Katie Kim (kjk5drb), Andrew Shin (ajs5qgm), Gabriella Woo (gdw5bv), Winston Zhang (wyz5rge)
CS 4750: Database Systems
27. October, 2023

CS 4750 Semester Project Checkpoint 2

Introduction

Checkpoint 2 is the second of four deliverables for the semester project. The purpose of checkpoint 2 is to set up the MySQL database and the related tables necessary for the application to run. Additionally, the database is to be populated with sample data that proves the DB design as well as the data are "practical and sufficient". SQL commands were also developed in anticipation of application code implementation that we believed would be commonly used in application usage.

1. Database Server

Initially, our team tried to host our database on a local server. We thought that this would be the most agile solution, and would provide longevity to the application, as SQL scripts used to set up and populate tables could be easily imported into phpMyAdmin by anyone who wanted to try out our app and clone our repository. However, since different team members would not be able to log in to the database, we settled for hosting our database on the UVA CS department's phpMyAdmin server.
There exists the future possibility of transitioning our app from having a centrally hosted database to allowing future users to set up their own database to run the application.

2. Table Schemas

After the database was created, tables were created using SQL statements. The tables were created in accordance with the normalized schema statements from our submission for Milestone 1. For reference, they are enumerated:

```
user(username, name, password, date_joined, streams, background_color, font_color)
follows(username, artist_id)
likes(username, release_id)
manages(username, playlist_id)
recommendations(username, song_id)
owns(artist, song_title, album_title)
song(song_id, artist, song_title, year, streams)
album(album_id, album_title, year, artist)
album_has(album_id, song_id)
artist(artist_id, artist_name)
artist(artist_name, spotify_followers)
releases(artist_id, release_id)
playlist(playlist_id, playlist_title, owner_username, date_created, time_length, song_titles)
playlist_songs(playlist_id, song_id)
library(username, playlists, artists, liked_songs, saved_albums)
library_playlists(username, playlist_id)
library_artists(username, artist_id)
library_liked_songs(username, song_id)
library_saved_albums(username,album_id)
```

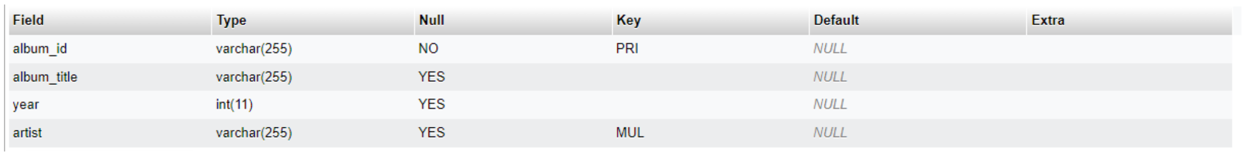
Some schema statements were merged together into one table. This decision was made because it was decided that the splitting of certain data properties after normalization would be impractical when it came to implementation. Other changes are identified and explained:

- "Duration" attribute was added to song schema - this does not affect the rest of the DB design and makes calculating the total duration of a playlist easier to implement
- "Track Number" attribute was also added to the song schema - this is to keep track of the order of songs within an album so that displaying albums will show songs in order. This data value is supported by Spotify API
- "Order" attribute was added to playlist and library playlist tables so that users are able to order their playlists as well as songs within their playlists, and the order is saved to the database in between user sessions
- "Song titles" attribute removed from playlist table, as a second table exists to associate songs with albums.
- For a similar reason, the library table was removed, as it consisted of only a username as a primary key, with all other attributes being multivalued. This resulted in a new table created for each multivalued attribute, leaving the original table with no meaningful data to store except for the username, which is already being used as a primary key in all of the other tables.

The SQL commands for table creation were saved to a SQL file and run against the database. The SQL file is [accessible on GitHub] for further reference.

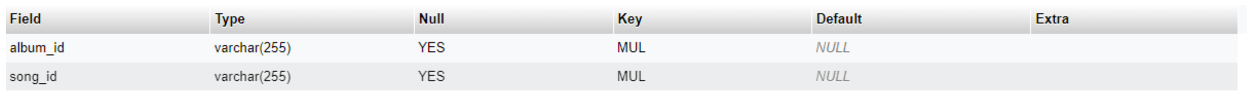
Table schema descriptions were retrieved by executing '`DESC` *`table-name`*'. Screenshots were taken for each table and are included:

album



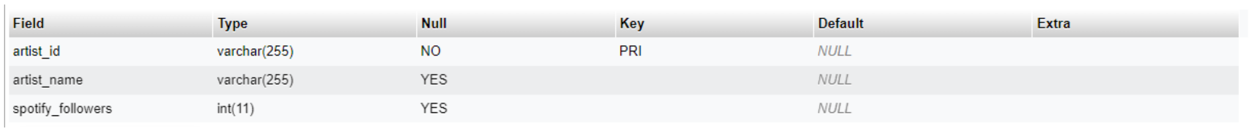
Field	Type	Null	Key	Default	Extra
album_id	varchar(255)	NO	PRI	*NULL*	
album_title	varchar(255)	YES		*NULL*	
year	int(11)	YES		*NULL*	
artist	varchar(255)	YES	MUL	*NULL*	

album_has



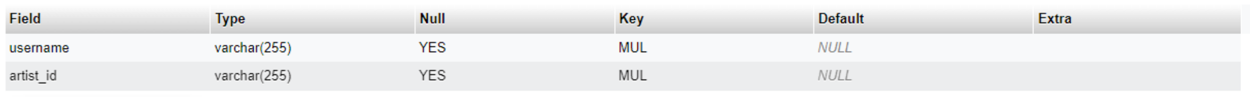
Field	Type	Null	Key	Default	Extra
album_id	varchar(255)	YES	MUL	*NULL*	
song_id	varchar(255)	YES	MUL	*NULL*	

artist



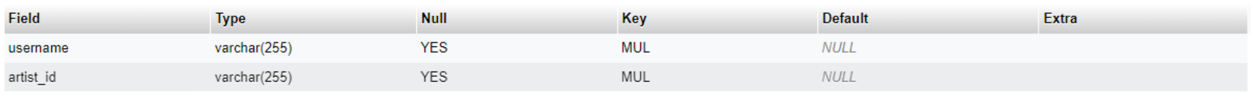
Field	Type	Null	Key	Default	Extra
artist_id	varchar(255)	NO	PRI	*NULL*	
artist_name	varchar(255)	YES		*NULL*	
spotify_followers	int(11)	YES		*NULL*	

follows



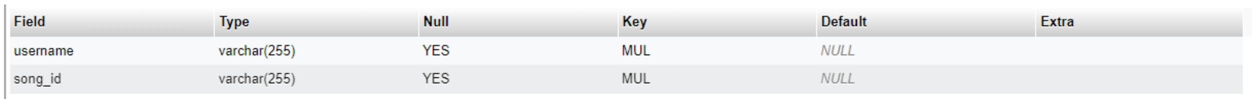
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
artist_id	varchar(255)	YES	MUL	*NULL*	

library_artists



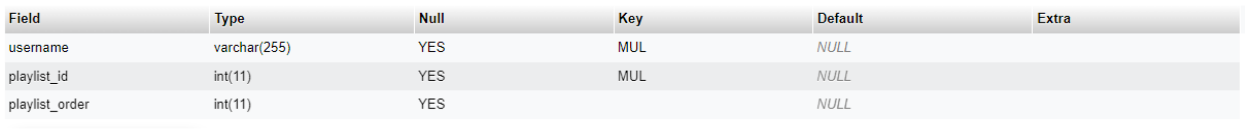
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
artist_id	varchar(255)	YES	MUL	*NULL*	

library_liked_songs



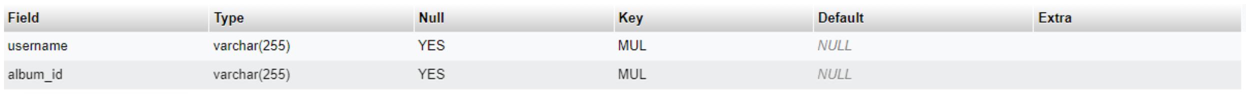
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
song_id	varchar(255)	YES	MUL	*NULL*	

library_playlists



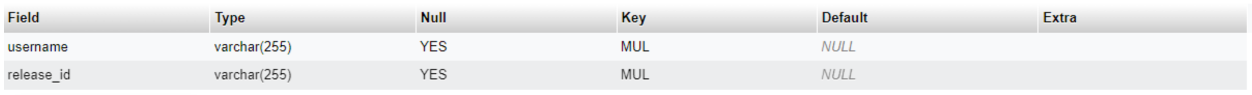
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
playlist_id	int(11)	YES	MUL	*NULL*	
playlist_order	int(11)	YES		*NULL*	

library_saved_albums



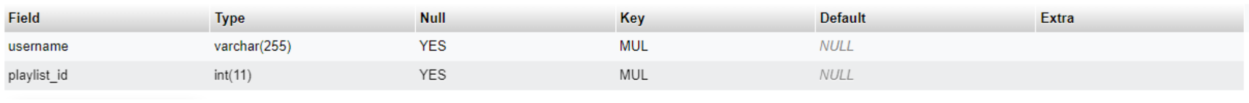
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
album_id	varchar(255)	YES	MUL	*NULL*	

