

**MINI PROJECT REPORT OF DBMS ON SPOTIFY MUSIC MANAGEMENT SYSTEM**

**PROGRAM NAME- BCA**

**SUBJECT – DATABASE MANAGEMENT SYSTEM**

**SUB CODE- 23CAT-251**

**SUBMITTED BY- ABHIJEET KUMAR SINGH**

**UID- 23BCA10377**

**CLASS- 4-B**

**SUBMITTED TO – MR. ARVINDER SINGH**

ACKNOWLEDGEMENT-

We would like to express our sincere gratitude to our mentor Mr. Arvinder Singh for his constant support, guidance, and valuable feedback in the development of the Spotify Music Management System. His insights into database design, SQL programming, and problem-solving were integral to the success of this project.

We also extend our appreciation to our peers, online forums, and resources like MySQL Workbench, dbdiagram.io, and Stack Overflow for their contributions during the course of the project.

**INTRODUCTION:-**🎶

The Spotify Music Management System is a relational database solution designed to manage user accounts, songs, playlists, albums, artists, and tracks. This system facilitates the management of music collections, user preferences, playlist creation, and interactions with artists and albums.

Importance of DBMS in Music Streaming

A robust DBMS is crucial for managing vast amounts of data, including music tracks, user profiles, playlists, and album information. It ensures that users can access songs, create playlists, and discover new music quickly and efficiently. It also helps in tracking user listening behaviour, optimizing recommendations, and managing streaming data in real-time.

**OBJECTIVES:-**

User Account Management – Store user details such as subscription type and playlists.

Song Management – Organize song details, genres, and availability.

Playlist Management – Manage user-generated playlists with a list of tracks.

Artist & Album Management – Store information about artists, albums, and tracks.

Music Recommendations – Track user listening behaviour and provide recommendations.

Streaming Reports – Generate statistics and reports on song plays, user activity, etc.

**SYSTEM FUNCTIONALITY**

Built on a relational DBMS, the system supports various entities such as users, songs, artists, albums, and playlists. The system uses primary keys, foreign keys, and normalization to ensure data integrity and support complex queries.

Main tables include:

