Homework Assignment #1

(SQL Refresher)

Congratulations are in order! You've just landed a database development position at a startup where you will be helping to create and launch a new *peer-to-peer* e-commerce service.

The New Startup

After attending various AI (artificial intelligence) music showcases and watching the growing popularity of generative AI in the music industry, you and a group of fellow UCI students have recognized a unique opportunity to create a platform that fully embraces AI-generated music. With some initial seed funding from the up-and-coming venture firm Synth Capital, your team plans to launch a platform tentatively called ***ZotMusic***. The idea is to give users—both artists and listeners—a space to release and explore AI-generated records. Artists will be able to showcase their creations, whether singles or full albums, while listeners will have the chance to discover new sounds and rate the content, contributing to a rich, engaging musical community. To enable AI artists to reach broader audiences, ***ZotMusic*** will also support features like location-based music recommendations and allow listeners to enjoy music seamlessly with or without a subscription. The platform's review and rating system will help users navigate the ever-growing library of AI-generated content, fostering trust and engagement between creators and listeners. As you plan to launch ZotMusic in time for the upcoming music festival season, you are eager to get development underway!

The Conceptual Data Model

An informal description of the business along with an E-R diagram of ZotMusic's conceptual data model is available (ZotMusic service vision).
Your first assignment is to get your hands on the data, familiarize yourself with it, load it pretty much "as is" into a relational database system, explore it via SQL queries to refresh your SQL skills, and then report back to your boss about what you did so far and how. You should also comment on any changes that you'd recommend to their current database plans, were they to decide to "stay relational" in terms of the underlying database platform.

Get Ready...

You should start by adding PostgreSQL to the set of applications that you have installed on your favorite laptop or another computer if you don't already have it (refer to the PostgreSQL setup document for helpful links). Once you manage to get Postgres installed, you should fire it up and use its ***psql*** command-line interface to create a toy table, insert a few toy rows, and then run a few toy queries to make sure that you're good to go. Once you get this far, you will probably also want to download and install a copy of pgAdmin4 so that you'll have a nice GUI to use to work with PostgreSQL.

Get (Data) Set...

Now that you have PostgreSQL installed and working, the next step is to grab yourself a copy of the relevant ***ZotMusic*** data.To facilitate that process, your boss has given you (a) a set of SQL DDL statements that the company's chief architect created from the E-R diagram, along with (b) a Google Drive location for a collection of actual CSV (comma-separated value format) files containing a set of representative data records for ***ZotMusic***.

*(a) SQL DDL (for a set of **ZotMusic** tables):*

PostgreSQL DDL script

*(b) Data Files (for the **ZotMusic** data itself):*

ZotMusic HW1 CSV Files

Go...!

Okay, it's time to get to work! Here is what you are to do (using ***PgAdmin4*** or ***psql*** or whatever other PostgreSQL interface you choose to use):

1. Run the provided SQL DDL script to create an appropriate PostgreSQL table (using **CREATE TABLE**) for each of the 10 tables in the RDBMS schema. Study the tables, paying attention to the chosen data types (**int**, **float**, **varchar**, **timestamp, etc...**) for each of their columns. Also, be sure to look at the **PRIMARY KEY** and **FOREIGN KEY** constraints for each table. (Notice how those constraints capture the conceptual model features expressed in the provided E-R model.) ***To show that you have done so, very briefly justify the chosen data types and NULL/NOT NULL constraints for each field in the Songs table.***

2. Use PostgreSQL's **COPY** statement to load each of the 10 tables from the CSV data files. (Use Google to locate PostgreSQL's online documentation and bookmark a link to those docs for your chosen PostgreSQL release.) You may want to look at the first few lines of the CSV files as well as PostgreSQL's documentation for the **COPY** command to figure out the command options that you will need to successfully import the data.

3. Explore the given data by using SQL to formulate and answer each of the following questions (queries) on it. Translate each of the following English queries into an appropriate SQL query (using ***one*** query per problem unless directed differently!) and show your queries and their results when you report back to your boss. Note for questions **A - I**, you can only use one **SELECT** statement.

