Music Domain

Team Pick A Noun (Group 18)
Student 1 ID: Anthony Salvati, ams6995
Student 2 ID: Michael Biedermann, mjb7036
Student 3 ID: Eli Lurie, ehl8022
Student 4 ID: Jack Sebben, jas8726
Student 5 ID: Jaden Seaton, jis6849

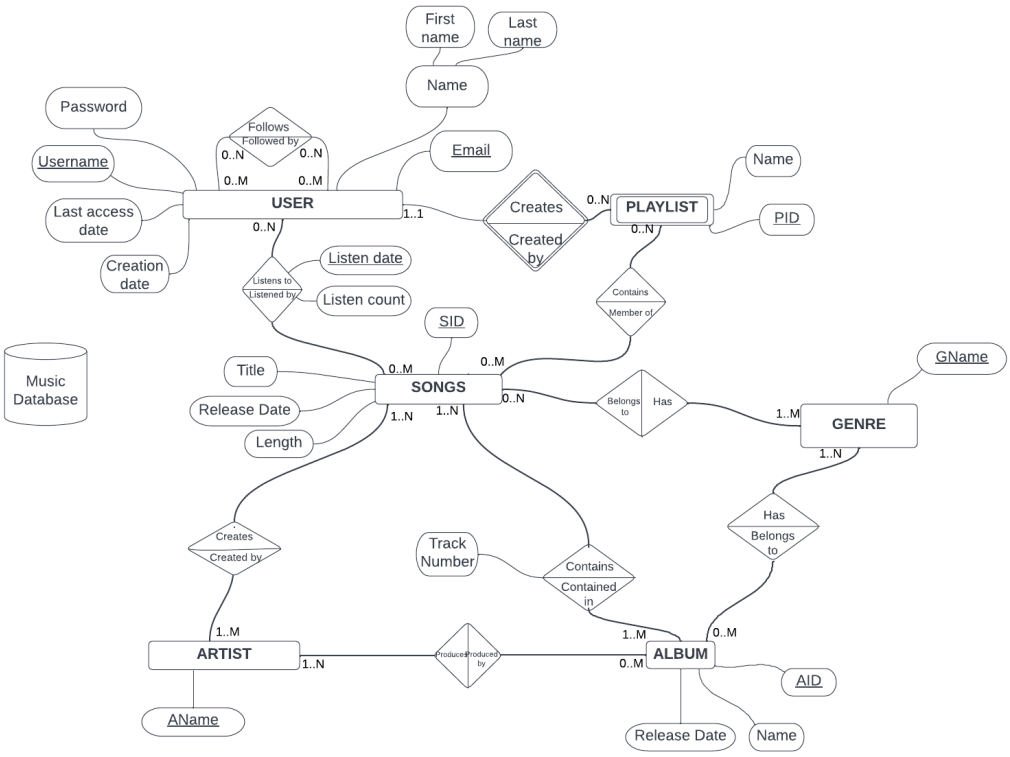
November 18, 2022

1 Introduction

The domain for which our database and manager will be built for is the music domain. Our plan for this project will be to use a Java driven console application using the command line. The application will use JSON objects to communicate with the server with SQL.

2 Design

2.1 Conceptual Model



In creating the EER diagram, we had to consider what unique identifiers each entity would have, their relationships to other entities, and what those relationships entailed. For instance, a user has the identifier of a unique username and relates to a playlist by creating them. With the music domain, songs are the primary focus, as shown by every other entity having a relationship to them.

2.2 Reduction to tables

Users(Username, Email, Password, First Name, Last Name, Last Access Date, Creation Date)
Artist(AName)
Song(SID, Title, Release Date, Length)
Playlist(PID, Username, Name)
Genre(GName)
Album(AID, Release Date, Name)

Genre_Song(*SID*, *GName*)
User_Song(*SID*, *Username, Listen Date*, *Listen Count*)
Song_Artist(*SID, AName)*
Song_Playlist(*SID*, *PID*)
Album_Song(*SID*, *AID*, Track Number)
Album_Artist(*AID, AName*)
Genre_Album(*AID*, *GName*)
User_User(*Username*, *Username*)

The User entity has been reduced to a table with the Username as a primary key, as this is the least likely to change. The same goes for:

- Artist and their name,
- album and AID (album ID)
- song and SID (song ID)
- genre and GName
- playlist and PID (playlist ID)

Playlists also have a foreign key for the user, which is unique, as shown in the reduction to tables above. The genre-album relationship was reduced to a table that included the AID and Gname, as those were the key factors in the relationship. Similarly, albums having songs became a table with a SID, AID, and track number, albums by artists are related by the AID and AName, songs are related to users by the SID and Username, songs are found on playlists by SID and PID, and songs can be related to artists by SID and AName.

