LemOn: A Database Project Report

Zechariah Frierson

1 Introduction

1.1 Purpose

This database stores data for an unimplemented music streaming service called LemOn. Its purpose is to store all user, artist, and song information, all interactions within the application (e.g., comments, likes, views), and all sorting methods for songs (e.g., albums, playlists).

1.2 Description

The database stores data on songs written by artists and can be sorted by users and artists into playlists and albums. Users will also be able to view, like, and leave comments on songs. Comments contain a body of text and can be liked, disliked, and replied to. Each song has a name and includes information about its duration, genre, likes and dislikes, which album it belongs to (if any), and which artist (or artists) wrote the song. Each user has a username, email, and password. Other information about users is also stored, including their country and birthdate. All artists must be users, but not all users are artists. Artists will have an artist name and a founding date, and the database also keeps track of each artist's total likes and views. Playlists can only be created by users, and have a name, an optional description, and also store information about the number of songs in the playlist, the total duration of the playlist, the date of playlist creation, and whether the playlist is public or private. Albums must be created by artists and contain the same information as playlists (minus the description and publicity), along with the album’s genre.

1.3 Potential Users

Anyone who enjoys listening to music; likely a younger audience.

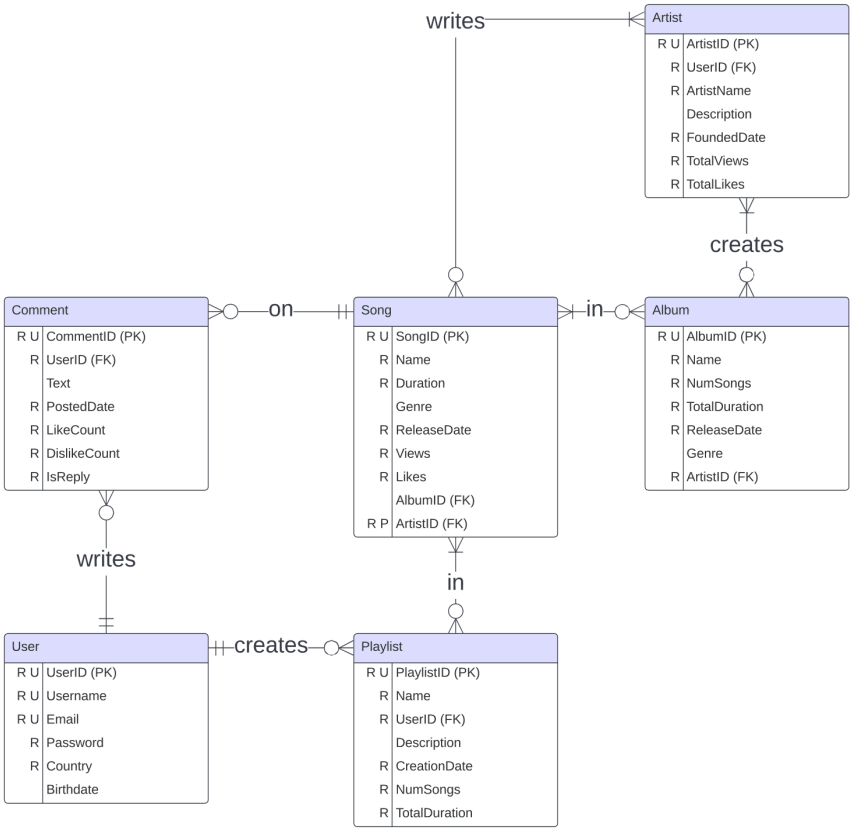
1.4 Sample User Questions

What songs are on this playlist? Which artist wrote this song? Which songs did this artist write? What genre does this song belong to? What genre(s) does this artist typically write? How many

minutes long is this song? How many views does this song have? How many views does this artist have?

2 Database Analysis

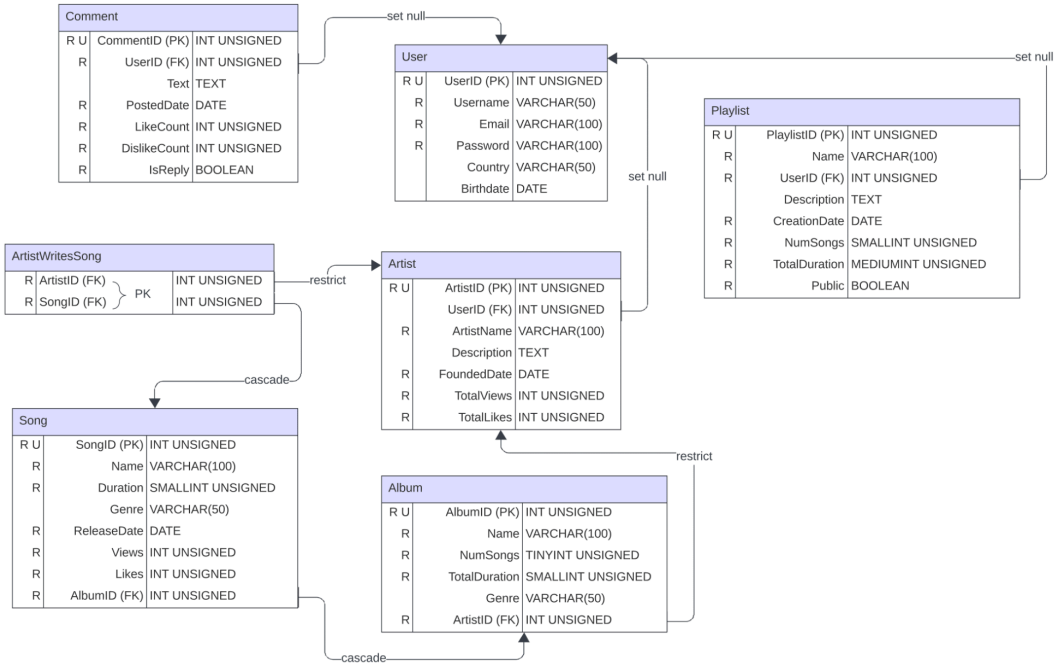
The ER diagram contains all strong entities in the database, labels the primary and foreign keys, and establishes the cardinality for all attributes and relationships. All relationships are named, and although there are multiple many-to-many relationships there are no intermediate tables, since they were implemented in the logical design step.



