Homework 2 : Data Challenge

(Deadline as per Canvas)

This homework deals with the following topics:

- Loading data from .csv files into your local SQLite database
- Creating tables in SQLite
- Writing queries to create new tables from existing tables

The Assignment

Part 1 - Form Groups

As with the prior assignment, homework 2 will be completed in groups. For exact details on group size, and forming your groups, please refer to Ed Discussion.

Part 2 - Setup:

a. Make sure you have the files below in the Codio submit folder:
   i. Data_Challenge.db - an empty database
   ii. movies.csv
   iii. ratings.csv
   iv. student_queries.py
b. Connect to the .db file we provided within DBeaver
c. Create a new SQLite script for the database you just connected to

Part 3 – Examine the .CSV files:

To view any of the .CSV files open them in a text editor of your choice. **DO NOT use excel** when opening and examining the files to avoid formatting issues.

Movies Data File (movies.csv) should look like the below:

movies.csv

```
1   movieId,title,genres
2   1,Toy Story (1995),Adventure|Animation|Children|
3   2,Jumanji (1995),Adventure|Children|Fantasy
4   3,Grumpier Old Men (1995),Comedy|Romance
5   4,Waiting to Exhale (1995),Comedy|Drama|Romance
6   5,Father of the Bride Part II (1995),Comedy
7   6,Heat (1995),Action|Crime|Thriller
8   7,Sabrina (1995),Comedy|Romance
9   8,Tom and Huck (1995),Adventure|Children
10  9,Sudden Death (1995),Action
11  10,GoldenEye (1995),Action|Adventure|Thriller
12  11,"American President, The (1995)",Comedy|Drama
13  12,Dracula: Dead and Loving It (1995),Comedy|Hor
14  13,Balto (1995),Adventure|Animation|Children
15  14,Nixon (1995),Drama
16  15,Cutthroat Island (1995),Action|Adventure|Roma
17  16,Casino (1995),Crime|Drama
18  17,Sense and Sensibility (1995),Drama|Romance
19  18,Four Rooms (1995),Comedy
20  19,Ace Ventura: When Nature Calls (1995),Comedy
```

Each line of this file after the header row represents one movie.

There are 3 columns to consider:

- movieId (int)
- title (varchar)
- genres (varchar)

Note: Movie titles include the year of release in parentheses and genres are a pipe-separated list

Ratings Data File (ratings.csv) should look like the below:

ratings.csv

```
1   userId,movieId,rating,timestamp
2   1,16,4.0,1217897793
3   1,24,1.5,1217895807
4   1,32,4.0,1217896246
5   1,47,4.0,1217896556
6   1,50,4.0,1217896523
7   1,110,4.0,1217896150
8   1,150,3.0,1217895940
9   1,161,4.0,1217897864
10  1,165,3.0,1217897135
11  1,204,0.5,1217895786
12  1,223,4.0,1217897795
13  1,256,0.5,1217895764
14  1,260,4.5,1217895864
15  1,261,1.5,1217895750
16  1,277,0.5,1217895772
17  1,296,4.0,1217896125
18  1,318,4.0,1217895860
19  1,349,4.5,1217897058
20  1,356,3.0,1217896231
21  1,377,2.5,1217896373
22  1,380,3.0,1217896030
23  1,457,4.0,1217896015
```

Each line of this file after the header row represents one rating of one movie by one user

There are 4 columns to consider:

- userId (int)
- movieId (int)
- rating (decimal)
- timestamp (datetime)

Note: Ratings are made on a 5-star scale, with half-star increments (0.5 stars to 5.0 stars).

The timestamp column represents the number of seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970

Part 4 – Update the database
Important reminder: When we ask for a table to be created it must end up in your final .db file. The sub-steps (a, b) will contain the names you must give each column, and a 2nd required query to show us your table. The 2nd query will always be a SELECT FROM operation with extra requirements to give a part of your table.

