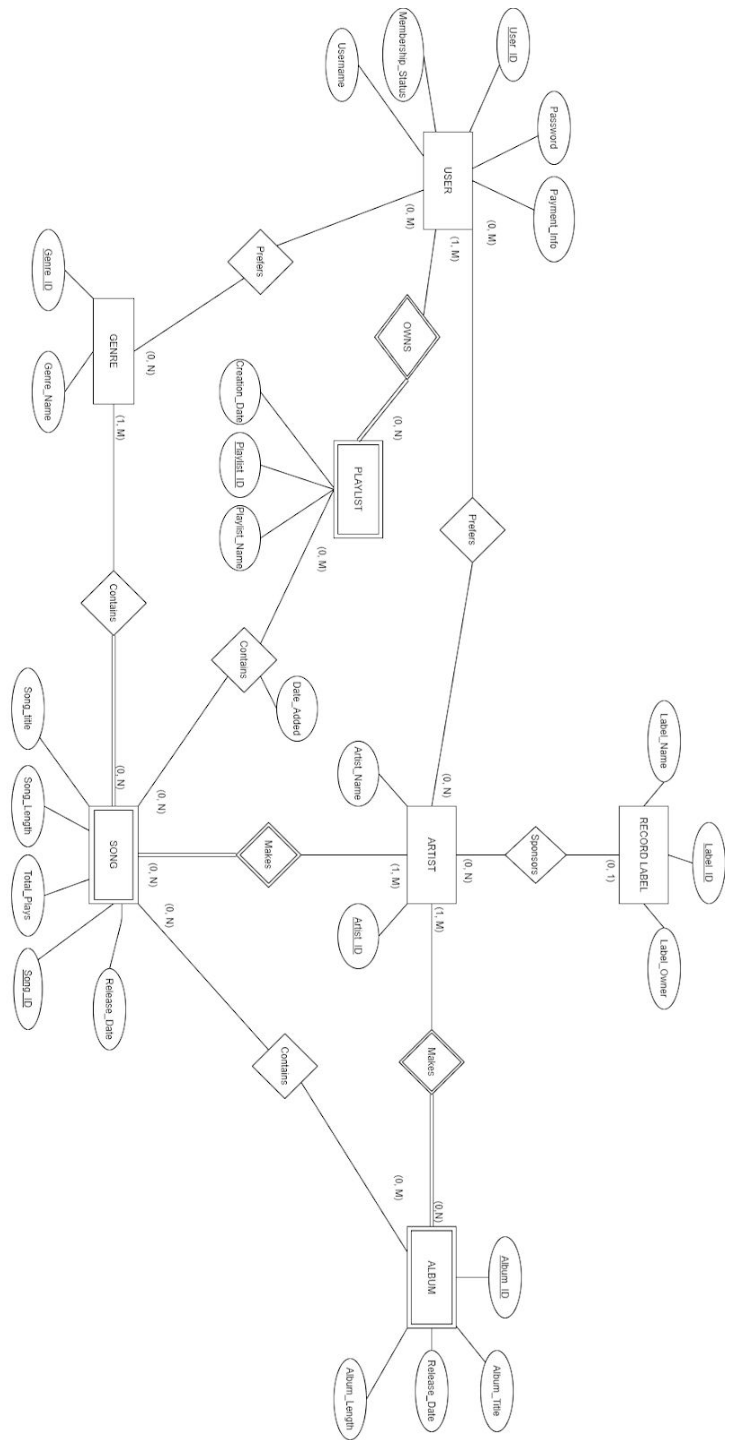
Andrew Montgomery, Chris Powell, Gary Yuen, Kolby Samson, Omed Mahshour
Team 10
March 16th, 2021
CSS 475

