Q2 [35 points] SQLite

SQLite is a lightweight, serverless, embedded database that can easily handle multiple gigabytes of data. It is one of the world's most popular embedded database systems. It is convenient to share data stored in an SQLite database — just one cross-platform file that does not need to be parsed explicitly (unlike CSV files, which must be parsed). You can find instructions to install SQLite here. In this question, you will construct a TMDb database in SQLite, partition it, and combine information within tables to answer questions.

You will modify the given `Q2.py` file by adding SQL statements to it. We suggest testing your SQL locally on your computer using interactive tools to speed up testing and debugging, such as DB Browser for SQLite.

Technology	• SQLite release **3.37.2** • Python **3.10.x**
Allowed Libraries	**Do not modify import statements**. Everything you need to complete this question has been imported for you. **Do not** use other libraries for this question.
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	• **`Q2.py`:** Modified file containing all the SQL statements you have used to answer parts a - h in the proper sequence.

IMPORTANT NOTES:

- If the **final output** asks for a **decimal** column, format it to **two** places using `printf()`. Do **NOT** use the `ROUND()` function, as in rare cases, it works differently on different platforms. If you need to sort that column, be sure you sort it using the actual decimal value and not the string returned by printf.
- A sample class has been provided to show example SQL statements; you can turn off this output by changing the global variable SHOW from True to False.
- In this question, you must only use INNER JOIN when performing a join between two tables, except for part 7 and 8. Other types of joins may result in incorrect results.

Tasks and point breakdown

1. [9 points] *Create tables and import data*.
   a. [2 points] Create two tables (via two separate methods, part_ai_1 and part_ai_2, in **`Q2.py`**) named `movies` and `movie_cast` with columns having the indicated data types:
      i. `movies`
         1. `id (integer)`
         2. `title (text)`
         3. `score (real)`
      ii. `movie_cast`
         1. `movie_id (integer)`
         2. `cast_id (integer)`
         3. `cast_name (text)`
         4. `birthday (text)`
         5. `popularity (real)`

   b. [2 points] Import the provided **movies.csv** file into the `movies` table and **movie_cast.csv** into the `movie_cast` table
      i. Write Python code that imports the `.csv` files into the individual tables. This will include looping though the file and using the **'INSERT INTO'** SQL command. Make sure you use paths relative to the Q2 directory.

   c. [5 points] *Vertical Database Partitioning.* Database partitioning is an important technique that divides large tables into smaller tables, which may help speed up queries. Create a new table `cast_bio` from the `movie_cast` table. Be sure that the **values are unique** when inserting into the new `cast_bio` table. Read this page for an example of vertical database partitioning.

      i. `cast_bio`
         1. `cast_id (integer)`
         2. `cast_name (text)`
         3. `birthday (text)`
         4. `popularity (real)`

2. [1 point] *Create indexes.* Create the following indexes. Indexes increase data retrieval speed; though the speed improvement may be negligible for this small database, it is significant for larger databases.
   a. `movie_index` for the `id` column in `movies` table
   b. `cast_index` for the `cast_id` column in `movie_cast` table
   c. `cast_bio_index` for the `cast_id` column in `cast_bio` table

3. [3 points] *Calculate a proportion*. Find the proportion of movies with a score between 7 and 20 (both limits inclusive). The proportion should be calculated as a percentage.
   a. Output format and example value:
      ```
      7.70
      ```

4. [4 points] *Find the most prolific actors*. List 5 cast members with the highest number of movie appearances that have a popularity > 10. Sort the results by the number of appearances in descending order, then by `cast_name` in alphabetical order.
   a. Output format and example row values (`cast_name,appearance_count`):
      ```
      Harrison Ford,2
      ```

5. [4 points] *List the 5 highest-scoring movies*. In the case of a tie, prioritize movies with fewer cast members. Sort the result by score in descending order, then by number of cast members in ascending order, then by movie name in alphabetical order.
   a. Output format and example values (`movie_title,score,cast_count`):
      ```
      Star Wars: Holiday Special,75.01,12
      Games,58.49,33
      ```

6. [4 points] *Get high scoring actors.* Find the top ten cast members who have the highest average movie scores. Sort the output by `average_score` in descending order, then by `cast_name` alphabetically.
   a. Exclude movies with score < 25 before calculating `average_score`.
   b. Include only cast members who have appeared in three or more movies with score >= 25.
      i. Output format and example value (`cast_id,cast_name,average_score`):
         ```
         8822,Julia Roberts,53.00
         ```

7. [2 points] *Creating views.* Create a view (virtual table) called `good_collaboration` that lists pairs of actors who have had a good collaboration as defined here. Each row in the view describes one pair of actors who appeared in at least 2 movies together AND the average score of **these movies** is >= 40.
   The view should have the format:
   ```
   good_collaboration(
         cast_member_id1,
         cast_member_id2,
         movie_count,
         average_movie_score)
   ```

   For symmetrical or mirror pairs, only keep the row in which `cast_member_id1` has a lower numeric value. For example, for ID pairs (1, 2) and (2, 1), keep the row with IDs (1, 2). There should not be any "self-pair" where `cast_member_id1` is the same as `cast_member_id2`. Remember that creating a view will not produce any output, so you should test your view with a few simple select statements during development. One such test has already been added to the code as part of the auto-grading. **NOTE: Do not submit any code that creates a 'TEMP' or 'TEMPORARY' view that**

you may have used for testing.

Optional Reading: Why create views?

8. [4 points] *Find the best collaborators.* Get the 5 cast members with the highest average scores from the `good_collaboration` view, and call this score the `collaboration_score`. This score is the average of the `average_movie_score` corresponding to each cast member, including actors in `cast_member_id1` as well as `cast_member_id2`.
   a. Order your output by `collaboration_score` in descending order, then by `cast_name` alphabetically.
   b. Output format and example values(`cast_id,cast_name,collaboration_score`):
   ```
    2,Mark Hamil,99.32
   1920,Winoa Ryder,88.32
   ```

9. [4 points] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation).
   a. [1 point] Import movie overview data from the **movie_overview.csv** into a new FTS table called `movie_overview` with the schema:
   ```
   movie_overview
       id (integer)
       overview (text)
   ```

   NOTE: Create the table using **fts3** or **fts4** only. Also note that keywords like NEAR, AND, OR, and NOT are case-sensitive in FTS queries.

   NOTE: If you have issues that fts is not enabled, try the following steps
   - Go to sqlite3 downloads page: https://www.sqlite.org/download.html
   - Download the dll file for your system
   - Navigate to your Python packages folder, e.g., C:\Users\... ...\Anaconda3\pkgs\sqlite-3.29.0-he774522_0\Library\bin
   - Drop the downloaded .dll file in the bin.
   - In your IDE, import sqlite3 again, fts should be enabled.

   b. [1 point] Count the number of movies whose `overview` field contains the word 'fight'. Matches are not case sensitive. Match full words, not word parts/sub-strings.
      i. Example:
         Allowed: 'FIGHT', 'Fight', 'fight', 'fight.'
         Disallowed: 'gunfight', 'fighting', etc.
      ii. Output format and example value:
         12
   c. [2 points] Count the number of movies that contain the terms 'space' and 'program' in the `overview` field with no more than 5 intervening terms in between. Matches are not case sensitive. As you did in h(i)(1), match full words, not word parts/sub-strings.
      i. Example:
         Allowed: 'In Space there was a program', 'In this space program'
         Disallowed: 'In space you are not subjected to the laws of gravity. A program.'
      ii. Output format and example value:
         6