Editing or creating tables: For this homework each step will require the creation of a properly named table. This way we can independently look at the results of each query you wrote during grading. Note that you will be working locally using DBeaver, and eventually uploading all your work to Codio. You must make sure everything is successfully uploaded to Codio, we will not be able to see your local work.

Group work:
We want to present two options for working as a group during this assignment. These options should limit issues needing a prior part of the SQLite script to complete a later part.

Fully synchronous group work option:
- We suggest working on each step together, coming up with your own solutions, and collaborating to find the best way to write each query. This way, all group members cover all concepts, and limit potential issues going through the steps, out of order.

Partly asynchronous option:
- Steps 1-2 must be completed as a group before any other parts.
- Then you can break steps 3 to 11 into asynchronous work. These sets of steps are recommended to allow for 6 to 8 to be completed before 3 to 5, or vice versa, as long as steps 1 and 2 have been completed first.
- Recommended sets of steps for the asynchronous option:
    - *Steps 1 to 2:* Completed synchronously
    - *Steps 3 to 5:* Completed by member A
    - *Steps 6 to 8:* Completed by member B
    - *Steps 9 to 11:* Completed by member C
        - Note: Step 11 contains the statement of work, and submitting the assignment

- Please ask questions of each other, and continue to work together to come with additional questions to ask on Ed.

1. **Use a SQLite Query to create “movies” and “ratings” tables to hold the data in the Data_Challenge.db**

   a. Use the .csv files you just examined to determine the column names, and column types. The column names must be identical to the .csv file column names, in the same order.
   b. varchar(50) can be used for title and genres, while for rating you will want to use DECIMAL(10, 5)
   c. Default values should be '0', '0.0', or NULL depending on the type of the column, note that the integer and doubles must have the single quotes around them as shown in lecture. DO NOT make an NULL or NOT NULL statements for numeric values when creating tables.
2. **Import the .csv files data into the 'movies' and 'ratings' tables using the DBeaver GUI and then view the first 10 rows of each table with a query**
   a. Make sure all settings are their default when importing the .csv files
   b. The GUI must be used to import the csv files into the tables, do not use a query
   c. Write 2 queries to view the first 10 rows from the 'movies' and 'ratings' table
3. Your next table must be called 's3_users_count', in your single query you must create the table, and fill it with relevant data from the 'ratings' table. We want to see how many times each userId left a review. **See step b for writing a 2nd query to show your results.**
   a. This table should have 2 columns: 'userId', 'num_of_reviews'
   b. Write a 2nd query to view all columns in your new table, but only show rows where the number of reviews is strictly greater than 500.
4. Your next table must be called 's4_popular_movies', in your single query you must create the table and fill it with relevant data from both the 'movies' table and the 'ratings' table. The goal of this table is to match each movie title with the number of times it was reviewed. **See step b for writing a 2nd query to show your results.**
   a. The resulting table will have 3 columns: 'movieId', 'title' and 'num_of_reviews'
   b. Write a 2nd query to view ONLY 2 columns 'title' and 'num_of_reviews'. Write the query in a way where only the top 10 most reviewed movies will be shown.
5. Your next table must be called 's5_highest_stars'. After seeing the most reviewed movies, we want to see the highest rating movies to compare. In your single query you must create the table and fill it with data on each movie's id, title and their average star rating.
   a. The resulting table will have 3 columns: 'movieId', 'title', 'average_rating'
   b. Write a 2nd query to view ONLY 2 columns 'title' and 'average_rating', and filter by 'title' to only include titles which have 'Toy' in them. Order the results by average_rating in ascending order. Hint: This should return 8 results.
6. Your next table must be called 's6_converted_date'. The ratings table currently contains a timestamp column which is a unix timestamp. Convert the unix timestamp into a human readable format. It should be exactly in the following format: 'YYYY-MM-DD HH:MM:SS'.
   a. The resulting table should have 4 columns userId, movieId, rating, date