# Final Project Report

# Table of Contents

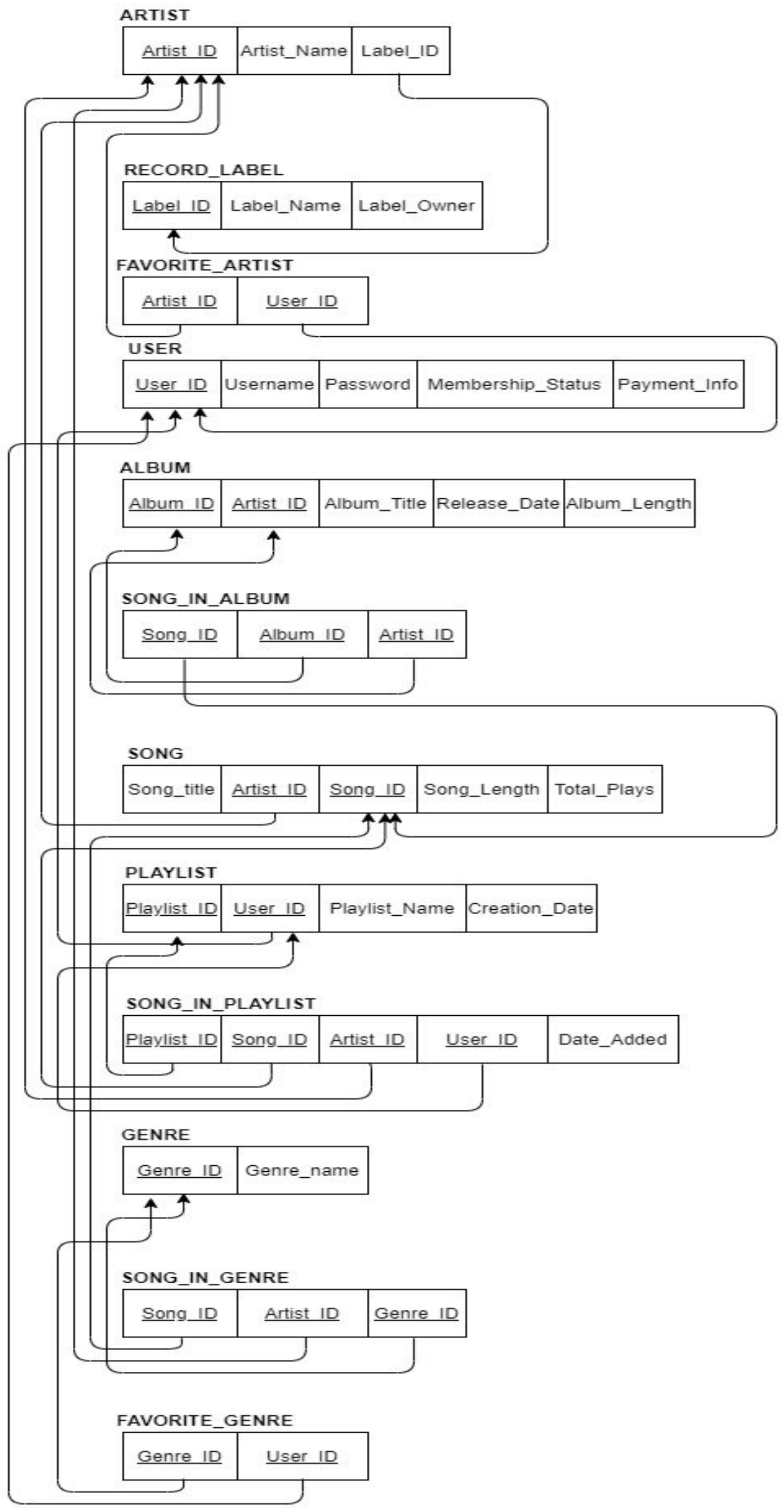
# Design Documentation

## Entity-Relationship Diagram



**(Please view ERD.png if this diagram is too small/you wish to view w/o rotating the doc)**

## Relational Data Model


ARTIST
Artist_ID | Artist_Name | Label_ID
RECORD_LABEL
Label_ID | Label_Name | Label_Owner
FAVORITE_ARTIST
Artist_ID | User_ID
USER
User_ID | Username | Password | Membership_Status | Payment_Info
ALBUM
Album_ID | Artist_ID | Album_Title | Release_Date | Album_Length
SONG_IN_ALBUM
Song_ID | Album_ID | Artist_ID
SONG
Song_title | Artist_ID | Song_ID | Song_Length | Total_Plays
PLAYLIST
Playlist_ID | User_ID | Playlist_Name | Creation_Date
SONG_IN_PLAYLIST
Playlist_ID | Song_ID | Artist_ID | User_ID | Date_Added
GENRE
Genre_ID | Genre_name
SONG_IN_GENRE
Song_ID | Artist_ID | Genre_ID
FAVORITE_GENRE
Genre_ID | User_ID