2.3 Data Requirements/Constraints

The most important attributes of entities from the above diagram are the unique email address for each user (since username may change over time),

the playlist ID (PID), song ID (SID), genre name (GName) album ID (AID), and artist name (AName). It is critical that each of these are unique, as they are they key identifiers for each of their respective entities. Another constraint of this relational model is that, when created, all songs must to have an album and vice versa. In the same vein, all artists must have at least one song, and all songs must have at least one artist.

2.4 Sample instance data

User

Username	Email	Password	First name	Last name	Last access date	Creation date
jadens12	jis6849@rit.edu	Awesome123	Jaden	Seaton	9/15/2022	9/15/2022
eli_lurie	ehl8022@rit.edu	hello_there	Eli	Lurie	9/2/2022	1/10/2019
michaelthebest	mjb7036@rit.edu	super_secure	Michael	Biedermann	9/14/2022	12/12/2020
anotherUser	bobjoe@gmail.c	password12345	Bob	Joe	3/1/2021	3/2/2021
hellooooooo	thisIsMyEmail@	yeeeeeee	Eli	Lurie	1/1/2011	10/19/2020

Artist

AName
thebestartist
theworstartist
justOKartist
Drake
mediumartist

Album

AID	Release Date	Name
1	1/20/2019	bestalbum
2	6/29/2022	worstalbum
3	1/1/2000	justOKalbum
4	5/23/2005	mediumalbum
5	7/9/2020	drakealbum

Songs

SID	Title	Release Date	Length
1	This is a Song	1/20/2019	3:42
2	This is not a Sor	6/29/2022	0:01
3	Maybe a song	1/1/2000	2:56
4	Long Song	5/23/2005	8:59
5	Short Song	7/9/2020	0:58

Playlist

PID	Username	PName
1	anotherUser	Yee
2	eli_lurie	Playlist #92
3	eli_lurie	Pop music
4	michaeltheBest	Rap music
5	jadens12	Yuh

Genre

GName
Pop
Rock
Indie
Rap
Soul

Genre-Song

SID	GName
1	Pop
2	Pop
3	Rap
4	Indie
5	Rock

Genre-Album

AID	GName
1	Soul
2	Pop
3	Rock
4	Indie
5	Rap

Album-Artist

AName	AID
mediumartist	1
Drake	2
Drake	3
justOKartist	4
thebestartist	5

Album-Song

SID	AID	Track Number
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1

User-Song

Username	SID
jadens12	1
eli_lurie	2
eli_lurie	1
anotherUser	4
anotherUser	2

Song-Playlist

SID	PID
1	3
1	2
4	4
3	5
2	5

Song-Artist

SID	AName
1	thebestartist
2	theworstartist
3	justOKartist
4	Drake
5	mediumartist

3 Implementation

Sample SQL statements used to create tables:

CREATE TABLE album (
aid INTEGER PRIMARY KEY,
release_date DATE NOT NULL,
name VARCHAR(50) NOT NULL)

CREATE TABLE song_playlist (
sid INTEGER REFERENCES song,
pid INTEGER REFERENCES playlist,
PRIMARY KEY (sid, pid))

Sample SQL statements to populate data:

“INSERT INTO genre VALUES(‘ ” + genre + “ ’)”

“INSERT INTO artist VALUES(‘ ” + artist + “ ’)”

“INSERT INTO album_artist VALUES(” + str(aid) + “, ‘ ”
+ currentArtist + “ ’)”

“INSERT INTO album VALUES(” + str(aid) + ”, ‘ ” + str(currentDate)
+ ” ’, ‘ ” + song + ” ’)”

“INSERT INTO song VALUES(” + str(sid) + ”, ‘ ” + song + ” ’, ‘ ”
+ str(currentDate) + ” ’, ” + str(randLength) + ”)”

“INSERT INTO album_song VALUES(” + str(sid) + ”, ” + str(aid)
+ ”, ” + str(trackNum) + ”)”

“INSERT INTO song_artist VALUES(” + str(sid) + ”, ‘ ”
+ currentArtist + ” ’)”