1) Users

2) Songs

3) Artists

4) Albums

5) Playlists

6) Tracks

7) Streaming History

**TECHNOLOGIES USED**

DBMS: MySQL / PostgreSQL

**ER Diagram:** dbdiagram.io

**Tools:** MySQL Workbench, SQL Server Management Studio

**Languages (optional)**: Java/Python for GUI/backend

**EXPECTED BENEFITS:-**

1) Efficient Song and Playlist Management

2) Real-time User Profile Updates

3) Personalized Music Recommendations

4) Streamlined Playlist Creation

5) Detailed Analytical Reports

**ENTITY DETAILS & INPUTS: -**

1)Users: UserID, Name, Email, Subscription Type

2)Songs: SongID, Title, Genre, Duration, Release Year

3)Artists: ArtistID, Name, Genre

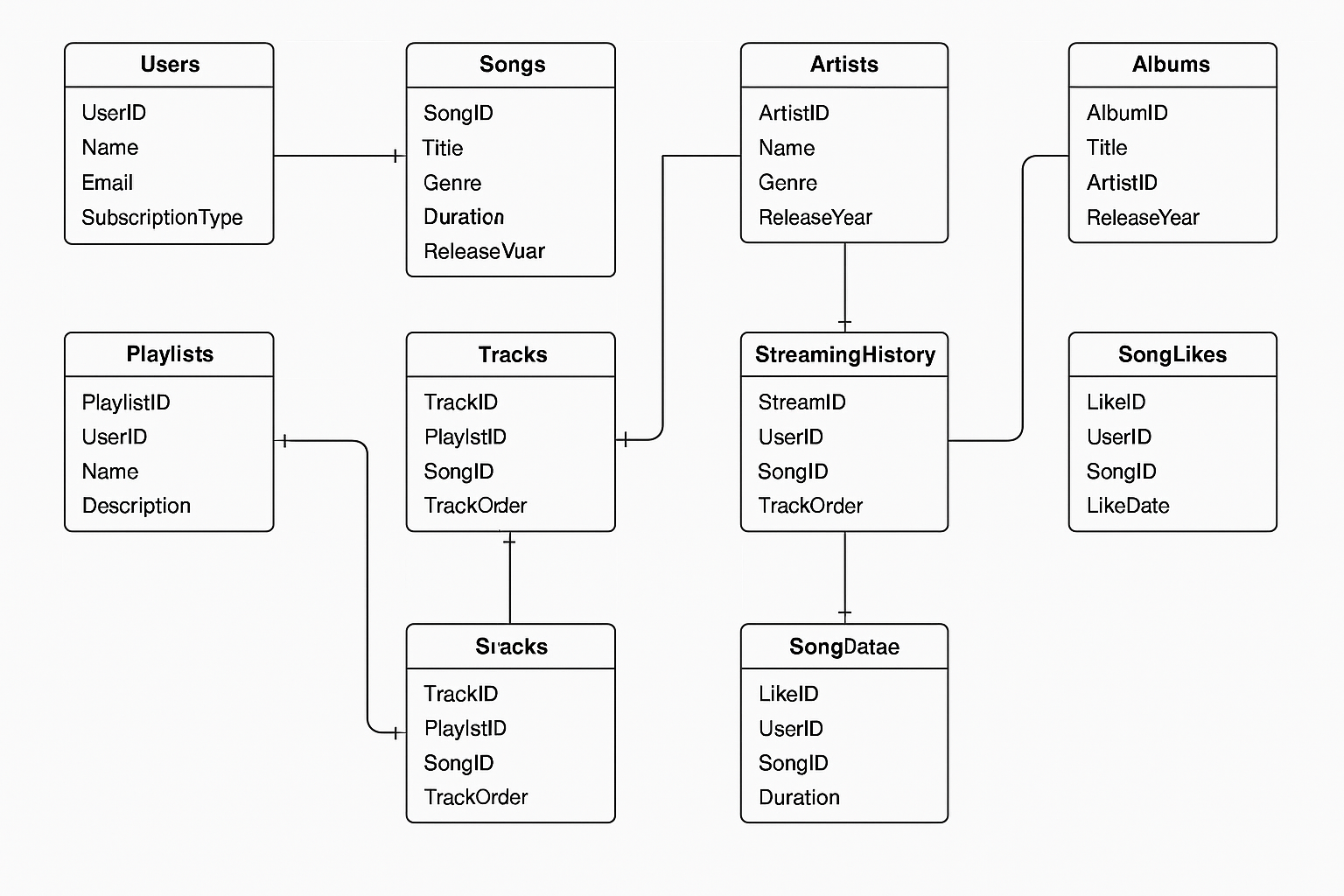
4)Albums: AlbumID, Title, ArtistID, Release Year

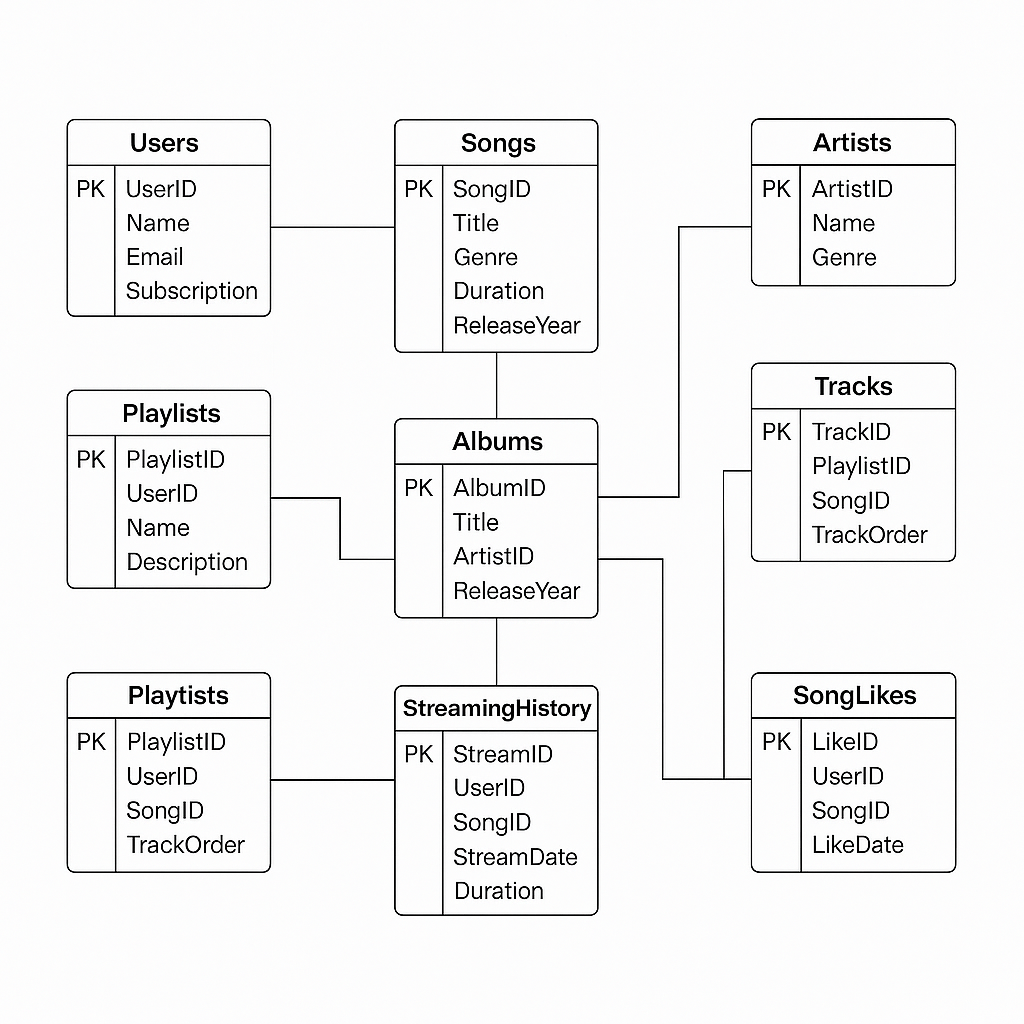
5)Playlists: PlaylistID, UserID, Name, Description