likes



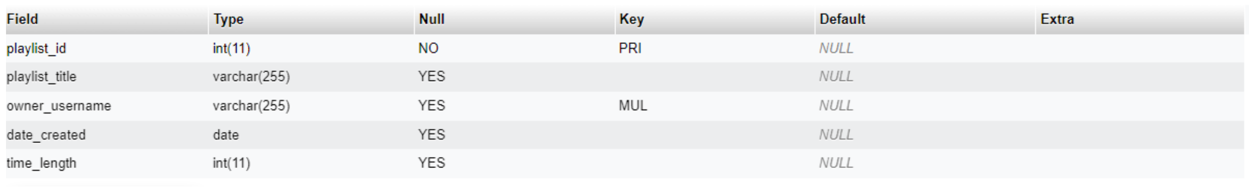
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
release_id	varchar(255)	YES	MUL	*NULL*	

manages



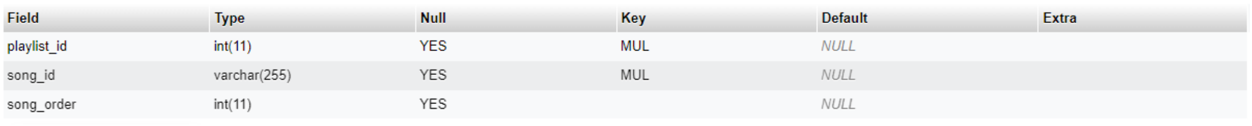
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	*NULL*	
playlist_id	int(11)	YES	MUL	*NULL*	

playlist



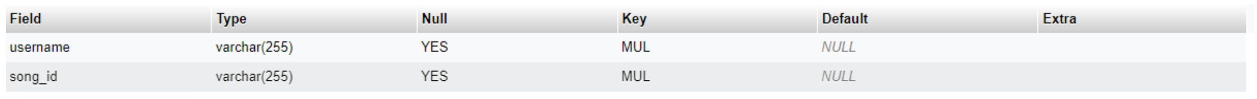
Field	Type	Null	Key	Default	Extra
playlist_id	int(11)	NO	PRI	*NULL*	
playlist_title	varchar(255)	YES		*NULL*	
owner_username	varchar(255)	YES	MUL	*NULL*	
date_created	date	YES		*NULL*	
time_length	int(11)	YES		*NULL*	

playlist_songs



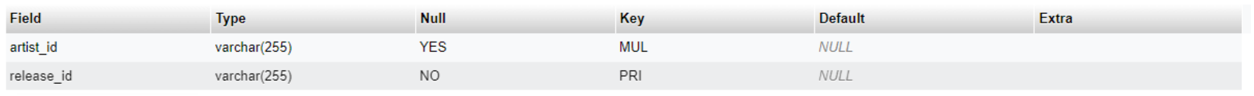
Field	Type	Null	Key	Default	Extra
playlist_id	int(11)	YES	MUL	*NULL*	
song_id	varchar(255)	YES	MUL	*NULL*	
song_order	int(11)	YES		*NULL*	

recommendations



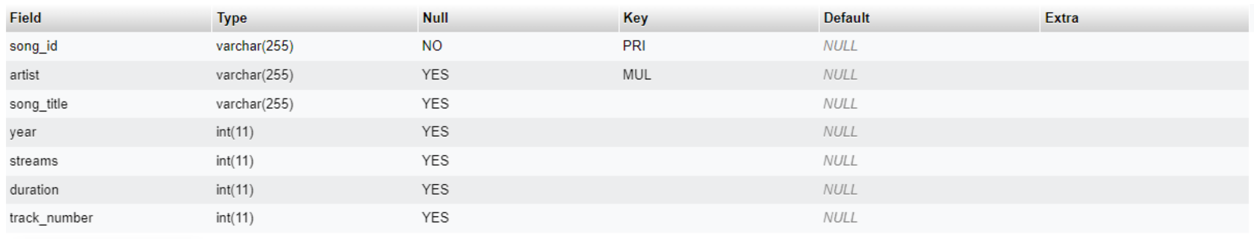
Field	Type	Null	Key	Default	Extra
username	varchar(255)	YES	MUL	NULL	
song_id	varchar(255)	YES	MUL	NULL	

