are not executing (syntax error) will not be evaluated.*
- *Table creation statements proposed that generate errors when executing the file inserts_db.sql provided will not be evaluated (this means that the number of columns of the tables is given in the file inserts_db.sql).*
- *We will not evaluate proposals of solutions that utilize domains (CREATE DOMAIN).*

Exercise 2 (15 % weight)

1. Write a query that returns the 5 active customers who have placed the most orders. We would like to display the customer code, customer name, total number of orders, and the date of the most recent order. The result of the query should be sorted by the number of orders in descending order, and in case of a tie, by the customer name in alphabetical order.

2. Write a query that returns the wines that belong to designations of origin (DOP) that have cataloged at least 5 wines in the database. We would like to display the name of each wine, its category, its color, and the name of the designation of origin (DOP) where it is produced, as well as the total number of wines for each zone. The result of the query should be ordered by the number of wines in descending order, and in case of a tie, by the zone name and the wine name.

3. Write a view named *gr_not_mediterranean* that retrieves information about wines that are categorized as '*grand reserve*' and that do not contain any grape varieties used in wines from designations of origin with a Mediterranean climate ('*Mediterranean*'). We would like to display the wine name, its category, the names of the grape varieties used, and the name of the designation of origin. The result of the query should be ordered by the wine name and by the names of the grape varieties.

Assessment criteria

- *All questions have the same weight.*
- *Questions not answered will not penalize the mark in this exercise.*
- *SQL statements that are not executing (syntax error) will not be evaluated.*
- *We will positively value the use of SQL standard (apart from other elements indicated in the exercise).*
- *To obtain the maximum mark:*
  - *The proposed solution in each question shall include the results of the query (screenshot or similar).*
  - *The proposed solution strictly provides what the exercise asks (for example, it shall provide the specified columns in the result set, sorting criteria, etc.)*
  - *You shall display the updated rows in a precise and friendly manner (proper column names, rounded decimals, etc.)*
  - *The proposed solution shall not contain unnecessary operations, calculations or statements.*

Exercise 3 (9 % weight)

A calculation error has been discovered regarding discounts on pending wine orders (*order_status* equal to '*pending*') where no wine from the designation of origin (DOP) '*Ribera del Duero*' is included, and furthermore, no discount has been applied to any of the items. For this reason, the wine distributor has decided to apply a 10% discount to all lines in the affected orders.

Please propose **a single SQL UPDATE statement** to correct the rows that need to be updated (if more than one statement is used, the task will be considered incorrect). Additionally, once the update has been made, display the set of rows that have been updated.

Important note: when you test the update, take into consideration that you should always start with the same database, that is, you should always have the same initial data set. Otherwise, you may find discrepancies in your analysis.

Assessment criteria

- *SQL statements that are not executing (syntax error) will not be evaluated.*
- *To obtain the maximum mark:*
  - *The proposed solution shall strictly provide what the exercise asks (that is, update only the data that is required – no more, no less) and with a single UPDATE statement.*
  - *The proposed solution should be efficient – for example, unnecessary joins in the UPDATE would penalize.*
  - *You shall provide clear rationale on why you're providing such answer, and you shall display all rows updated.*
  - *You shall display the updated rows in a precise and friendly manner (proper column names, rounded decimals, etc.)*
  - *The proposed solution shall not utilize unnecessary operations, calculations or statements.*