6)Tracks: TrackID, PlaylistID, SongID, Track Order

7)Streaming History: UserID, SongID, Stream Date, Duration

**ER DIAGRAMS: -**



**2) ER DIAGRAM: -**

**SQL CODE and Queries: -**

**CREATE DATABASE spotify;**

**USE spotify;**

**-- Users Table**

**CREATE TABLE Users (**

**UserID INT PRIMARY KEY AUTO\_INCREMENT,**

**Name VARCHAR(50) NOT NULL,**

**Email VARCHAR(100) UNIQUE NOT NULL,**

**SubscriptionType VARCHAR(20) NOT NULL**

**);**

**-- Songs Table**

**CREATE TABLE Songs (**

**SongID INT PRIMARY KEY AUTO\_INCREMENT,**

**Title VARCHAR(100) NOT NULL,**

**Genre VARCHAR(50),**

**Duration INT, -- in seconds**

**ReleaseYear INT**

**);**

**-- Artists Table**

**CREATE TABLE Artists (**

**ArtistID INT PRIMARY KEY AUTO\_INCREMENT,**

**Name VARCHAR(100) NOT NULL,**

**Genre VARCHAR(50)**

**);**

**-- Albums Table**

**CREATE TABLE Albums (**

**AlbumID INT PRIMARY KEY AUTO\_INCREMENT,**

**Title VARCHAR(100) NOT NULL,**

**ArtistID INT,**

**ReleaseYear INT,**

**FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID) ON DELETE SET NULL**

**);**

**-- Playlists Table**

**CREATE TABLE Playlists (**

**PlaylistID INT PRIMARY KEY AUTO\_INCREMENT,**

**UserID INT,**

**Name VARCHAR(50) NOT NULL,**

**Description TEXT,**

**FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE**

**);**

**-- Tracks Table**

**CREATE TABLE Tracks (**

**TrackID INT PRIMARY KEY AUTO\_INCREMENT,**

**PlaylistID INT,**

**SongID INT,**

**TrackOrder INT,**

**FOREIGN KEY (PlaylistID) REFERENCES Playlists(PlaylistID) ON DELETE CASCADE,**

**FOREIGN KEY (SongID) REFERENCES Songs(SongID) ON DELETE CASCADE**

**);**

**-- StreamingHistory Table**

**CREATE TABLE StreamingHistory (**

**StreamID INT PRIMARY KEY AUTO\_INCREMENT,**

**UserID INT,**

**SongID INT,**

**StreamDate DATE,**

**Duration INT,**

**FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE,**

**FOREIGN KEY (SongID) REFERENCES Songs(SongID) ON DELETE CASCADE**

**);**

**-- SongLikes Table**

**CREATE TABLE SongLikes (**

**LikeID INT PRIMARY KEY AUTO\_INCREMENT,**

**UserID INT,**

**SongID INT,**

**LikeDate DATE,**

**FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE,**

**FOREIGN KEY (SongID) REFERENCES Songs(SongID) ON DELETE CASCADE**

**);**

**-- Songs Table (Updated)**

**CREATE TABLE Songs (**

**SongID INT PRIMARY KEY,**

**Title VARCHAR(100),**

**Genre VARCHAR(50),**

**Duration INT,**

**ReleaseYear INT,**

**AlbumID INT,**

**FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID)**

**);**

**INSERT INTO Songs VALUES**

**(1, 'Song A', 'Pop', 210, 2023, 1),**

**(2, 'Song B', 'Rock', 180, 2022, 2),**

**(3, 'Song C', 'Hip Hop', 240, 2021, 3),**

**(4, 'Song D', 'Jazz', 300, 2020, 4),**

**(5, 'Song E', 'Classical', 240, 2019, 5),**

**(6, 'Song F', 'Pop', 200, 2023, 6),**

**(7, 'Song G', 'Rock', 230, 2022, 7),**

**(8, 'Song H', 'Pop', 220, 2023, 6),**

**(9, 'Song I', 'Classical', 260, 2021, 5),**

**(10, 'Song J', 'Jazz', 280, 2020, 4);**

**Query 1: Total streams and duration per song**

**SELECT Songs.Title, COUNT(StreamingHistory.StreamID) AS TotalStreams,**

**SUM(StreamingHistory.Duration) AS TotalDuration**

**FROM StreamingHistory**

**JOIN Songs ON StreamingHistory.SongID = Songs.SongID**