Although the UserID attribute of the Artist entity was initially labeled as required in this diagram, the cardinality was removed during schema implementation, as I wanted to be able to delete user instances without deleting artists. I also changed the User table later to have only UserID as a unique attribute to ensure Boyce-Codd Normal Form. During the logical design step, I implemented the Artist-Writes-Song intermediate table to account for the many-to-many relationship between the Artist and Song entities, and I implemented the Song-In-Album intermediate table during the schema creation step.

3 Database Logical Design

The table diagram contains all strong entities from the ER diagram as tables, and implements one of two final version intermediate tables, ArtistWritesSong, to properly implement the many-to-many relationship between the Artist and Song tables and allow for multiple artists to be associated with one song, and multiple songs to be associated with one artist. The table diagram also includes referential integrity actions on delete between foreign keys and their respective reference tables.



There were a variety of foreign keys that could have had many different referential integrity actions, but in the end, I made my decisions based on a few overarching rules that I created for the database. Since the hypothetical app associated with my database is a music streaming app, I wanted to be sure that, even if users were deleted, other users could still listen to all songs uploaded on the app unless the songs themselves were deleted. To implement this, I used restrict actions on every foreign key leading back to the "Artist" table, making it virtually impossible to delete artists themselves (and subsequently the songs associated with artists), even if the artist's original user account was deleted. I also placed "set null' actions on several relationships, such as the "UserID" foreign key in the "Comment" table, allowing comments to stay posted even if the original user was deleted.

4 SQL Documentation

4.1 Procedures

- GetPlaylistSongs - takes a playlist ID and returns the songs in the playlist.
- GetSongArtist - takes a song ID and returns the name of the artist who wrote the song.
- GetArtistSongs - takes an artist ID and returns the songs written by the artist.
- GetSongGenre - takes a song ID and returns the genre of the song.
- FindArtistGenre - takes an artist ID and returns the most common genre out of all of the songs written by that artist.
- GetSongDurationMinutes - takes a song ID and returns the duration of the song in minutes.
- GetSongViews - takes a song ID and returns the number of views on the song.
- GetArtistViews - takes an artist ID and returns the total number of views the artist has across all of their songs.
- GetTopArtist - takes no arguments and returns the artist's name with the most total views.
- GetTopArtistByGenre - takes a genre name and returns the artist's name who has the most views out of all artists on a song of the given genre.
- FindLatestCommentsByGenre - takes a genre name and returns the comment ID of the top 10 most recent comments on songs of the given genre.

4.2 Triggers

- UpdatePlaylistOnDeleteUser - before a user is deleted, checks whether the playlists associated with that user are public or private. If a playlist is private, the playlist is deleted when the user is deleted. If the playlist is public, only the UserID of the playlist is deleted, and the rest of the playlist information is kept in the database.

5 Data Description

All data currently implemented in the database is synthetic and was generated with LLMs. I used ChatGPT to generate an initial 5-10 rows per table[1], and then I used GitHub Copilot (with the prompt: "Add 45 more lines") to complete the data up to 50 rows per table. I went through a few of the data generation scripts and adjusted some data slightly for more realistic query results.

[1] Link to the ChatGPT transcript for the generated data:
https://chat.openai.com/share/905c911e-a5d8-41ce-9a56-5c7f4d8d36b0

6 Conclusion

Over the course of this project, I have learned how to successfully implement a real database, using Amazon AWS RDS, MySQL Workbench, and Visual Studio Code in tandem. I have successfully built several realistic procedures, one of many possible unique triggers, and set up a well-designed backend for a possible future project. While the database would be usable in the real world with a front end for interaction, there are a few real-world implications to consider. Since user data is stored directly in the database, unencrypted, anyone who can access the database could access users' private information, which is a large security and privacy concern, especially with passwords. If the database were implemented with real data and especially if the database stored real music files in the Song table, there could be storage limits or slow access time with the large volume and size of real-world music data. Finally, for a more social implication, the unmoderated forum-style interactions of comments, likes, and views could lead to issues concerning user-generated content like cyberbullying, explicit language, and disclosure of private information. If I were to continue work on this project, I might implement music file storage (possibly through the cloud); build an application or GUI to access the database, listen to music, and leave comments and interactions; remove some data redundancy that could be accessed via queries, such as Album Genre, Artist Total Views, etc.; and add real data and more queries for every possible interaction with the database. Overall, this project has been extremely helpful for furthering my knowledge of SQL and databases in general, and is the perfect starting point for a side project later down the line if I remain interested in this project.