**(Please view RDM.png if the relational data model display is too small)**

## SQL Statements

### Table Creation Statements

```
.open MUSICSERVICE.db

--DROP TABLES USING:
drop table if exists RECORD_LABEL;
drop table if exists ARTIST;
drop table if exists USER;
drop table if exists FAVORITE_ARTIST;
drop table if exists SONG;
drop table if exists ALBUM;
drop table if exists SONG_IN_ALBUM;
drop table if exists PLAYLIST;
drop table if exists SONG_IN_PLAYLIST;
drop table if exists GENRE;
drop table if exists SONG_IN_GENRE;
drop table if exists FAVORITE_GENRE;

CREATE TABLE RECORD_LABEL (
     Label_ID INT NOT NULL,
     Label_Name VARCHAR(50) NOT NULL,
     Label_Owner VARCHAR(50),

     PRIMARY KEY (Label_ID)
);

CREATE TABLE ARTIST (
     Artist_ID INT NOT NULL,
     Artist_Name VARCHAR(30),
     Label_ID INT,

     FOREIGN KEY (Label_ID) REFERENCES RECORD_LABEL (Label_ID)
ON UPDATE CASCADE ON DELETE SET NULL,

     PRIMARY KEY (Artist_ID),
     UNIQUE(Artist_Name)
);
```

```
CREATE TABLE USER (
     User_ID INT NOT NULL,
     Username VARCHAR(30) NOT NULL,
     Password VARCHAR(30) NOT NULL,
     Membership_Status BOOLEAN NOT NULL,
     Payment_Info VARCHAR(50),

     PRIMARY KEY (User_ID)
);

CREATE TABLE FAVORITE_ARTIST (
     Artist_ID INT NOT NULL,
     User_ID INT NOT NULL,

     FOREIGN KEY (Artist_ID) REFERENCES ARTIST (Artist_ID) ON
UPDATE CASCADE ON DELETE CASCADE,

     FOREIGN KEY (User_ID) REFERENCES USER (User_ID) ON UPDATE
CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Artist_ID, User_ID)
);

CREATE TABLE SONG (
     Song_ID INT NOT NULL,
     Song_Title VARCHAR(30) NOT NULL,
     Artist_ID INT NOT NULL,
     Song_Length TIME NOT NULL ,
     Total_Plays INT CHECK(Total_Plays >= 0),

     FOREIGN KEY (Artist_ID) REFERENCES ARTIST (Artist_ID) ON
UPDATE CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Song_ID, Artist_ID)
);
```

```
CREATE TABLE ALBUM (
     Album_ID INT NOT NULL,
     Artist_ID INT NOT NULL,
     Album_Title VARCHAR(30) NOT NULL,
     Release_Date DATE NOT NULL,
     Album_Length TIME NOT NULL,

     FOREIGN KEY (Artist_ID) REFERENCES ARTIST (Artist_ID) ON
UPDATE CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Album_ID, Artist_ID)
);

CREATE TABLE SONG_IN_ALBUM (
     Song_ID INT NOT NULL,
     Album_ID INT NOT NULL,
     Artist_ID INT NOT NULL,

     FOREIGN KEY (Song_ID, Artist_ID) REFERENCES SONG (Song_ID,
Artist_ID) ON UPDATE CASCADE ON DELETE CASCADE,

     FOREIGN KEY (Album_ID, Artist_ID) REFERENCES ALBUM
(Album_ID, Artist_ID) ON UPDATE CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Song_ID, Album_ID, Artist_ID)
);

CREATE TABLE PLAYLIST (
     Playlist_ID INT NOT NULL,
     User_ID INT NOT NULL,
     Playlist_Name VARCHAR(30),
     Creation_Date DATE NOT NULL,

     FOREIGN KEY (User_ID) REFERENCES USER (User_ID) ON UPDATE
CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Playlist_ID, User_ID)
);
```

```
CREATE TABLE SONG_IN_PLAYLIST (
     Playlist_ID INT NOT NULL,
     Song_ID INT NOT NULL,
     Artist_ID INT NOT NULL,
     User_ID INT NOT NULL,
     Date_Added DATE NOT NULL,

     FOREIGN KEY (Playlist_ID, User_ID) REFERENCES PLAYLIST
(Playlist_ID, User_ID) ON UPDATE CASCADE ON DELETE CASCADE,

     FOREIGN KEY (Song_ID, Artist_ID) REFERENCES SONG (Song_ID,
Artist_ID) ON UPDATE CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Playlist_ID, Song_ID, Artist_ID, User_ID)
);

CREATE TABLE GENRE (
     Genre_ID INT NOT NULL,
     Genre_Name VARCHAR(30) NOT NULL,

     PRIMARY KEY (Genre_ID),
     UNIQUE(Genre_Name)
);

CREATE TABLE SONG_IN_GENRE (
     Song_ID INT NOT NULL,
     Artist_ID INT NOT NULL,
     Genre_ID INT NOT NULL,

     FOREIGN KEY (Song_ID, Artist_ID) REFERENCES SONG (Song_ID,
Artist_ID) ON UPDATE CASCADE ON DELETE CASCADE,

     FOREIGN KEY (Genre_ID) REFERENCES GENRE (Genre_ID) ON
UPDATE CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Song_ID, Artist_ID, Genre_ID)
);

CREATE TABLE FAVORITE_GENRE (
     Genre_ID INT NOT NULL,
     User_ID INT NOT NULL,

     FOREIGN KEY (Genre_ID) REFERENCES GENRE (Genre_ID) ON
UPDATE CASCADE ON DELETE CASCADE,

     FOREIGN KEY (User_ID) REFERENCES USER (User_ID) ON UPDATE
CASCADE ON DELETE CASCADE,

     PRIMARY KEY (Genre_ID, User_ID)
);


.mode columns
.headers on
PRAGMA foreign_keys=1;
```