“INSERT INTO genre_song VALUES(” + str(sid) + ”, ‘ ”
+ genre + ” ’)”

“INSERT INTO genre_album VALUES(” + str(aid) + ”, ‘ ”
+ genre + ” ’)”

The data was loaded into the database using a Python script reading from a csv file of a subset of the million song dataset. The file contained song names, artist names, and release years. First, genres were added from a separate list of genres. Artists were added using the artist names from the file. Albums were created by using song names as album names, and each album contains a random number of songs from 1 to 12. Each album's date was taken from the song's date and random months and days were used; each album was assigned one or more random genres from a list of genres. Album IDs (aid) were generated sequentially. Songs were added using the song name from the file, and song IDs (sid) were generated sequentially. Song_artist relationships were added using the artist name corresponding to the song name in the file. Album_song relationships were added using the generated aid and sid numbers and songs were added to albums with sequential track numbers until the album reached its designated number of songs. About 6000 songs were added, about 900 artists were added, and about 1500 albums were added, along with many relationships between them.

Sample insert statements:

PreparedStatement pst = conn.prepareStatement(”INSERT INTO playlist VALUES (?, ?, ?)”);
pst.setInt(1, newID);
pst.setString(2, username);
pst.setString(3, playlistName);

PreparedStatement insertQuery = conn.prepareStatement(”INSERT INTO song_playlist (sid, pid) ” + ”SELECT ?, ? WHERE NOT EXISTS (SELECT * FROM song_playlist WHERE sid = ? and pid = ?)”);
insertQuery.setInt(1, sid);
insertQuery.setInt(2, collection.pid);
insertQuery.setInt(3, sid);
insertQuery.setInt(4, collection.pid);

```
PreparedStatement addFollow = conn.prepareStatement("INSERT INTO user_user
VALUES (?, ?)");
addFollow.setString(1, this.currentUsername);
addFollow.setString(2, friendEmail);

PreparedStatement pst = conn.prepareStatement("INSERT INTO users
VALUES (?, ?, ?, ?, ?, ?, ?, ?)");
pst.setString(1, email);
pst.setString(2, newUsername);
pst.setString(3, Integer.toString(hashPass));
pst.setString(4, name[0]);
pst.setString(5, name[1]);
pst.setDate(6, todayDate);
pst.setDate(7, todayDate);
pst.setString(8, saltValue);

PreparedStatement addListen = conn.prepareStatement("INSERT INTO user_song
VALUES (?, ?)");
addListen.setInt(1, sid);
addListen.setString(2, username);
```

4 Data Analysis

4.1 Hypothesis

We hypothesized that people who liked a particular genre would also be inclined to listen to another similar genre, such as metal listeners all listening to rock next most or hip hop to pop.

We ended up seeing that some genres are tied together, such as indie users liking pop and vice versa.

4.2 Data Preprocessing

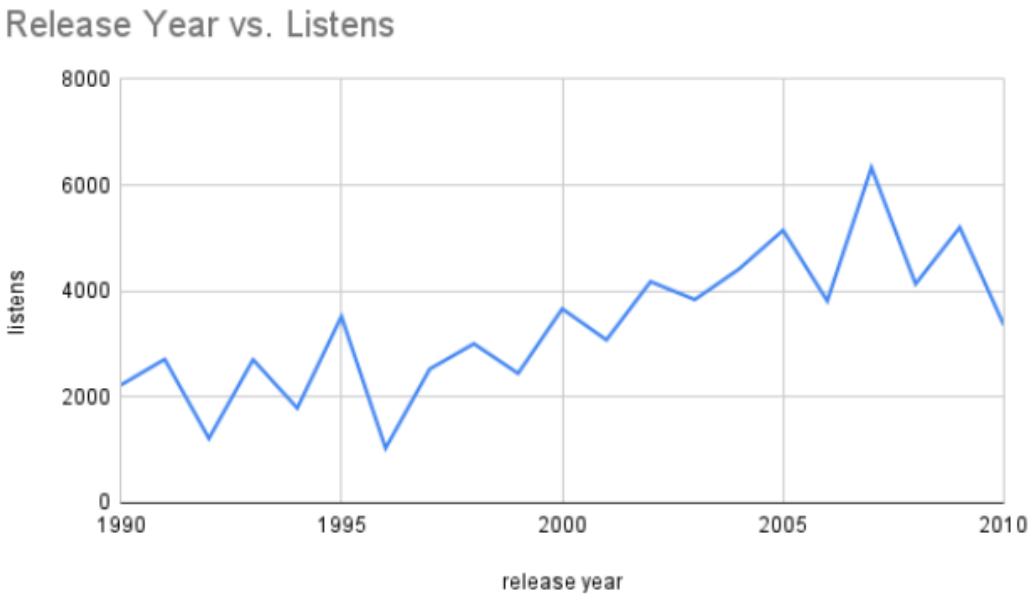
We did not do any data preprocessing aside from using queries to extract the data from the database because there were no issues with the data that needed to be fixed. The indices of our database for all of the tables were