releases



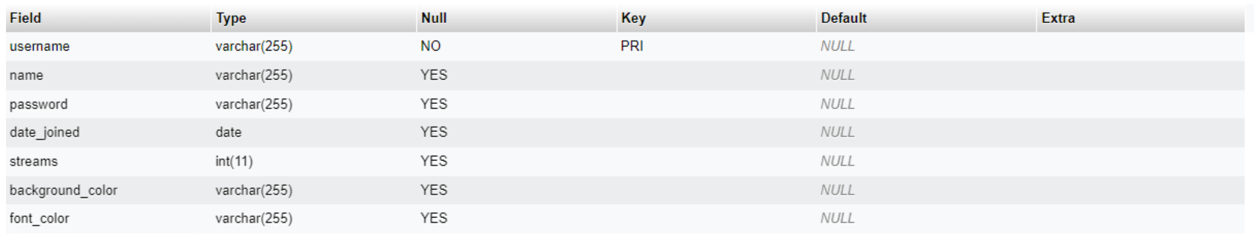
Field	Type	Null	Key	Default	Extra
artist_id	varchar(255)	YES	MUL	NULL	
release_id	varchar(255)	NO	PRI	NULL	

song



Field	Type	Null	Key	Default	Extra
song_id	varchar(255)	NO	PRI	NULL	
artist	varchar(255)	YES	MUL	NULL	
song_title	varchar(255)	YES		NULL	
year	int(11)	YES		NULL	
streams	int(11)	YES		NULL	
duration	int(11)	YES		NULL	
track_number	int(11)	YES		NULL	

user



Field	Type	Null	Key	Default	Extra
username	varchar(255)	NO	PRI	NULL	
name	varchar(255)	YES		NULL	
password	varchar(255)	YES		NULL	
date_joined	date	YES		NULL	
streams	int(11)	YES		NULL	
background_color	varchar(255)	YES		NULL	
font_color	varchar(255)	YES		NULL	

3. Data

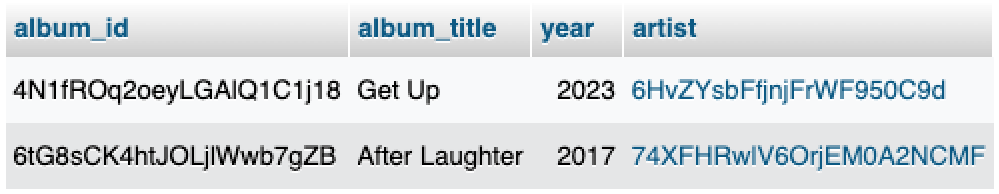
Test data were added to each of the tables. Due to the nature of the multiplicity of songs for any given album, and no application code being written yet, some tables contain fewer rows than others simply due to time and manual effort being a limiting factor. Values were taken from Spotify's Web API and correspond to real albums, artists, and songs.

album table:

The album table contains:
(album_id, album_title, year, artist)

An example sample of data may look like:
("6tG8sCK4htJOLjlWwb7gZB", "After Laughter" , 2017, "74XFHRwlV6OrjEM0A2NCMF")

We have two sample playlists populated in the table currently. Since we intend to utilize our database sort of like a cache, it won't contain all the albums on Spotify. The application will function in such a way that it performs a SQL query for a certain song/album/artist first, and if the app database does not have what a user is looking for, it will then resort to making an API call to Spotify. For example, with our current sample data only containing two albums with all of their songs and artists, if a user wanted to add “Harder Better Faster Stronger” by Daft Punk, the application will see that that song nor its corresponding artist and album are in the database, and an API call will be made to add Daft Punk and their album Discovery to the database. Because of this, having a small number of albums represents a scenario where our first user is accessing their first couple albums, our limited sample conveys an adequate amount of data. Additionally, our sample data shows the possibility of a user to have certain songs saved to a playlist without needing to like them, or follow the artist who made that song, a basic feature of actual Spotify.