## Table Queries / Operations

**(Please view Queries.txt for the SQL queries in raw text format [in the same order as this table])**

| **SQL Statement** | **Purpose<br>(General Functionality and Specific Query Usage)** |
|---|---|
| SELECT date(Release_date) as Release_date, Album_title, Artist_id<br>FROM ALBUM<br>ORDER BY Release_date; | Find all albums in the database ordered by when they were released<br><br>(List all albums sorted by release date) |
| SELECT s.Song_Title<br>FROM SONG s, ARTIST art, ALBUM alb, SONG_IN_ALBUM sia<br>WHERE (sia.Song_ID = s.Song_ID) AND (sia.Album_ID = alb.Album_ID) AND (s.Artist_ID = | Find Recent Music For a Specific Artist<br><br>(Select all songs by The Weeknd Released After January 21st, 2021) |

| | |
|---|---|
| art.Artist_ID) AND (alb.Release_Date > julianday('2021-01-01')) AND (art.Artist_Name = 'The Weeknd'); | |
| SELECT Label_Name, Artist_Name<br>FROM ARTIST a, RECORD_LABEL r<br>WHERE a.Label_ID = r.Label_ID<br>ORDER BY Artist_Name; | Find All Artists associated with a Record Label<br><br>(List all artists for each record label sorted by artist name) |
| SELECT Song_Title<br>FROM SONG s<br> JOIN SONG_IN_GENRE sig ON s.Song_ID = sig.Song_ID<br> JOIN GENRE g ON sig.Genre_ID = g.Genre_ID<br>WHERE g.Genre_Name = 'Hip-Hop'; | Retrieve all songs in a specific genre<br><br>(List all songs in the “Hip-Hop” genre) |
| SELECT Album_Title<br>FROM ALBUM alb<br>JOIN SONG_IN_ALBUM sia ON alb.Album_ID = sia.Album_ID AND<br>alb.Artist_ID = sia.Artist_ID<br>JOIN SONG s ON s.Song_ID = sia.Song_ID AND s.Artist_ID = sia.Artist_ID<br>JOIN ARTIST art ON s.Artist_ID = art.Artist_ID<br>WHERE art.Artist_Name = 'Migos'<br>GROUP BY alb.Album_ID<br>HAVING SUM(s.total_plays) > 50000; | Find popular albums for a specific artist<br><br>(Select all albums from the artist “Migos” that have more than 50,000 plays) |
| SELECT SUM(s.Total_Plays)<br>FROM SONG s<br>JOIN Artist a ON a.Artist_ID = s.Artist_ID<br>WHERE Artist_Name = 'The Weeknd'; | Calculate the total amount of plays for an artist (to calculate payout and/or popularity)<br><br>(Select the sum of total plays across all songs from The Weeknd) |
| SELECT Password<br>FROM USER<br>WHERE Username = 'Wynn'; | Retrieve the password for a user logging in for authentication<br><br>(Select password for User with |