    A. To start, you should get an overview of the lay of the land on ***ZotMusic***. Write a query (yes, just ***one*** query!) that prints out how many users, records, and reviews there are in the given data - i.e., find the total number of entities in each of those

three entity sets.

B. You're interested in understanding the nature of the user base. Find the users who are both artists and listeners whose email addresses are based at "icloud.com" and print their user_id, email address, nickname, and zip code.
C. You're interested in the records that have been released by a particular artist. Find the records that were released by the artist whose email address is "fwilson@outlook.com" and print their record_id, title, genre, and release_date. Order the listed records by their release date, from earliest to latest.
D. Next you want to drill down further to see what sort of records this artist is making. In addition to their user id, for each genre print the number of albums and the number of singles the artist has released in that genre.
E. You're now more generally curious about artists and their records' genres and you want to promote artists with a wider variety of records. Print the stage names and email addresses of artists whose set of released records span 9 or more genres. When printing their names, if an artist didn't provide a stage name, print their nickname instead.
F. Identify the users who have released records in the "R&B" and "Hip-Hop" genres but **not** in the "Indie" or "Jazz" categories. Print the user_id and email address of such users, and order the results by (ascending) user_id.
G. Now you want to turn your attention to users in general and their indicated interests in genres. Unfortunately, users' genres have been provided in a "cheating 1NF" format in a single attribute with a string of genres. As a first step towards making such searches against the given data, read the "Hints on Multivalued Data" appendix at the end of this assignment to learn about the **string_to_array()** function in PostgreSQL. You will also find the **array_length()** function useful, so check out the PostgreSQL documentation to learn about that function. Now you're ready to go! To get a sense of users' interests, write a query that lists the email address, nickname, and number of interested genres for each user. Order its output in descending order of number of genres (call it num_genres) and only print the first 10 such users (i.e., 10 users with the most interests).
H. Write a query against (just) the Users table that, for each genre available on ZotMusic, indicates how many users have included that genre in their genres attribute. You might also find the **UNNEST** keyword to be useful. Your query's output should end up being 20 tuples of the form (genre, num_users); list them in highest to lowest num_users order.
I. Your boss wants you to compile a list of the most popular songs on ZotMusic. Write a query to get the **top 10 songs** with the highest number of listeners. Note multiple streaming sessions from the same listener counts only once. Exclude the streaming sessions where a user "cuts" a song without listening to most of its content, i.e., it has NOT been replayed at least once and its remaining time is larger than 20% of the song's length. Print the record title, track number, song title and number of listeners.
J. Now it's time to study records' ratings! Views can be a helpful tool when preparing data for analyses, especially when some of the data is derived. Your boss has

specified that each record's rating (i.e., the derived attribute in the E-R diagram for ***ZotMusic***) should be a weighted average of all the reviews for this record, where the weight for each review is the number of upvotes that the review gets, plus 1 (so that reviews with no upvotes also get counted). Note the result of a weighted average rating should still range from 0 - 5. Based on that specification, then, create a view of the form *RatedRecords(title, record_id, rating, num_reviews)* that can be used by analysts to study records' ratings. Show that your view works as desired by printing the first 10 of its tuples for records with at least 5 reviews, in descending rating order.

K. You have heard a rumor that your data analysis colleagues are going to want to study the ratings of all the records. Thinking about performance at scale, i.e., if ***ZotMusic*** is successful, that rumor has you a bit worried. As a result you've decided to materialize the ratings in the Records table for faster access. Alter the database schema to ***store*** this information by writing an **ALTER TABLE** command to add a rating attribute to the Records table and then write an **UPDATE** command to deposit the current rating value for each row into the newly added field. (You can use your view to do this.) Last but not least, verify that you've successfully materialized the rating by writing a query to join the RatedRecords view with the table and print the rating field value from the Records table alongside (all) the information from the RatedRecords view. Again, print only the first 10 records with at least 5 reviews and in descending rating order. (The new rating value from Records and the overall value from RatedRecords should match!) Note: Don't worry about maintaining this value going forward; one could use triggers or one could periodically do what you just did, but we'll ignore that part for this HW assignment.