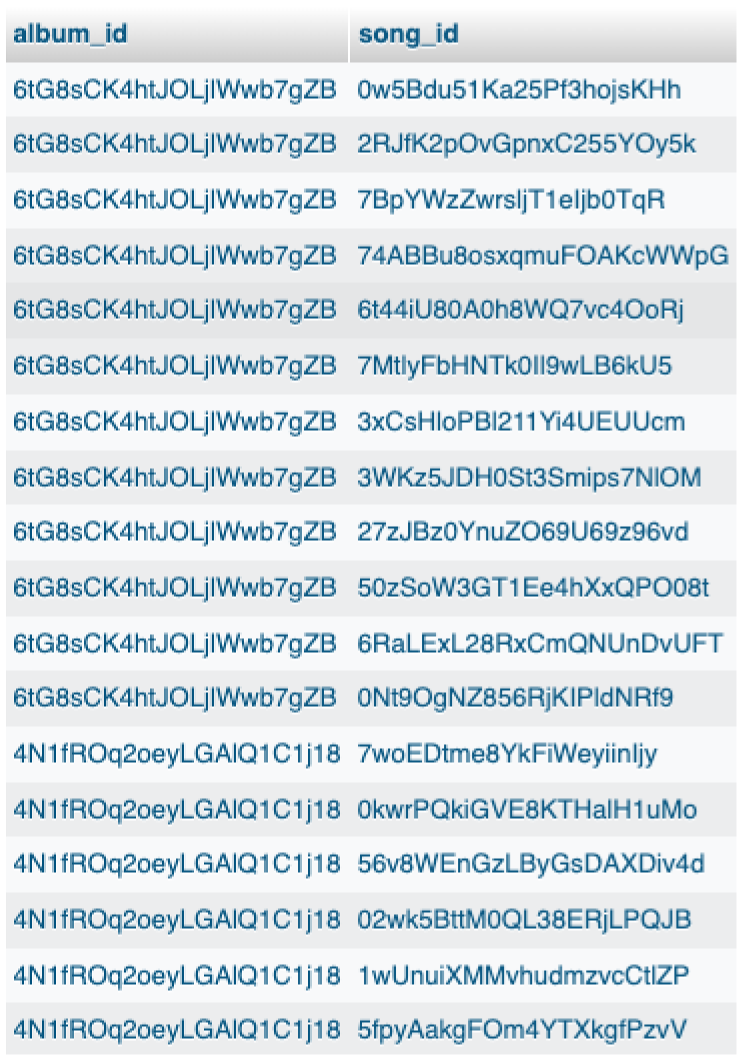
album_id	album_title	year	artist
4N1fROq2oeyLGAlQ1C1j18	Get Up	2023	6HvZYsbFfjnjFrWF950C9d
6tG8sCK4htJOLjlWwb7gZB	After Laughter	2017	74XFHRwlV6OrjEM0A2NCMF

album_has

The album_has table contains:
(album_id, song_id)

An example sample of data may look like:
("6tG8sCK4htJOLjlWwb7gZB", "0w5Bdu51Ka25Pf3hojsKHh")

For this table, we have 2 separate playlists and all of their songs populating it with the format album_id, song_id. The given example above is the album_id for the group Paramore with the song_id for After Laughter. Because this relationship has the information of which songs are in a given album, once a user accesses an album, it will store the album’s song in this table. The given dataset is feasible and adequate because it portrays a scenario where the user accesses their two favorite albums by Paramore and New Jeans. By storing the information in our database, if the user were to search the album again, it won’t be necessary to make an API call, saving additional time and resources.



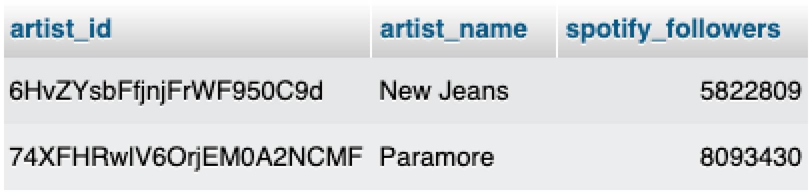
album_id	song_id
6tG8sCK4htJOLjlWwb7gZB	0w5Bdu51Ka25Pf3hojsKHh
6tG8sCK4htJOLjlWwb7gZB	2RJfK2pOvGpnxC255YOy5k
6tG8sCK4htJOLjlWwb7gZB	7BpYWzZwrsljT1eljb0TqR
6tG8sCK4htJOLjlWwb7gZB	74ABBu8osxqmuFOAKcWWpG
6tG8sCK4htJOLjlWwb7gZB	6t44iU80A0h8WQ7vc4OoRj
6tG8sCK4htJOLjlWwb7gZB	7MtlyFbHNTk0Il9wLB6kU5
6tG8sCK4htJOLjlWwb7gZB	3xCsHloPBl211Yi4UEUUcm
6tG8sCK4htJOLjlWwb7gZB	3WKz5JDH0St3Smips7NlOM
6tG8sCK4htJOLjlWwb7gZB	27zJBz0YnuZO69U69z96vd
6tG8sCK4htJOLjlWwb7gZB	50zSoW3GT1Ee4hXxQPO08t
6tG8sCK4htJOLjlWwb7gZB	6RaLExL28RxCmQNUnDvUFT
6tG8sCK4htJOLjlWwb7gZB	0Nt9OgNZ856RjKlPldNRf9
4N1fROq2oeyLGAlQ1C1j18	7woEDtme8YkFiWeyiinljy
4N1fROq2oeyLGAlQ1C1j18	0kwrPQkiGVE8KTHalH1uMo
4N1fROq2oeyLGAlQ1C1j18	56v8WEnGzLByGsDAXDiv4d
4N1fROq2oeyLGAlQ1C1j18	02wk5BttM0QL38ERjLPQJB
4N1fROq2oeyLGAlQ1C1j18	1wUnuiXMMvhudmzvcCtlZP
4N1fROq2oeyLGAlQ1C1j18	5fpyAakgFOm4YTXkgfPzvV

artist

The artist table contains:
(artist_id, artist_name, spotify_followers)

An example sample of data may look like:
("74XFHRwlV6OrjEM0A2NCMF", "Paramore", 8093430)

For this table, we have populated with sample data from two popular groups on spotify. Whenever a user searches for an artist, this table will be populated with the artist_id, name and will update the followers depending on whether the artist was queried previously. The given dataset is proficient because it portrays a scenario where a user searched for the artist paramore, and New Jeans, which caused their data to be stored in the artist table. If they were to query once more the same group, the number of followers for that group would be updated accordingly, and the querying of a new group would add another row to the table.



