24303 DATABASES – LABS

Challenge 2: Database management, querying and SQL scripting

Goals:
The lab will cover the following aspects:

1. **Import/export data** into DB (Data Manipulation Language – DML)
2. Implement **queries in SQL** (Data Query Language – DQL)
3. Implement **PL/SQL requirements**: Procedures, Triggers and Events

Methodology:
This challenge is divided into four sessions. The statement of the lab must be read in order, following the instructions given step by step.

Session 1:

- Create new tables and import data. Deliver this part in an SQL file: **challenge2-load-data.sql**

Session 2:

- Create and deliver a new SQL file called **challenge2-queries.sql** for the queries and write your own queries based on the statements.

Session 3:

- Continue working on the queries from the previous session (**challenge2-queries.sql**).
- Start working on the list of new *requirements* for the database that will consist of creating procedures, trigger and events structures. You should create an SQL file for each: (**challenge2-req01.sql**, **challenge2-req02.sql […] challenge2-reqNN.sql**). One for every new requirement.

Session 4:

- Continue working on the implementation of the new requirements (procedures, triggers and events as asked).
- Collect your .sql files and prepare your deliverables.

All the used scripts used to solve whole challenge 2 in a ZIP file. The ZIP file should be called with the NIA of each member of the group, Uxxxxxx_Uxxxxxx_Uxxxxxx.zip.
ONLY A MEMBER OF THE GROUP WILL BE RESPONSIBLE FOR RETURNING THE CHALLENGE.
All the scripts have to be created and also runned over your database instance.

It is important to follow each point of the submission instructions. Otherwise, the lab will not be properly qualified, that is, it will be evaluated with a 0. **Please, before your submission check the following requirements:**

- All your scripts work BEFORE your submission
- You have defined your tables and fields using the names specified in this document.
- The file scripts must include the UTF-8 codification.
- The files scripts must follow the names specified in this document.

Grouping:
The challenge must be developed in teams of the same **three students of challenge 1**.

LAB Database Server access:
You will be provided with a database server connection to deploy your database (if you are listed in a LAB Team Group).
Ensure that you are able to connect from home using UPF's VPN connection: https://www.upf.edu/intranet/intern-universitat/connexio-vpn

Software:
We encourage you to use the **DBeaver** client or **MySQL Workbench**. For **DBeaver**, you can get the **Enterprise Edition** licence for one year simply filling an easy form on their website using UPF email: https://dbeaver.com/academic-license/

Doubts:
For any doubts outside LABs hours regarding the challenge, please **contact always both** LAB teachers in your emails and we will come back to you asap:

- raimon.izard@upf.edu
- guillem.casacuberta@upf.edu

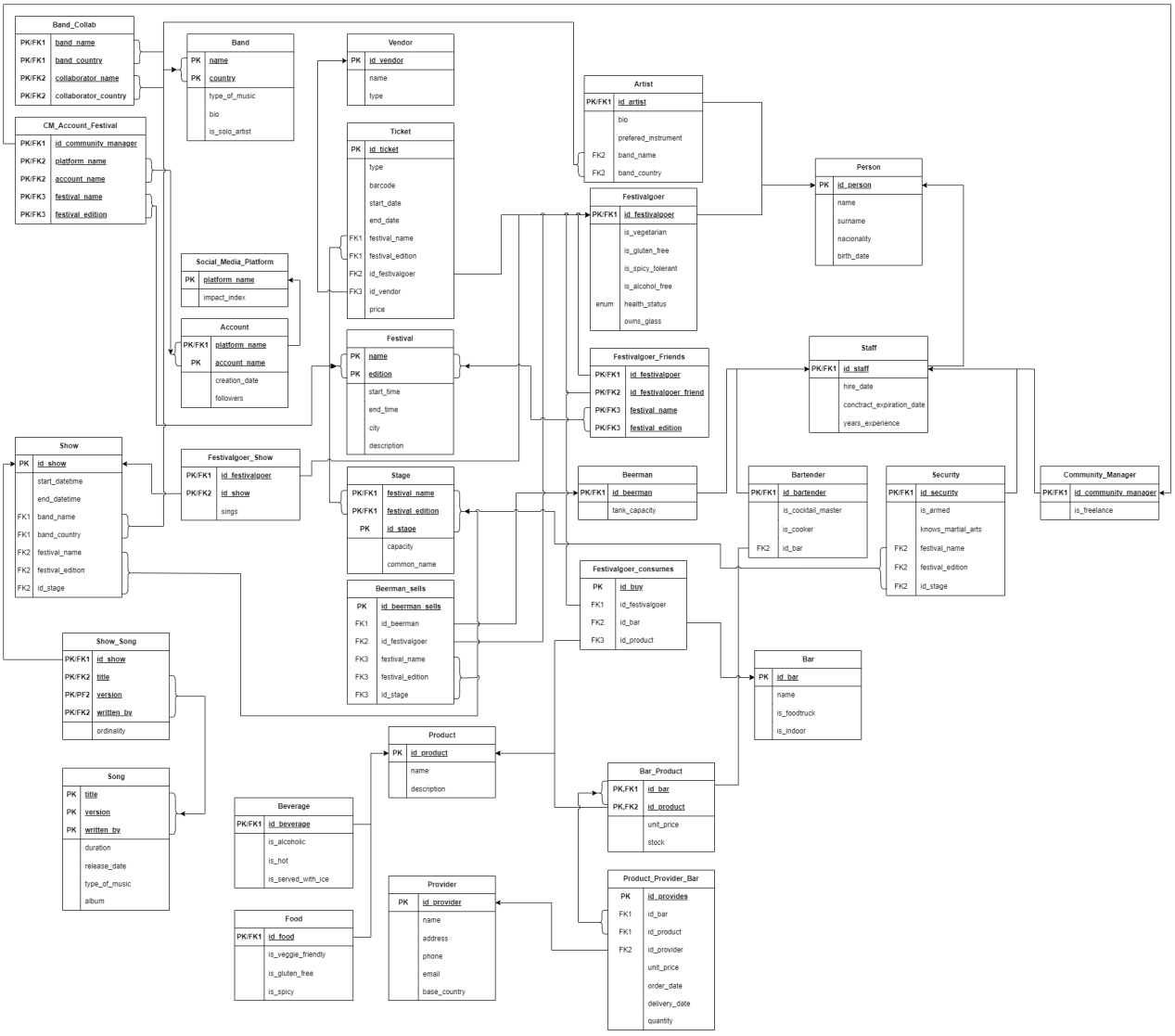
Note: The University has provided the required software (DBeaver and MySQL Workbench) to develop this lab, which is properly installed in the computers of the laboratory room classes. You will always have the option of working using the university computers. Furthermore, the professors have provided installation guides and manuals to facilitate the installation in your personal computers and for diverse OS systems (Windows, MacOS and other Unix-Like systems). The PCs from the room 54.004 have direct access to the database server instances without passing through VPN.

Deadline:
06/12/2024 23:59h CET

1. Introduction

In the remote database server, each group should find a database called PXXX_XX_challange2_music_festival (where PXXX is your lab group and XX is your team number).

This provided database represents a scenario of a music festival's company organiser. It is fully implemented and populated with data. Here you have the relational model of it:



Disclaimer: The contained information within the database is the result of random content generation. Any resemblance to reality is purely coincidental. In order to solve the queries, do not be guided by your previous musical knowledge or common sense. Expect the unexpected. There may be easter eggs.

2. New requirements for the database (50 points)

First of all, make sure that you already have the currency conversion tables created in your database as is asked in Challenge 2 - Part 1.2 (Insert New Data). They will be necessary for the following steps.