| | |
|---|---|
| | username Wynn) |
| UPDATE USER<br>SET Password = 'DJN2kiHkl'<br>WHERE Username = 'Fischer'; | Update the user’s password to a new password (password change/reset)<br><br>(Change password of user Fischer to DJN2kiHkl) |
| SELECT Genre_Name<br>FROM GENRE g<br>    JOIN FAVORITE_GENRE fg<br>        ON g.Genre_ID = fg.Genre_ID<br>GROUP BY fg.Genre_ID<br>ORDER BY COUNT(fg.Genre_ID)<br>DESC<br>LIMIT 3; | Find the most popular genres across all users<br><br>(Select the top 3 most popular genres using all user’s favorite genres) |
| SELECT s.Song_Title, alb.Album_Title, date(alb.Release_Date)<br>FROM SONG s<br>    JOIN SONG_IN_ALBUM sia ON sia.Song_ID = s.Song_ID<br>    JOIN ALBUM alb ON sia.Album_ID = alb.Album_ID<br>    JOIN FAVORITE_ARTIST fa ON fa.Artist_ID = sia.Artist_ID<br>    JOIN USER u ON u.User_ID = fa.User_ID<br>WHERE u.Username = 'Sears' AND alb.Release_Date >= julianday(date('now', '-7 day')); | Get all newly-released songs for a user’s favorite artists (recommend new music)<br><br>(Select all songs created by the favorite artists of the user Sears that were released in the last week) |
| SELECT DISTINCT art.Artist_Name<br>FROM ARTIST art<br>    JOIN FAVORITE_ARTIST fa ON art.Artist_ID = fa.Artist_ID<br>    WHERE fa.User_ID IN (<br>        SELECT u.User_ID<br>        FROM USER u<br>        JOIN FAVORITE_ARTIST fa ON fa.User_ID = u.User_ID<br>        WHERE fa.Artist_ID = 4<br>        ) AND fa.Artist_ID != 4; | Get related artists from another specific artist (similar to Spotify’s “Fans Also Like” feature for Artists)<br><br>(Select all artists by name who are favorited by users who also have artist with the Artist_ID of 4 favorited) |
| SELECT DISTINCT s.Song_Title<br>FROM SONG s | Recommend songs to add to a user’s playlist based on a song |

| | |
|---|---|
| JOIN SONG_IN_PLAYLIST sip ON (s.Song_ID = sip.Song_ID AND s.Artist_ID = sip.Artist_ID)<br>JOIN PLAYLIST p ON p.Playlist_ID = sip.Playlist_ID<br>JOIN USER u ON p.User_ID = u.User_ID<br>WHERE s.Song_ID IN (<br> SELECT DISTINCT sip2.Song_ID<br> FROM SONG_IN_PLAYLIST sip2<br> WHERE sip2.Playlist_ID IN (<br>  SELECT sip3.Playlist_ID<br>  FROM SONG_IN_PLAYLIST sip3<br>  WHERE sip3.Song_ID = 34 AND sip3.Artist_ID = 7<br>                                         AND sip3.Playlist_ID != 1<br>  )<br> )<br>AND (s.Song_ID != 34 OR s.Artist_ID != 7) AND u.User_ID != 1; | already in the user's playlist<br><br>(Select all songs by name that are not already in user playlist with Playlist_ID = 1 but are added in other user's playlists with the song with Song_ID = 34 and Artist_ID = 7) |
| SELECT Album_Title, Genre_Name<br>FROM (<br>SELECT alb.Album_Title, g.Genre_Name, COUNT(sig.Genre_ID) as Genre_Count<br>  FROM SONG_IN_GENRE sig<br>  JOIN GENRE g ON g.Genre_ID = sig.Genre_ID<br>  JOIN SONG s ON sig.Song_ID = s.Song_ID<br>  JOIN SONG_IN_ALBUM sia ON sia.Song_ID = s.Song_ID<br>  JOIN ALBUM alb ON sia.Album_ID = alb.Album_ID<br>GROUP BY sig.Genre_ID, alb.Album_ID<br>)<br>WHERE Album_Title = 'Culture'<br>GROUP BY Album_Title<br>HAVING Genre_Count = MAX(Genre_Count); | Summarize the genre of an entire album based on its songs genres<br><br>(Select genre name having the max count of songs within that genre within the album "Culture") |
| DELETE FROM SONG_IN_PLAYLIST<br>WHERE Song_ID =<br>  (SELECT sg.Song_ID<br>   FROM SONG s, SONG_IN_GENRE sg, GENRE g<br>   WHERE g.Genre_Name = 'Pop' AND g.Genre_ID = sg.Genre_ID AND s.Song_ID = sg.Song_ID); | Remove all entries from a given playlistID with genre "Pop" |
| UPDATE USER | Update the payment info of |