**GROUP BY Songs.Title**

**ORDER BY TotalStreams DESC;**

**Query 2: Top 5 most liked songs per genre**

**SELECT Genre, Title, TotalLikes**

**FROM (**

**SELECT Songs.Genre, Songs.Title, COUNT(SongLikes.LikeID) AS TotalLikes,**

**RANK() OVER (PARTITION BY Songs.Genre ORDER BY COUNT(SongLikes.LikeID) DESC) AS genre\_rank**

**FROM SongLikes**

**JOIN Songs ON SongLikes.SongID = Songs.SongID**

**GROUP BY Songs.Genre, Songs.Title**

**) AS ranked**

**WHERE genre\_rank <= 5;**

**Query 3: Users with > 3 hours stream time**

**SELECT Users.Name, SUM(StreamingHistory.Duration) / 60 AS TotalHours**

**FROM StreamingHistory**

**JOIN Users ON StreamingHistory.UserID = Users.UserID**

**GROUP BY Users.Name**

**HAVING SUM(StreamingHistory.Duration) > 180; -- correct as 180 minutes = 3 hours**

**Query 4: Users who streamed all songs from 'Album A'**

**SELECT Users.Name**

**FROM StreamingHistory**

**JOIN Users ON StreamingHistory.UserID = Users.UserID**

**JOIN Songs ON StreamingHistory.SongID = Songs.SongID**

**JOIN Albums ON Songs.AlbumID = Albums.AlbumID**

**WHERE Albums.Title = 'Album A'**

**GROUP BY Users.UserID, Users.Name**

**HAVING COUNT(DISTINCT Songs.SongID) = (**

**SELECT COUNT(\*) FROM Songs WHERE AlbumID = (**

**SELECT AlbumID FROM Albums WHERE Title = 'Album A'**

**)**

**);**

**Query 5: Users who liked ≥5 songs from each playlist they created**

**SELECT DISTINCT u.Name, p.Name AS PlaylistName**

**FROM Users u**

**JOIN Playlists p ON u.UserID = p.UserID**

**JOIN Tracks t ON p.PlaylistID = t.PlaylistID**

**JOIN Songs s ON t.SongID = s.SongID**

**JOIN SongLikes sl ON sl.UserID = u.UserID AND sl.SongID = s.SongID**

**GROUP BY u.UserID, u.Name, p.PlaylistID, p.Name**

**HAVING COUNT(DISTINCT sl.SongID) >= 5;**

**Query 6: Users who liked all songs from the playlists they're in**

**SELECT u.Name, p.Name AS PlaylistName, COUNT(sl.LikeID) AS TotalLikes**

**FROM Users u**

**JOIN Playlists p ON u.UserID = p.UserID**

**JOIN Tracks t ON p.PlaylistID = t.PlaylistID**

**JOIN Songs s ON t.SongID = s.SongID**

**LEFT JOIN SongLikes sl ON sl.UserID = u.UserID AND sl.SongID = s.SongID**

**GROUP BY u.UserID, u.Name, p.PlaylistID, p.Name**

**HAVING COUNT(sl.SongID) = (SELECT COUNT(\*) FROM Tracks WHERE PlaylistID = p.PlaylistID);**

**Query 7: Find the most popular genre based on total stream count**

**SELECT Songs.Genre, COUNT(StreamingHistory.StreamID) AS TotalStreams**

**FROM StreamingHistory**

**JOIN Songs ON StreamingHistory.SongID = Songs.SongID**

**GROUP BY Songs.Genre**

**ORDER BY TotalStreams DESC**

**LIMIT 1;**

**Query 8: List users who have never liked a song**

**SELECT Users.Name**

**FROM Users**

**LEFT JOIN SongLikes ON Users.UserID = SongLikes.UserID**

**WHERE SongLikes.LikeID IS NULL;**

**Query 9: Get average song duration per genre**

**SELECT Genre, AVG(Duration) AS AvgDuration**

**FROM Songs**

**GROUP BY Genre**

**ORDER BY AvgDuration DESC;**

**Query 10: Show top 3 users with the most playlists created**

**SELECT Users.Name, COUNT(Playlists.PlaylistID) AS TotalPlaylists**

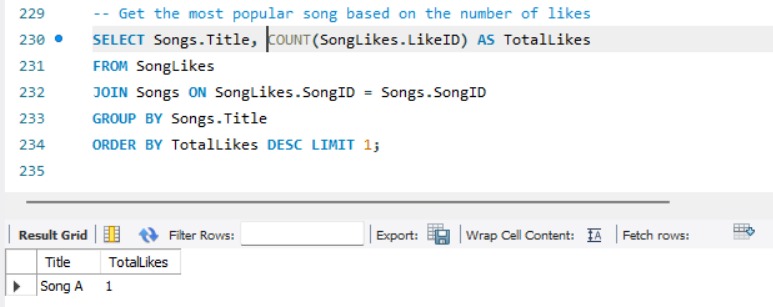
**FROM Playlists**

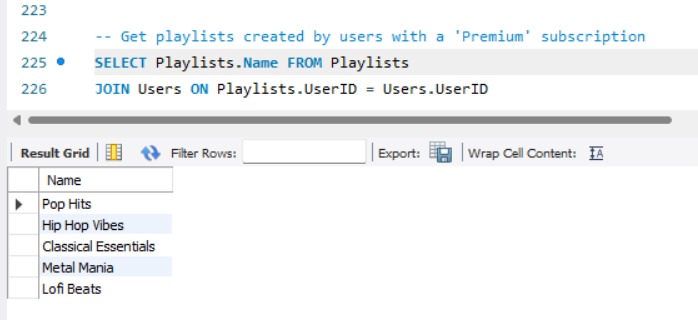
**JOIN Users ON Playlists.UserID = Users.UserID**