B+-trees in order to maximize the efficiency of our queries. We used complex queries to extract the data, which are shown below.

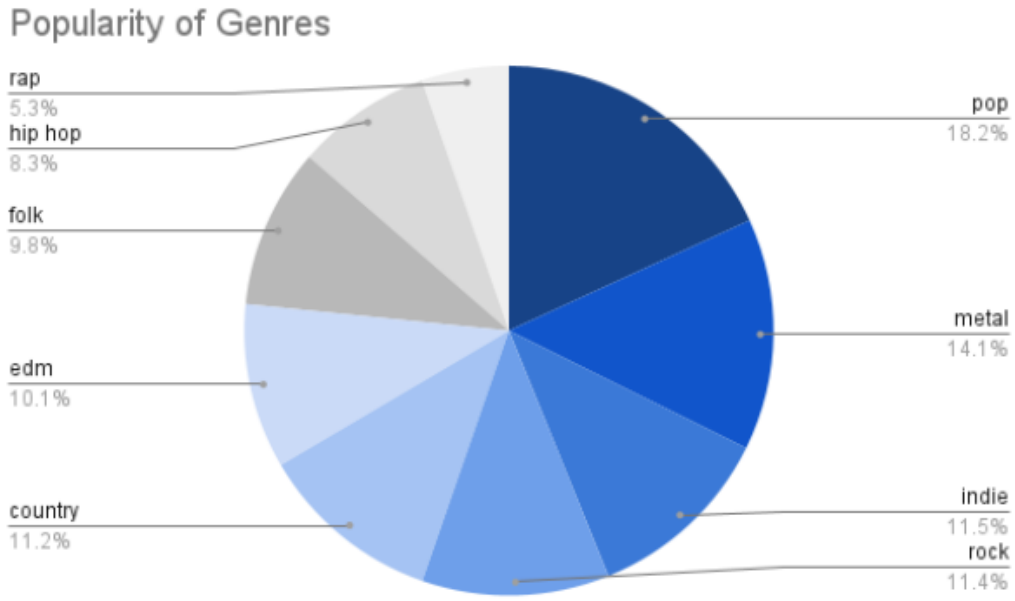
Used to extract the top 2 genres for each user:
WITH subquery AS (SELECT user_genre.name AS name, user_genre.listen_count AS listen_count, user_genre.username AS username, row_number() OVER (PARTITION BY username ORDER BY listen_count DESC) AS rank FROM (SELECT genre_song.genre_name AS name, SUM(user_song.listens) AS listen_count, user_song.username AS username FROM user_song, genre_song WHERE user_song.sid = genre_song.sid GROUP BY user_song.username, genre_song.genre_name) AS user_genre)
SELECT q1.name as genre1, q2.name as genre2, q1.username FROM subquery q1, subquery q2 WHERE q1.username = q2.username AND q1.rank = 1 AND q2.rank = 2

Used to extract song listening data: SELECT title, release_date, length, username, listen_date, listens, genre_name FROM song JOIN user_song ON song.sid = user_song.sid JOIN genre_song ON genre_song.sid = song.sid;

4.3 Data Analytics & Visualization



This graph represents the amount of listen per song release year.



This graph represents the popularity of genres based on song listens.

The relation between the most popular genre a listener enjoys and their second favorite genre.

4.4 Conclusions

From the graphs above, you can see that the most listened songs were between the release years of 2005 and 2008 and the least listened songs were between the release years of 1993 and 1994. The most popular genres in according to listen count is pop and our least popular genre is rap. Our genre recommendations are very accurate to the real world, in the way that indie is most similar to pop, country is most similar to folk, and metal is most similar to rock.

5 Lessons Learned

Writing the complex queries for phase 4 was challenging, especially the song recommendation query in our program. Which query has multiple subqueries and joins many tables from our database. This took many tries to get working correctly, but helped us to better understand how to write complex SQL queries.

