Homework Assignment 4

SQL Database

Worth 100 points.

Posted Tuesday, November 30

Due Saturday, Dec 11 at 11 PM in Canvas

No Late Submissions Accepted

You will work on the assignment in pairs that you chose, or that we had randomly assigned. Make sure that as a pair, you abide by the **DCS Acadmic Integrity Policy for Programming Assignments.**

Implement a Music relational (SQL) database, of the kind that might be used by Spotify or Amazon Music. The database has artists, albums, songs, users, playlists, and ratings.

Type up all the required work specified in the following sections in any word processor, then convert to a PDF file named `music_db.pdf`. (Handwritten work is NOT acceptable.) Submit this file to Canvas - **only one submission per group, please.**

You are allowed up to 3 submissions, only the last submission will be graded.

NOTE: You may populate the tables with data for your own testing, but we **do not** want you to turn in any of the data, or the results of any of your queries. We are only asking for the document with the required SQL statements for table creation and queries.

Database Schema (50 pts)

You are given the following description of the entities that need to be stored in the database. Your task is to design a database schema (set of tables) to store these entities.

Your schema must be minimally redundant in storing data. In other words, you should build a set of tables that minimize the repetition of data, by using foreign keys - credit will be in accordance with this.

- **Artist**: An individual or a group/band, uniquely identified by their name. An artist might release albums, as well as songs that are not in albums (singles).
- **Song**: A song has a title and is performed by an artist, either as a part of an album, or as a single that's not part of an album. Every song in an album has the release date of the album, but a single song has its own release date. A song title is unique to an artist (the same artist records a song exactly once), but the title may be shared by multiple artists (i.e. covers).

  A song belongs to one or more genres. For example, a song could be in a single genre, such as *R & B*, or could be in multiple genres such as *Pop* and *Rock*. Genres are pre-defined, and every song must be in at least one genre. Also, songs in an album need not all be in the same genre.

- **Album**: An album is a collection of songs released by an artist, on a certain date. For example, the album *Achtung Baby* was made by the artist (band) *U2*, released on *November 19, 1991*. An album name is not unique, but the combination of album name and artist name is unique.

- **User**: A user is uniquely identified by their *username*. A user can optionally have one or more playlists, and optionally have ratings for songs, albums, or playlists. In other words, it's possible that a user has no playlists, and hasn't given any ratings.
- **Playlist**: A user can make any number of playlists of songs. Note: A playlist may not include an entire album, only individual songs. Each song is either from some album, or a single that's not in any album.

  Every playlist has a title, and a date+time when it was created. A playlist may be modified any number of times after creation by adding or removing songs, but the title and date+time will not change.

  The title of a playlist is not unique since different users might create playlists with the same title. However, a user's playlists will have unique titles.
- **Rating**: A user could rate an album, a song (even if it's in an album), or a playlist. A rating is limited to 1,2,3,4, or 5 (numeric), and is made on a specific date.

Your database structure should have the most appropriate data type and size for each column in each table.

For size of data, think of a realistic online music service and imagine how many songs/artists/albums/playlists/users/ratings it might have to support. The idea is to use the least amount of storage space for each column that will be able to store the entire range of foreseeable values.

Make sure you define and specify all primary keys, foreign keys, unique valued columns or unique valued combination of columns, and null/non-null properties for columns.

In the document you will submit, type in the `create table` statement for each of the tables you create in the database. If you don't have the full create statement for a table, you will not get credit for it.

Note: When you test your design in MySQL, you might use `alter table` statements after the initial create. However, for the submission, you are required to rewrite the whole sequence as a single `create table` statement per table.

Queries (50 points)

Every query must be written in a single SQL statement, meaning that if you were to write it in a MySQL client session on a terminal, there would be a single terminating semicolon. So, for example, you can have nested or multiple SQLs for a query, provided you can write it all up with a single terminating semicolon in a MySQL client session. No Python code!

For any of the queries:

- If the result might require breaking ties, then *unless otherwise specified in the query*, let the MySQL engine deal with it (you need not do anything explicit)
- If the result has fewer than the required number of entities, report all of them.
- For all queries that ask for 'top n' or 'most', the result must appear from highest ranked to lowest ranked.

Type the SQL queries in the document you will submit, and make sure to write the query number against each query. (If you want to play it extra safe, copy the query statement from this list, then write your answer SQL query.)

Write queries for the following.

1. Which 3 genres are most represented in terms of number of songs in that genre?

   The result must have two columns, named **`genre`** and **`number_of_songs`**.

2. Find names of artists who have songs that are in albums as well as outside of albums (singles).

   The result must have one column, named **`artist_name`**

3. What were the top 10 most highly rated albums (highest average user rating) in the period 1990-1999?. Break ties using alphabetical order of album names. (**Period refers to the rating date, NOT the date of release**)

   The result must have two columns, named **`album_name`** and **`average_user_rating`**.

4. Which were the top 3 most rated genres (this is the number of ratings of songs in genres, not the actual rating scores) in the years 1991-1995? **(Years refers to rating date, NOT date of release)**

   The result must have two columns, named **`genre_name`** and **`number_of_song_ratings`**.

5. Which users have a playlist that has an average song rating of 4.0 or more? (This is the average of the average song rating for each song in the playlist.) A user may appear multiple times in the result if more than one of their playlists make the cut.

   The result must 3 columns named **`username`**, **`playlist_title`**, **`average_song_rating`**

6. Who are the top 5 most engaged users in terms of number of ratings that they have given to songs or albums? (In other words, they have given the most number of ratings to songs or albums combined.)

   The result must have 2 columns, named **`username`** and **`number_of_ratings`**.

7. Find the top 10 most prolific artists (most number of songs) in the years 1990-2010? Count each song in an album individually.

   The result must have 2 columns, named **`artist_name`** and **`number_of_songs`**.

8. Find the top 10 songs that are in most number of playlists. Break ties in alphabetical order of song titles.

   The result must have a 2 columns, named **`song_title`** and **`number_of_playlists`**.

9. Find the top 20 most rated singles (songs that are not part of an album).
   Most rated meaning number of ratings, not actual rating scores.

   The result must have 3 columns, named **`song_title, artist_name, number_of_ratings`**.

10. Find all artists who discontinued making music after 1993.

    The result should be a single column named **`artist_title`**