L. ZotMusic is considering giving "Top-rated Artist" awards to its content creators based on the artist ratings. An artist's content_rating is simply the average of their records' ratings. Use the RatedRecords view to write and run a query that lists the user_id, nickname, and content_rating of artists with a rating of 3.3 or higher. Run the query a few times to get a sense of its average execution time. In addition, dust off your query performance cobwebs by using PostgreSQL's **Explain** feature in PgAdmin (or its **EXPLAIN** command) to analyze the query plan for this query. Now, with the new field that you've added, you can write the same query using the Records table. Before you do, write a CREATE INDEX statement to index Records on the rating field. Then rewrite your RatedRecords query to get the answer from the Records table instead. Again, run your query a few times and also take a look at the query plan that PostgreSQL is using to run it. Compare your queries' running times with and without the materialized data (and its index) and briefly comment on the differences that you see, what you would expect to happen if the database grows to have many more sellers someday, and what could be done to further improve this query's performance.

M. Your boss is interested in doing some slicing and dicing of streaming sessions to analyze the music quality and device information. Use the ROLLUP operation in SQL to study the number of streaming sessions by musicQuality and by device. That is, for each musicQuality and device combination, use ROLLUP to count the

number of associated sessions for that musicQuality and device as well as the overall per-quality/session count and the overall (grand total) sessions count. Call your query's resulting count field "num_session" and order its results in descending order by this field value. Replace any ROLLUP-induced null values in the subtotal and total rows with the string "ALL" for user-friendlier presentation.

What To Turn In

Once you have finished the assignment you should use Gradescope to turn in a *PDF file* that lists all of the PostgreSQL statements that you ran – from start to finish – in order to load the data into the tables and run all of the queries – along with the results that they produce when run against the given data. Please follow the following steps in order to generate the SQL file for submission:

1. Download the HW1 template file and edit your copy of this SQL file as you work, and once your editing is done, you can export the final version as a PDF file and submit it to Gradescope.
2. Place the table creation statements from the given SQL DDL script, with any changes that you make, into the "Table Creation" section of the template (so you'll have a freshly loaded database instance when you run the queries to produce their results). Include your type/constraint notes for Items' fields as SQL comments (/* multi-line comment */).
3. Place all of your SQL COPY commands into the "Data Loading" section of the template.
4. Place all of your queries and supporting SQL for Question 3 into the appropriate spaces and make sure to include your notes as SQL comments (/* multi-line comment */).
    - *Note*: Problems K & L are near the end of the query set intentionally since problem J will have side effects on the database's schema and content.
5. Finally, after filling in your copy of the template file with your SQL statements, run it **all at once** using ***psql***'s command *"\i filename;"* option to make sure that all of your queries' answers will indeed be included in the output produced when you run it.
6. Once you have the resulting ***hw1.sql*** file all finished, with all of your commands and comments, open up your command-line tool or your terminal window and navigate to the folder where your *.sql* file is located.
7. Type ***psql*** *-a -f hw1.sql >> results.txt* (Note that *hw1.sql* is your final SQL input file and *results.txt* is an output file that will be produced by running your work - i.e., *results.txt* is the file where your SQL statements, interleaved with the results that they produce, will go. To get credit for this assignment it is **mandatory** that you turn in your work in this interleaved form.)
8. Open *results.txt* and re-verify that you indeed see the proper results after **every** query.
9. Create a copy of your results file as a PDF file (*results.pdf*), e.g., by "PDF printing" it.
10. Submit the resulting PDF file to GradeScope. Done!

APPENDIX: Hints on Multivalued Data

Running and pondering the following set of PostgreSQL statements together with their results will probably be helpful to you when you tackle a few of the queries in this first assignment:

```
CREATE TABLE Person (
    id int,
    name varchar(20),
    hobbies varchar(80)
);

INSERT INTO Person VALUES
   (1, 'Joe Cool', 'karate,skiing,skydiving'),
   (2, 'Susan Smith', 'scuba,piano'),
   (3, 'Hans Solo', 'flying');

SELECT * FROM Person p;

SELECT p.id, p.name, string_to_array(p.hobbies,',')
FROM Person p;

SELECT p.id, p.name, hobby
FROM Person p, UNNEST(string_to_array(p.hobbies,',')) AS hobby
ORDER BY p.id;
```