| | |
|---|---|
| SET Payment_Info = '2324132312344423'<br>WHERE User_ID = 1; | User with User_ID 1 |

## Normal Form

After evaluating our database through our models, we have determined that all of our relations have achieved **Boyce-Codd Normal Form (BCNF)**. We will discuss how we have achieved this normal form overall by discussing how we achieved BCNF in each unique relation.

### RECORD_LABEL

1NF - No multi-valued attributes
2NF - Candidate Key of Label_ID is minimal, and all other non-key attributes are fully functionally dependent on it
3NF - Label_Name and Label_Owner are not dependent on each other (which would cause a transitive dependency) since labels may have the same name and/or owners with the same name.
BCNF - Due to the same possibility identified in 3NF, Label_Name and Label_Owner as non-key attributes cannot identify the Label_ID key attribute.

### ARTIST

1NF - No multi-valued attributes
2NF - Candidate Key of Artist_ID is minimal, and all other non-key attributes are fully functionally dependent on it.
3NF - No non-key attributes are dependent on each other, so they only depend on the candidate key.
BCNF - The candidate key is not dependent on any non-key attributes.

### ALBUM

1NF - No multi-valued attributes (SONGs in ALBUMs are handled by SONG_IN_ALBUM relation)
2NF - Candidate Key of (Artist_ID, Album_ID) is minimal to identify albums uniquely per each artist, and all other non-key attributes are fully functionally dependent on it.

3NF - No non-key attributes are dependent on each other, so they only depend on the candidate key. Note that albums may have the same title, so Album_Title does not have any transitive dependencies.
BCNF - The candidate key is not dependent on any non-key attributes.

## SONG

1NF - No multi-valued attributes
2NF - Candidate Key of (Artist_ID, Song_ID) is minimal to identify songs uniquely per each artist, and all other non-key attributes are fully functionally dependent on it.
3NF - No non-key attributes are dependent on each other, so they only depend on the candidate key.
BCNF - The candidate key is not dependent on any non-key attributes.

## SONG_IN_ALBUM

1NF - No multi-valued attributes (this relation itself is a solution to 1NF for SONG and ALBUM)
2NF - Candidate Key of (Artist_ID, Album_ID, Song_ID) is minimal, and has no non-key attributes that would cause partial dependencies
3NF - No non-key attributes are present to have possible transitive dependencies
BCNF - No non-key attributes are present to have non-superkey LHS dependencies

## GENRE

1NF - No multi-valued attributes
2NF - Candidate Key of Genre_ID is minimal, and all other non-key attributes are fully functionally dependent on it.
3NF - There is only one non-key attribute, so no transitive dependencies are possible.
BCNF - The candidate key is not dependent on any non-key attributes. Note that we allow Genre's to have the same name since they are uniquely identified by Genre_ID. This makes it so that the Genre_ID cannot be identified by Genre_Name.