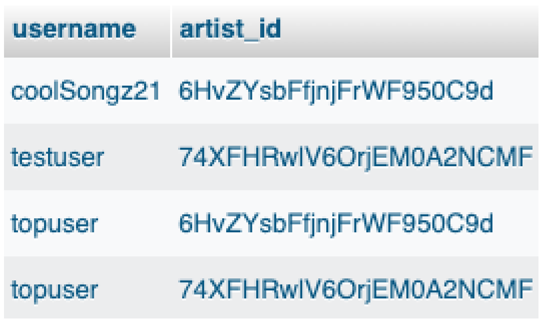
artist_id	artist_name	spotify_followers
6HvZYsbFfjnjFrWF950C9d	New Jeans	5822809
74XFHRwlV6OrjEM0A2NCMF	Paramore	8093430

follows

The follows table contains:
(username, artist_id)

An example sample of data may look like:
("testuser", "74XFHRwlV6OrjEM0A2NCMF")

This table is filled with four total rows with three separate users and the artist_ids of the artists they follow. Whenever the user follows a new artist, this table would be populated with a new row with the artist they follow. The given dataset is feasible and adequate because it shows the relation between user and artist perfectly and shows the scalability of the table structure. As more users follow more artists, this table will become more populated, allowing us to use the standard SELECT, FROM, WHERE query method to portray the user's artists on the application.



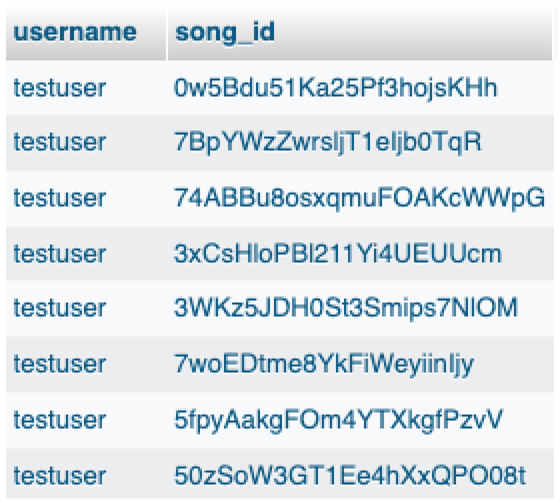
username	artist_id
coolSongz21	6HvZYsbFfjnjFrWF950C9d
testuser	74XFHRwlV6OrjEM0A2NCMF
topuser	6HvZYsbFfjnjFrWF950C9d
topuser	74XFHRwlV6OrjEM0A2NCMF

library_liked_songs

The library_liked_songs table contains:
(username, song_id)

An example sample of data may look like:
("testuser", "0w5Bdu51Ka25Pf3hojsKHh")

This table is filled with 8 total songs "testuser" liked. This table will be further populated with more user song_id pairs as more users like more songs. This dataset is feasible and adequate because it provides a good example of what the table should look like and shows its scalability as more users like more songs. By using the SELECT FROM WHERE query method, the user will be able to access a "playlist" populated with their liked songs on the application.



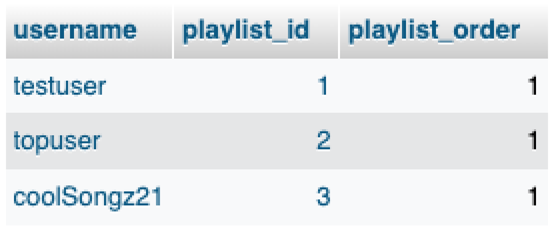
username	song_id
testuser	0w5Bdu51Ka25Pf3hojsKHh
testuser	7BpYWzZwrsljT1eljb0TqR
testuser	74ABBu8osxqmuFOAKcWWpG
testuser	3xCsHloPBl211Yi4UEUUcm
testuser	3WKz5JDH0St3Smips7NlOM
testuser	7woEDtme8YkFiWeyiinljy
testuser	5fpyAakgFOm4YTXkgfPzvV
testuser	50zSoW3GT1Ee4hXxQPO08t

library_playlists

The library_playlists table contains:
(username, playlist_id, playlist_order)

An example sample of data may look like:
("testuser", 1, 1)

This table is currently populated with a sample dataset of three rows consisting of three separate users and their first created playlist. This dataset is adequate and feasible because as users increase the number of playlists they generate, they will be saved into this database and will be portrayed in the application in their corresponding playlist order. Even with a small sample dataset, the idea is fully captured because it shows the specificity, reasoning, and scalability of the table given the set.