In this part you will be asked to develop new database objects: **Functions, Procedures**, **Triggers** and **Events**.
For all the following requirements, remember to use:

```
DROP db_object IF EXISTS db_object_name;

CREATE db_object IF NOT EXISTS db_object_name ...
```

All database objects should follow this pattern for their names: reqNN_OBJECT_NAME.

For instance, for the requirement 1, it should be: '*req01_currency_rounder*'

Deliver all needed PL/SQL scripts of every requirement on a new sql file using the following naming convention: **challenge2-reqNN.sql** *(being NN the number of the requirement)*.

If the requirement is a procedure or a function, add an example of use/execution in a comment at the end of the .sql of the delivered requirement.

Note: Submitted scripts have to be able to be launched several times and deliver always the same final result.

2.1. Requirement 1 (1 points)

Create a new function that, given a float number, if its value has more than 2 decimals returns it rounded with two decimals at max. If the number does not have more than 2 decimals return the number as is.

> **Hint:** Refer to official documentation to see how to create a **function**: https://dev.mysql.com/doc/refman/8.4/en/create-procedure.html

2.2. Requirement 2 (4 points)

Create a new function that given a currency code returns TRUE if the currency code exists or FALSE if it does not exist. Input should always be capitalised once we receive the parameter.

> **Hint**: For a currency code to exist, it should be USD or appear in any of the two conversion tables where you import your CSV as a column name.
> **Hint**: use information_schema.COLUMNS

2.3. Requirement 3 (4 points)

Create a new procedure that given **two parameters** (date to be checked, and currency code) returns in **another parameter** FALSE if the given date is not present in conversion tables (check both tables) or the value for our specified currency and date is 0 or null. Return TRUE otherwise.

> **Hint**: you may need to use a prepared statement: https://dev.mysql.com/doc/refman/8.4/en/sql-prepared-statements.html

2.4. Requirement 4 (10 points)

As you may know, in every music festival you can find people from around the world. Therefore there are many coin currencies moving in and out.

In this requirement we need to create a **stored procedure** to solve currency conversions form USD and to USD using the conversion tables.

Create a stored procedure with the following **parameters**:

1. Currency of origin: The 3 letters of the currency of origin (name of the column, the tables of conversions).
2. Currency of destination. The 3 letters of the currency of destination (name of the column, the tables of conversions).
3. Date of conversion. Date of the conversion could be today or we could need to convert with a past date.
4. Numeric amount to be converted. Value that is needed to be converted. This parameter will also be reused to return the value after the conversion *(INOUT parameter)*.
5. Error message. Parameter with an error text if something goes wrong.

Considerations:

1. Make use of the previous developed functions to simplify the code.
2. Assume that the date of conversion will always be given as 'YYYY-MM-DD' format.
3. Input currency parameters should be converted to uppercase before use.
4. If there is no problem with our checks, the error message parameter should return null value.
5. If there is a problem with our checks, the returned amount parameter should be 0.
6. The monetary amounts must have only two decimal places.
7. If some of the following considerations fails (8 to 14), an error message should be written in the specified parameter for that matter.
8. The amount to be converted should be bigger than 0.
9. Conversion date should be in the past up to today; not a future date.
10. The currency of origin should be USD or a currency in our conversion tables.
11. The currency of destination should be USD or a currency in our conversion tables.
12. The currency conversion has to always involve USD (from or to) and it makes no sense to convert to and from the same currency.
13. Conversion rate should exist for the given date and should not be 0 or null.

Hint: You can use the LEAVE command to end the procedure before ending all the execution.

2.5. Requirement 5 (8 points)

As you may have detected, the conversion tables only have data until 2024-10-24, we need a procedure to insert new data as needed.

In the action of inserting new conversion data for every currency, two situations can be found:

1. The last conversion rate has value 0: then we will keep a 0 for further days.
2. The last conversion is a value >0: then we will pick a random currency conversion value bigger than 0 from some day of the last year.

Keep in mind that from and to USD **conversions** should be **consistent** *(it must be the consistent changing from USD to EUR than to EUR to USD)***.**
For each currency, the **random** pick should be different.