   i. Do not include the old timestamp column, date is the new converted timestamp
   b. Write a 2nd query to view the first 10 rows of the new table
   c. Hint: https://www.sqlite.org/lang_datefunc.html view the datetime() function
7. Your next table must be called 's7_good_oldies. We want to filter the prior table from step 6 into a smaller number of just the best old movies. The table should have all columns in 's6_converted_date', but only includes the rows where **rating is strictly above 4**. Further **filter the results to only include dates before 1997-01-01** (not inclusive). It should further **sort the result by the rating column in descending order**.
   a. The resulting table will have 4 columns userId, movieId, rating, date
   b. Write a second query to get the count of rows in our new table. The result should be a 1 column table with the column alias being "row_count".
8. Your next table must be called 's8_frequent_critics'. In this table we want to know who the most common users were to review movies prior to 1997. Since you have this data from Step 7, you should use that existing table to make this new one. You should sort in descending order by review_count a new column you will make. The value in review_count should be the number of times that user made a review prior to 1997.
   a. The resulting table should have 2 columns userId and review_count.
   b. Write a 2nd query to show all columns and rows from this new table.
   c. Hint: There should be 102 rows
9. Create an exact copy of the movies table called 's9_movies'. You may use any method as long as it's done via a query and not the user interface.
   a. You do not have to write any query to show the table for step 9, but you may wish to in order to check your work.
10. You are not creating a new table in this step. Insert a new movie into the 's9_movies' table. It should be called '591 Breakout Hit (2024)' with the genre of 'Indie'. Its movieId must be 1 higher then highest movie ID in the current table (This the value in the column, not the value of the index) . You can write any query you want to get that information on the movieId. Make sure to use the name exactly as given in the instruction, including spaces.
   a. The resulting table should have 3 columns movieId, title, genres
   b. Write a 2nd query to show all columns with rows that have the genre of 'Indie'
   c. Hint: Ensure your new movie is in the table with the query we ask you to write in step b
11. This step is not a query. For step 11 you are responsible for some of the administrative steps in any group assignment:
   a. Add in the statement of work including all group members names, and a complete list of any resources you all used. Please collect resource information from other group members.
   b. Make sure the final database you're submitting has all 9 tables (2 from CSV imports, 7 derived)

c. **Add all your queries to student_queries.py**
    i. Do not change any of the variable names
    ii. Only add the queries between the quotation marks " "
    iii. If you have a quote in your query, put the backslash symbol before it like the following: **\'**
d. Submit in Codio on behalf of your group

What to submit

Please submit the SQL script you wrote as part of your work, Python file with your copied queries, and the updated .db file. Put all files in the submit directory in Codio, and run your pre-submission tests at least once. To add a file of the same name, you must first remove the starter files we give you.

You must name your files exactly as the below:

- student_queries.py - **Only edit where marked in file**
- student_script.sql
- Data_Challenge.db

In your script you must include a multiline comment at the very top which is a statement of work. Please include the names of your group members as well. To avoid academic integrity violations please include any resources you use. See the below image for an example:

```
/*
 * Statement of work
 * Names of Group Members:
 * List resources used:
 *
 *
 * */
```

How will you be graded – General outline at a high level

This assignment is out of a total of 100 points, and each section below is broken down further when grading. The first 40 points will be manually graded. The last 60 points are autograded and are the same as the pre-submission tests.

Regrades will not be accepted on the autograded portion, please run the pre-submission tests.

Administrative (10 points):

- 5 points - Submit all required files – your SQLite script, a python script with queries, and updated Data_Challenge.db file

- 5 points – Files are named exactly as the below:
    - Data_Challenge.db
    - student_queries.py - **Only edit where marked in file**
    - student_script.sql

Functionality (30 points):

- 20 points - All 9 requested tables exist in the .db and the INSERT INTO for step 10
- 10 points - Your SQLite script contains select queries when requested

Autograding for Accuracy (60 points):

- 30 points - All your final tables have the correct columns and rows. This also includes naming of the tables and columns
- 30 points - All select queries produce the correct result