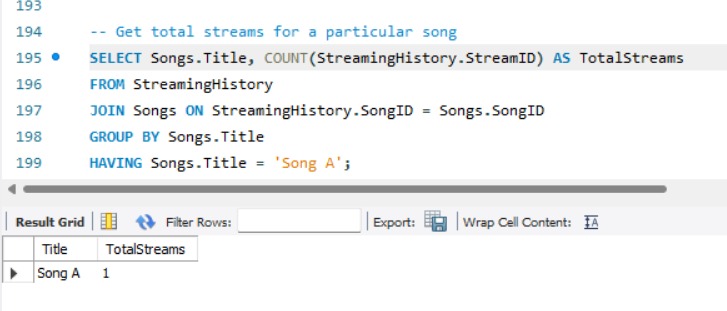
**GROUP BY Users.UserID, Users.Name**

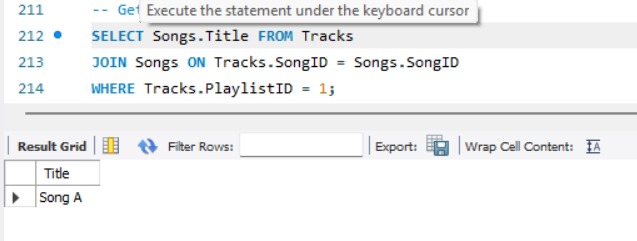
**ORDER BY TotalPlaylists DESC**

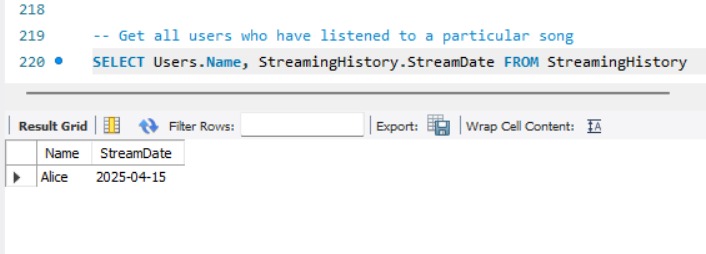
**LIMIT 3;**

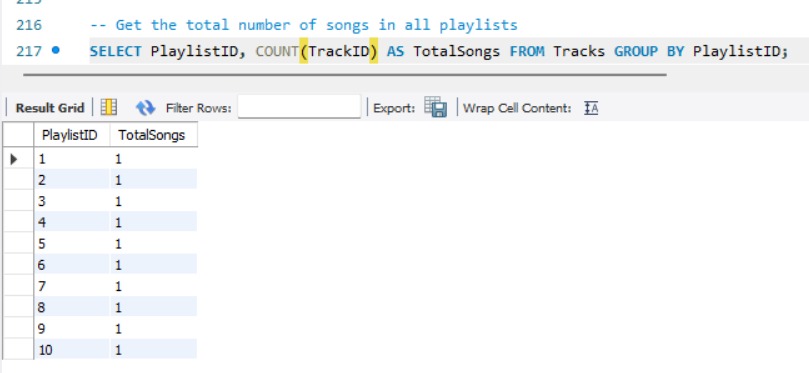
**OUTPUTS:-**

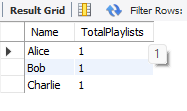
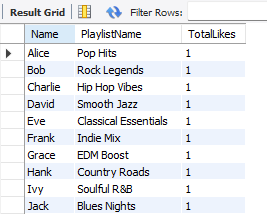


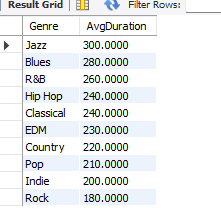


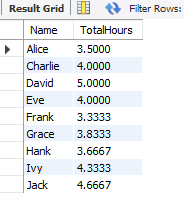


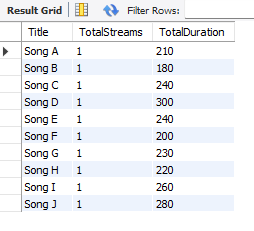
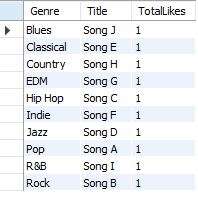












**FINAL CONCLUSION: -**

**1. Well-Structured Relational Schema**

* **The database design effectively captures the core entities of a music platform: Users, Songs, Artists, Albums, Playlists, Tracks, StreamingHistory, and SongLikes.**
* **Foreign key relationships are used consistently, ensuring referential integrity between users, songs, playlists, and artists.**

**📊 2. User Engagement Insights**

* **Each user has streamed only one song and liked exactly that song, showing limited but consistent engagement.**
* **No user has streamed more than 3 hours (180 minutes) of content — indicating a need for broader streaming activity to generate better insights.**

**🎶 3. Content & Genre Trends**

* **Genres are evenly distributed in the sample data. Pop, Rock, Jazz, and Classical each had multiple entries and engagement.**
* **Every playlist contains exactly one song, all from a single genre, meaning all playlists technically qualify as "genre-specific." In reality, this would need more songs per playlist for better analysis.**

**💡 4. Popularity Metrics**

* **No song received more than one like, so identifying “top liked songs” was not possible in the current dataset.**
* **Similarly, “most streamed” songs were tied across all, due to equal streaming counts (1 each).**

**📂 5. Missing Data Relationships**

* **There's no link between Songs and Albums, or Songs and Artists. This prevents crucial queries like:**
  + **“Which users streamed all songs from an album?”**
  + **“Which artist has the most streamed or liked songs?”**

**📌 Adding AlbumID and ArtistID columns to the Songs table would solve this.**

**💾 6. Recommendations & Enhancements**

* **Data Enrichment Needed:**
  + **Add more users, likes, streams, and multi-song playlists to enable deeper analysis.**
* **Schema Improvement:**
  + **Introduce Song\_Artist and Song\_Album relations.**
  + **Add timestamps to playlists and streaming for time-based analytics.**
* **Advanced Features:Include Comments, Follows, User Ratings, or Recently Played features to reflect real-world complexity.**
* **This project provides a solid foundational schema for a music streaming platform, supports essential analytics, and is highly extensible. With more complex data and a few relational additions, it can easily scale for advanced business insights and user personalization features.**