At the moment of execution, this procedure will create conversion entries *(new rows)* from the next missing day in the conversion tables and until the next day after the execution of the procedure.

> **Hint:** Try first to populate the next missing day in conversion tables and then automate this action with a LOOP for the rest of the days in the past.
> **Hint**: Use RAND() and DATE_ADD functions.

2.6. Requirement 6 (2 points)

Executing every day the currency conversions for the next day by hand is hard, easy to forget and prone to errors.

Schedule the execution of the procedure in requirement 5 **every day until the end of the year** using this timetable:

Class-Group	Hour	Minute
P101	20h	Group number
P102	21h	Group number
P201	22h	Group number
P202	23h	Group number

For example, group P201_07 should do the execution at **22**:**07**h, as 22h is their specified hour, and minute 7 is their group number.

2.7. Requirement 7 (3 points)

Make a function to identify default or preferred currency for a given person/festivalgoer. Given an id as input, it will return:

1. *USD* if the person/festivalgoer has United States nationality.
2. *CAD* if the person/festivalgoer has Canada nationality.
3. *GBP* id the person has U.K. nationality.
4. *JPY* if the person/festivalgoer has Japanese nationality.
5. *EUR* if the person/festivalgoer is an Eurozone member.
6. 0 if the id is not found.
7. null in any other case.

2.8. Requirement 8 (10 points)

To keep track of bar and beer sellings, for every new consumption or beer sold we need to keep track of the payments automatically.

At least, all this information must be filled in a new payments table:

1. **Id** of the transaction and if it's from the beerman team or from a bar.
2. **Date and hour** of the transaction.
3. **Price** of the consumption that is in USD by default. (If it's a beer sold by a beerman, assume a price of 3$).
4. **Id** of the buyer.
5. The default **currency** for the buyer (use req. 7, if it returns null, use a random currency that does not default to 0 in our conversion tables).
6. The **price** in the default currency (use req. 4)
7. **Status** of the payment: True if all is OK or false if currency conversion raises any error.
8. Error: null if it's all OK or the raised error in the currency conversion.

Create all the needed database objects and structures to solve this requirement. Ensure data consistency: we cannot be able to store fake data.

2.9. Requirement 9 (3 points)

Eventhough we already provided all music festivals with food dishes that include rice and vegetables, the artist Lew Sid is now angry because Hanumandkind and Kalmi are generating more Youtube views with their new song *'Big Dawgs'.*

So he hired a hacker in order to manipulate the played songs in the festivals. The hacker inserted the data of Lew Sid's song called *'Fried rice'* in the database and he also created a programmed virus that, for each song played in a show, it will create fake data inserting a new row for Lew's song as if it were also played in the same show.
The goal of Lew Sid is to make his song the most played one and it seems that you are the hacker. Help him or pay the consequences.

2.10. Requirement 10 (5 points)

As music festival organisers, we don't want to be in trouble with the law. So create a procedure that is able to identify all the consumptions of the under age (<18) in order to "fix" them.
The goal is that NO minors have bought any alcoholic drink in our festivals and we will defend this until the grave.

If some minor bought a beer to a beerman during a concert, it should be considered as a system error and we will assign this consumption to the festivalgoer number 998192 in order to *fix it*.

Regarding the bars' sales of alcoholic drinks to under age youngsters, we need to correct their data and assign those consumptions to the festivalgoer 27582.

> **Warning**: Keep in mind that you are about to modify data of the given database; ensure that your approach is correct.

2.11. Correction criteria

This part will be evaluated **with 50 points**, on this terms:

- Each requirement has its own points specified.
- Each requirement will be evaluated on those terms:
  - **Zero points** if the requirement is not done, have compilation errors or does not deliver what is asked.
  - **Half** the requirement **points** if it compiles and delivers nearly all of what it is asked.
  - **Full** requirement points if it compiles and delivers all of what it is asked.