Another issue faced in the project was deciding how to store the listen count for songs. We originally had a table where username and sid were the primary keys so we could only record whether a user had ever listened to a song, and weren't able to record how many times that song was listened to or the date the user listened to it. We changed the attributes of this table in phase 4 to have username, sid, and listen date to be the primary key, as well as including a listen count, which let us count every time a user listened to a song and what date that occurred on. We learned that being flexible with our design is important, as our previous design was inadequate, despite thinking originally that it would be sufficient.

6 Resources

We used complex queries to export data into Google Sheets and LucidChart in order to make the graphs shown above.

7 SQL Appendix

Extractx the top 2 genres for each user for data analysis:
WITH subquery AS (SELECT user genre.name AS name, user genre.listen count AS listen count, user genre.username AS username, row number() OVER (PARTITION BY username ORDER BY listen count DESC) AS rank FROM (SELECT genre song.genre name AS name, SUM(user song.listens) AS lis- ten count, user song.username AS username FROM user song, genre song WHERE user song.sid = genre song.sid GROUP BY user song.username, genre song.genre name) AS user genre) SELECT q1.name as genre1, q2.name as genre2, q1.username FROM sub- query q1, subquery q2 WHERE q1.username = q2.username AND q1.rank = 1 AND q2.rank = 2

Extracts song listening data for data analysis:
SELECT title, release date, length, username, listen date, listens, genre name FROM song JOIN user song ON song.sid = user song.sid JOIN genre song ON genre song.sid = song.sid;

Gets the 50 most listened to songs in the last month:
SELECT song.title, song_artist.artist_name, album.name, song.length, songs_listened.listen_count

FROM song, song_artist, album, album_song, (SELECT sid, SUM(listens) AS listen_count FROM user_song WHERE listen_date ¿ ? GROUP BY sid) songs_listened WHERE song.sid = songs_listened.sid AND song_artist.sid = song.sid AND album_song.sid = song.sid AND album.aid = album_song.aid

Gets the 50 most listened to songs among the user's friends:
SELECT song.title, song_artist.artist_name, album.name, song.length, songs_listened.listen_count FROM song, song_artist, album, album_song, (SELECT sid, SUM(listens) AS listen_count FROM user_song WHERE username IN (SELECT username2 FROM user_user WHERE username1 = ?) GROUP BY sid) songs_listened WHERE song.sid = songs_listened.sid AND song_artist.sid = song.sid AND album_song.sid = song.sid AND album.aid = album_song.aid" ORDER BY songs_listened.listen_count DESC, song.title ASC, song_artist.artist_name ASC LIMIT 50

Gets the top 5 genres of the current month:
SELECT genre_song.genre_name, SUM(songs_listened.listen_count) AS genre_listen_count FROM genre_song, (SELECT sid, SUM(listens) AS listen_count FROM user_song WHERE listen_date ¿ ? GROUP BY sid) songs_listened WHERE genre_song.sid = songs_listened.sid GROUP BY genre_song.genre_name ORDER BY genre_listen_count DESC LIMIT 5

Recommends 20 songs that have been listened to by users with similar top genres:
SELECT DISTINCT song.title AS title, song_artist.artist_name AS artist, album.name AS album, song.length AS length, random() FROM song_artist, album, album_song, song, user_song WHERE user_song.username IN (SELECT DISTINCT username FROM (SELECT user_genre.name AS name, user_genre.listen_count AS listen_count, user_genre.username AS username, row_number() OVER (PARTITION BY username ORDER BY listen_count DESC) AS rank FROM (SELECT genre_song.genre_name AS name, SUM(user_song.listens) AS listen_count, user_song.username AS username FROM user_song, genre_song WHERE user_song.sid = genre_song.sid AND user_song.username != ? GROUP BY user_song.username, genre_song.genre_name) AS user_genre) genre_ranks WHERE genre_ranks.rank ¡= 3 AND name IN (SELECT genre_song.genre_name FROM user_song, genre_song WHERE user_song.username = ? AND user_song.sid = genre_song.sid GROUP BY genre_song.genre_name ORDER BY SUM(user_song.listens) DESC LIMIT 3)) AND song.sid = user_song.sid AND song_artist.sid = song.sid

AND album_song.sid = song.sid AND album.aid = album_song.aid AND song.sid NOT IN (SELECT sid FROM user_song WHERE username = ?) ORDER BY random() LIMIT 20