Homework Assignment #1
Database Systems – Fall 2022/2023

- This assignment has two parts, your final submission should contain a single zip file containing all submission requirements from both parts and a details.txt file that should contain the ID and name of each submitting student, such as:

  first_name last_name 123456789

  first_name last_name 987654321

  In total, your zip file needs to contain:

  1. details.txt
  2. 11 .sql files for each question in part A.
  3. reviewer.py

  Make sure you don't add any other unnecessary files to the zip. The zip file should be named with your ID, such as "123456789.zip". If multiple students are submitting, then separate the ID numbers with underscore, such as "123456789_987654321.zip".

- Submission is through the course website and in pairs – only one of the two should submit and that student will also be the one to receive feedback. A submission for three people is not permitted, if you are unable to find a pair then you should submit by yourself (it is not necessary to ask for permissions to submit by yourself).
- For both parts you need to have on your computer:
  - Mysql server
  - Mysql workbench (recommended)
  - Python connector
  - Sakila datbase

    Installation instructions are available on the course website in the *MySQL Installation Guide* file and in the recitation1 presentation.

Part A – SQL Queries

Objective

Understanding and manipulating SQL queries.

Data

All the questions in this part will assume the Sakila database.

Requirements

For each question you are required to submit a single file called **q{n}.sql** (for example q1.sql, q2.sql etc.) that will include the following:

- The SQL query you used.
- Any assumptions you made as notes (with preceding '#'). If you think that the question can be understood in more than one way, explain according to which interpretation you solved it.
- Documentation, if necessary.

Important Notes

- Your query must return the answer to the question exactly; no more and no less attributes or rows.
- Do NOT return duplicated rows in the answers, unless you are specifically asked to do so.
- Do NOT use views, they are not in the scope of this exercise
- Please make sure that the queries are well formatted (use tabs and newlines, parentheses etc.) to make them readable (See the example format).
- Make sure that your files can be executed without errors in MySQL.

Example Format

```
# Any assumptions you made regarding q1 should be written here...

SELECT      film_id, title
FROM        film
WHERE       length > 10
ORDER  BY   title
```

Questions

1. Find all category names that contain the letter 'a' only once. (Hint: Regular Expression - you can use MySQL documentation for more information: https://dev.mysql.com/doc/refman/8.0/en/regexp.html).
2. Find all children-friendly films (i.e, less than 1.5 hours long and rated PG or G). Return film_id, and title.
3. We would like to find inconsistencies between the title of a film in the *film* table and the title of the film in the *film_text* table. For each pair of non-identical titles of the same film_id, return the title from the *film* table and the title from the *film_text* table. Order the results alphabetically by the title from the *film* table.
4. Which customer rented the largest number of films in May 2005? Return the customer's full name in a single column.
5. List the average movie length for every film category.

6. Find all the actors who have acted in at least 10 movies more than the average number of movies per actor. For example, if the average number of films per actor is 6, return actors who have played in 16 or more films. Return the first and last names of these actors, ordered by their first names and then by their last names.
7. How many weeks was the longest film rental? Who is the customer? What film did this customer rent? Return the customer's full name (in one column), the film title and the rental duration in weeks (rounded to the first 4 decimal points).
8. Find all customers and actors that do not have an actor or customer respectively with the same first name (Assume there are no actors who are also customers).
9. Add constraints to the Sakila database which require that no last name appear as both a customer and a movie actor
10. Assume that we added a constraint with the script:

    ```
    ALTER TABLE film

    ADD CHECK (length>=10);
    ```

    Now we would like to delete it. To do so we need to run the following script:

    ```
    ALTER TABLE Persons

    DROP CONSTRAINT constraint_name;
    ```

    But we are missing the constraint name. What query can we run to get the film table constraint's names?

11. Return the store name, and the total "earning difference" of the store who had the largest difference in earning, between two succeeding months. e.g., if store A earned 24$ on July 2005 and 38,000$ on August 2005 then the difference is 37,976$, which may be the highest.

Part B - Using Databases with Python

Objective

Get some hands-on experience working with databases.

Requirements

Introduction

The *sakila* database is a toy example of a DVD rental store. In order to recommend films to customers, the store owner decided to collect film ratings. Your task is to help the store owner by adding film ratings to the existing sakila database. Specifically, you need to create a program that enables film reviewers to add film ratings.

Reviewer Interface

You will create a python program that runs in the command line and allows a reviewer to add ratings for films. The following steps describe how the reviewer will interact with your command line program.

Step 1 – Asking for the reviewer ID

When the program starts, it will first ask the reviewer for their ID. If the ID doesn't exist in the database, then the program should continue to step 2. Otherwise, it should continue to step 3.

Step 2 – Asking for the reviewer's name

In case the reviewer's ID doesn't exist, we will now ask the reviewer for their name. First, we will ask for the first name, and then for the last name.

This information should be saved in the database. If a reviewer enters the system for a second time, then it should remember their name, and shouldn't ask for it again.

Step 3 – Asking for a film name to rate

The reviewer should be greeted with a message in the following format: "Hello, <first_name> <last_name>". The reviewer should then be prompted to rate a film.

First, they will be asked to enter the film name. The program should then identify if the film name exists in the database or not.

- If the film name does not exist in the database, the program should tell the user that it doesn't exist and ask for a different film name.
- If the film exists and there is more than one film with this name, the user will be presented with the IDs of the films, together with their release year, and the user will have to choose a film from the list based on the film ID. An invalid input from the user (an ID not in the list) will start over and ask for a film name again. If the input is valid, then continue.
- If the film exists and there is exactly one film with this name, then continue.

Step 4 – Asking for a rating

Next, once the reviewer chooses a film, they will be asked to enter a rating. The program should identify an invalid rating. If the rating is invalid, the program should ask for a new rating. If the rating is valid, it should be saved to the database.