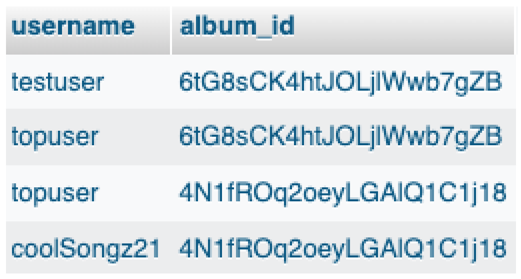
username	playlist_id	playlist_order
testuser	1	1
topuser	2	1
coolSongz21	3	1

library_saved_albums

The library_saved_albums table contains:
(username, album_id)

An example sample of data may look like:
("testuser", "6tG8sCK4htJOLjlWwb7gZB")

The table is populated with a sample dataset of 4 rows consisting of 3 distinct users and their corresponding saved albums. This dataset is adequate and feasible because the given data format provides enough insight on how the table would be further populated. In the scope of the application, by using the SELECT FROM WHERE query method, the user will be able to browse through the albums that they enjoy and saved, allowing for quick and easy access to those albums.



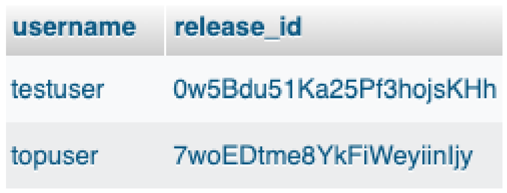
username	album_id
testuser	6tG8sCK4htJOLjlWwb7gZB
topuser	6tG8sCK4htJOLjlWwb7gZB
topuser	4N1fROq2oeyLGAlQ1C1j18
coolSongz21	4N1fROq2oeyLGAlQ1C1j18

likes

The likes table contains:
(username, release_id)

An example sample of data may look like:
("testuser", "0w5Bdu51Ka25Pf3hojsKHh")

This table is populated with a sample dataset of two rows, consisting of two users and their corresponding releases that they enjoy. This dataset is adequate and feasible because the table will be populated every time a user likes a new release from an artist, allowing for quick access after the initial liking moment. Although the dataset is small, it conveys the format of the data and shows the scalability of the table along with the efficiency boost of being able to query a table instead of having to make an API call to access the user data.



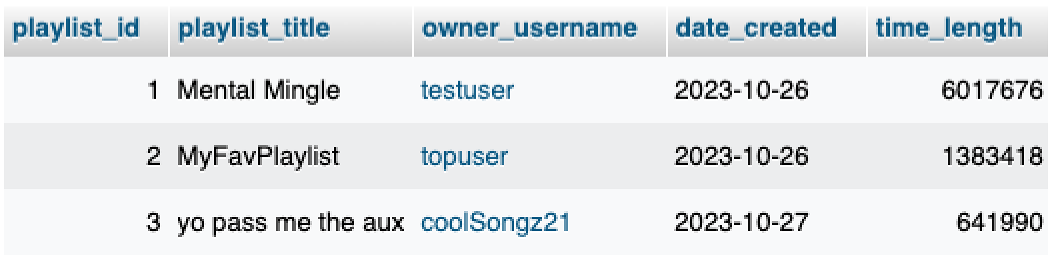
username	release_id
testuser	0w5Bdu51Ka25Pf3hojsKHh
topuser	7woEDtme8YkFiWeyiinljy

playlist

The playlist table contains:
(playlist_id, playlist_title, owner_username, date_created, time_length)

An example sample of data may look like:
(1, "Mental Mingle", "testuser", "2023-10-26", 3008838)

The table is currently populated with three sample playlists from 3 distinct users. We plan to display on the application under playlists the playlists the users create. This can be done by doing a SELECT FROM WHEN query. This dataset is feasible and adequate because it shows the format the playlists will be added to the table when they would be created, whether there are 2 or 100 rows, the conveyability would be the same.