## SONG_IN_GENRE

1NF - No multi-valued attributes (this relation itself is a solution to 1NF for SONG and GENRE)

2NF - Candidate Key of (Song_ID, Artist_ID, Genre_ID) is minimal, and has no non-key attributes that would cause partial dependencies
3NF - No non-key attributes are present to have possible transitive dependencies
BCNF - No non-key attributes are present to have non-superkey LHS dependencies

## USER

1NF - No multi-valued attributes
2NF - Candidate Key of User_ID is minimal, and all other non-key attributes are fully functionally dependent on it.
3NF - No non-key attributes are dependent on each other, so they only depend on the candidate key.
BCNF - The candidate key is not dependent on any non-key attributes. We allow users to have the same username -- and it is also possible to have different users using the same payment info.

## FAVORITE_ARTIST

1NF - No multi-valued attributes (this relation itself is a solution to 1NF for USER and ARTIST)
2NF - Candidate Key of (Artist_ID, User_ID) is minimal, and has no non-key attributes that would cause partial dependencies
3NF - No non-key attributes are present to have possible transitive dependencies
BCNF - No non-key attributes are present to have non-superkey LHS dependencies

## FAVORITE_GENRE

1NF - No multi-valued attributes (this relation itself is a solution to 1NF for USER and GENRE)
2NF - Candidate Key of (Genre_ID, User_ID) is minimal, and has no non-key attributes that would cause partial dependencies
3NF - No non-key attributes are present to have possible transitive dependencies
BCNF - No non-key attributes are present to have non-superkey LHS dependencies

### PLAYLIST

1NF - No multi-valued attributes
2NF - Candidate Key of (User_ID, Playlist_ID) is minimal to uniquely identify playlists per each user, and all other non-key attributes are fully functionally dependent on it.
3NF - No non-key attributes are dependent on each other, so they only depend on the candidate key.
BCNF - The candidate key is not dependent on any non-key attributes. We allow playlists to have the same name within user accounts.

### SONG_IN_PLAYLIST

1NF - No multi-valued attributes (this relation itself is a solution to 1NF for SONG and PLAYLIST)
2NF - Candidate Key of (Song_ID, Artist_ID, Playlist_ID) is minimal, and has no non-key attributes that would cause partial dependencies
3NF - No non-key attributes are present to have possible transitive dependencies
BCNF - No non-key attributes are present to have non-superkey LHS dependencies

# Project Evaluation

## How much effort was spent overall?

Across all group members, we have spent a substantial amount of time (20-30+ hours) on this project to complete all deliverables with high quality and ensuring it meets requirements and various design, correctness, clarity, and other standards.

Our team was able to spend less overall time on our ER Diagram and Relational Data Model due to solid initial planning and design during our team meetings -- requiring little-to-no modifications in later stages of the project. In contrast, we put a lot of effort into constructing solid tables and associated queries to meet functional requirements as they would appear in a real application built on top of our database. We also spent a lot of time making realistic sample data for our database to have coverage over many different use cases/queries. We also spent a good amount of time explaining our design decisions and the resulting normalization and queries within this report to clearly and concisely describe our work.

Overall, we believe we have exceeded expectations on our effort input into this project.

## What went right?

Our team was able to propose a solid project idea and its associated design for the database and provided functionalities -- enabling us to quickly move onto the next stages of the project. As discussed in the prior section, our group had designed our ER Diagram and Relational Data Model well which also enabled us to quickly transition into translating it into SQLite statements and queries.

Throughout the timeline of our project, we were also successful in reaching deadlines for components of the project on time while overcoming roadblocks using solid communication amongst team members.

## What went wrong?

A majority of the issues we encountered during the project was during the translation of our functional requirements into SQLite queries. We spent a good amount of time creating sample data to test our queries but ran into issues (such as the storage and comparison of DATE data) which had us stuck in roadblocks until we were able to figure it out. We also did not have a clear outline of important functionalities that we wanted to translate into queries beyond the requirements we specified within our proposal.

The scale of our sample data also caused unnecessary confusion at times that required extra debugging to ensure that data matched across all of our relations and were useful to examine the correctness of our SQL queries.

## How would you do it differently if you had to do it again?

If we had to do the project again, we would have focused on making a small, but high-coverage sample data set for our database to reduce confusion while allowing us to test the correctness of many queries supporting useful high-level functionalities. Related to these functionalities, we would have spent more time outlining the most important functionalities we wanted to cover in queries so we did not waste time writing redundant queries that had to be modified or removed later on in the project.