If the film was already rated by the reviewer, then the previous review should be updated.

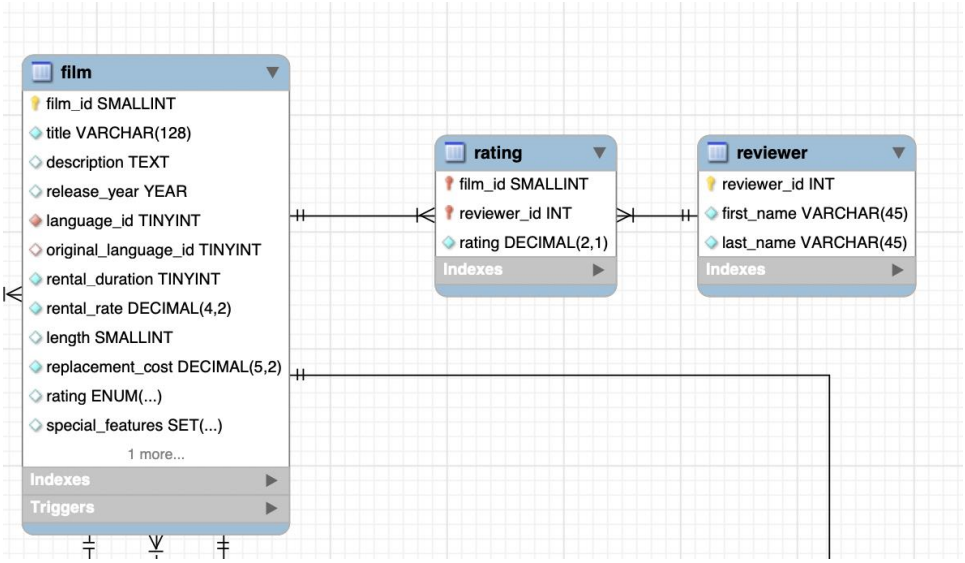
Step 5 – Print all ratings

The program should then print to the reviewer all the ratings in the database, up to 100 results. For each rating, print the film title, the reviewer's full name (first name and last name concatenated), and the rating. (Note: the concatenation should be implemented as part of the SQL query, and not as part of the python code)

Definitions

This section describes the formal definitions of what is a rating, and what is a reviewer.

Table design



Film

The film table already exists in the *sakila* database, so you don't have to make any changes to it.

Rating

1. A rating is a single digit with a single floating point. For example, valid ratings are 9.5 and 2.3. The highest possible rating is 9.9, and the lowest possible rating is 0.0. Examples for **invalid** rating are 2.33 (two floating points), 9 (no floating point), 10.0 and –1.0 (both are outside the range of valid values).
2. A rating record rates a single film, uniquely identified by *film_id*.
3. A rating record is rated by a single reviewer, uniquely identified by *reviewer_id*.

A reviewer can rate a film only once, and the rating can't be null.

Reviewer

1. A reviewer has an ID (an int, not null). No two reviewers can have the same ID.
2. A first name (up to 45 characters, not null).
3. A last name (up to 45 characters, not null).

Database Restrictions

Constraints

It is important that if someone tries to input invalid values directly into the database from MySQL workbench, and not from the code, they will fail. Therefore, the constraints mentioned above should be implemented as part of the database, and not as part of the code.

Referential Integrity

It is possible that a film or a reviewer will be deleted from the database, or their ID will be updated. You need to design the database such that in either case, these changes will cascade to the *rating* table. This means that if you delete a reviewer or a film, the ratings of the reviewer or the film should also be deleted. If you update a reviewer id or a film id, the reviewer id or film id should also change in the rating table.

Code

Getting started

1. Install MySQL and MySQL Connector/Python.
2. Create the sakila database.
3. Use the course github for numerous examples of working with databases from python (link).

Code requirements

- The code must be written in Python, and the entire code should be in a single file called *reviewer.py* .
- The code should create the tables. When the program starts, you should first check if the *rating* and *reviewer* tables exist. If not, you should first create them. You can assume that the *film* table is already created. **Do not create the tables manually from the MySQL client!**
- The code should use prepared statements if running the same query twice with different parameters.
- Consider possible sudden crashes of your code. Think about which C/U/D operations should be grouped, and use commit as soon as possible.
- Any input from the user that is being used for queries should be sanitized (i.e., prevent an SQL injection).
- Tip: when writing long queries, you can use three quotation marks to avoid writing the long query in a single line. For example:
    - query = “””
      SELECT *
      FROM film
      “””

Connecting to the database

You should connect to the database with the *root* user and password. However, you should not submit your root password. Instead, your code should read the password from a file. Specifically, your file should be called *.env*, and you do **not** submit this file. The contents of the file should look like this (change *MY_PASSWORD* to your actual password):

```
MYSQL_ROOT_PASSWORD=MY_PASSWORD
```

In the code, you can then get your password by using the following lines:

```
# Load dotenv file
from dotenv import load_dotenv
load_dotenv()


# Read the password from the environment variable
password = os.getenv('MYSQL_ROOT_PASSWORD')
```

To be able to run these lines, you need to install the *python-dotenv* package by running: *`pip install python-dotenv`* . If there are any questions, please ask in the moodle forum course.

Running python programs

To start the program, running the following line from the command line should work:

```
> python reviewer.py
```

External python packages

Your code should run only with the *MySQL Connector/Python* package, and the *python-dotenv* package. If you want to use any other library, you first need to get an approval by writing a message in the moodle forum course. Use only the moodle forum course, so we can approve or reject it once for all the students.

What to submit

A file called reviewer.py in the submitted zip file.

Reminder: do **not** submit the *.env* file with your password.