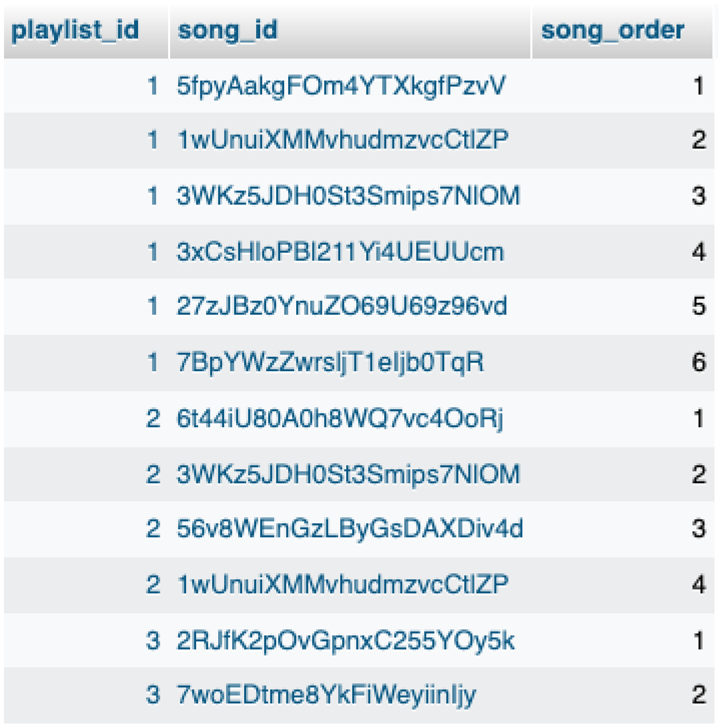
playlist_id	playlist_title	owner_username	date_created	time_length
1	Mental Mingle	testuser	2023-10-26	6017676
2	MyFavPlaylist	topuser	2023-10-26	1383418
3	yo pass me the aux	coolSongz21	2023-10-27	641990

playlist_songs

The playlist_songs table contains:
(playlist_id, song_id, song_order)

An example sample of data may look like:
(1, "5fpyAakgFOm4YTXkgfPzvV", 1)

The table is populated with a sample dataset of 12 rows consisting of the songs within 3 separate playlists. This dataset is adequate and feasible because it conveys the intended format of the data values and can be queried the same with the SELECT FROM WHERE query method regardless of the number of rows that populated the playlist_songs table. As more playlists are created, the larger this table will become.



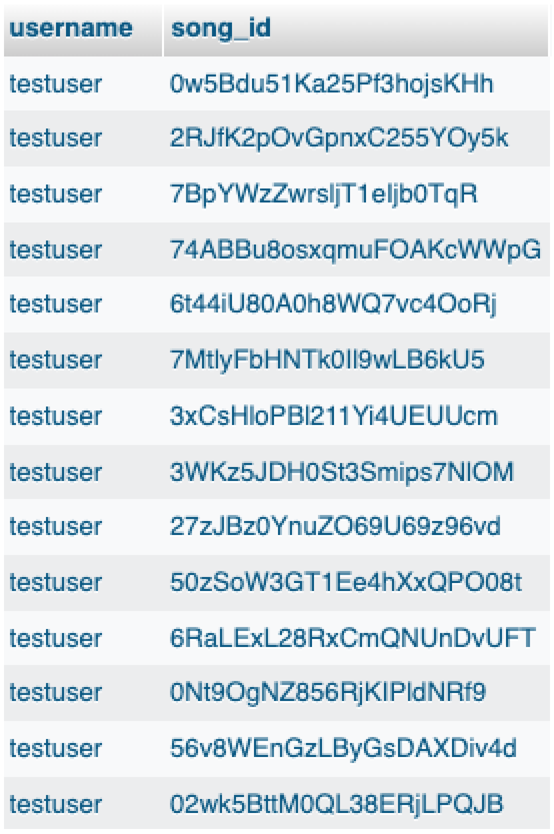
playlist_id	song_id	song_order
1	5fpyAakgFOm4YTXkgfPzvV	1
1	1wUnuiXMMvhudmzvcCtlZP	2
1	3WKz5JDH0St3Smips7NlOM	3
1	3xCsHloPBl211Yi4UEUUcm	4
1	27zJBz0YnuZO69U69z96vd	5
1	7BpYWzZwrsljT1eljb0TqR	6
2	6t44iU80A0h8WQ7vc4OoRj	1
2	3WKz5JDH0St3Smips7NlOM	2
2	56v8WEnGzLByGsDAXDiv4d	3
2	1wUnuiXMMvhudmzvcCtlZP	4
3	2RJfK2pOvGpnxC255YOy5k	1
3	7woEDtme8YkFiWeyiinljy	2

recommendations

The recommendations table contains:
(username, song_id)

An example sample of data may look like:
("testuser", "0w5Bdu51Ka25Pf3hojsKHh")

This table is populated with more than a dozen songs that a single user recommends that others listen to. This table is in theory the same concept as the table liked songs, but differs with the fact that they select these songs because they think others would be missing out if they didn't come across them. This dataset is adequate and feasible because it shows the dedicated format of the data values and conveys a clear message on how this table will be used to implement the application to recommend songs to others.



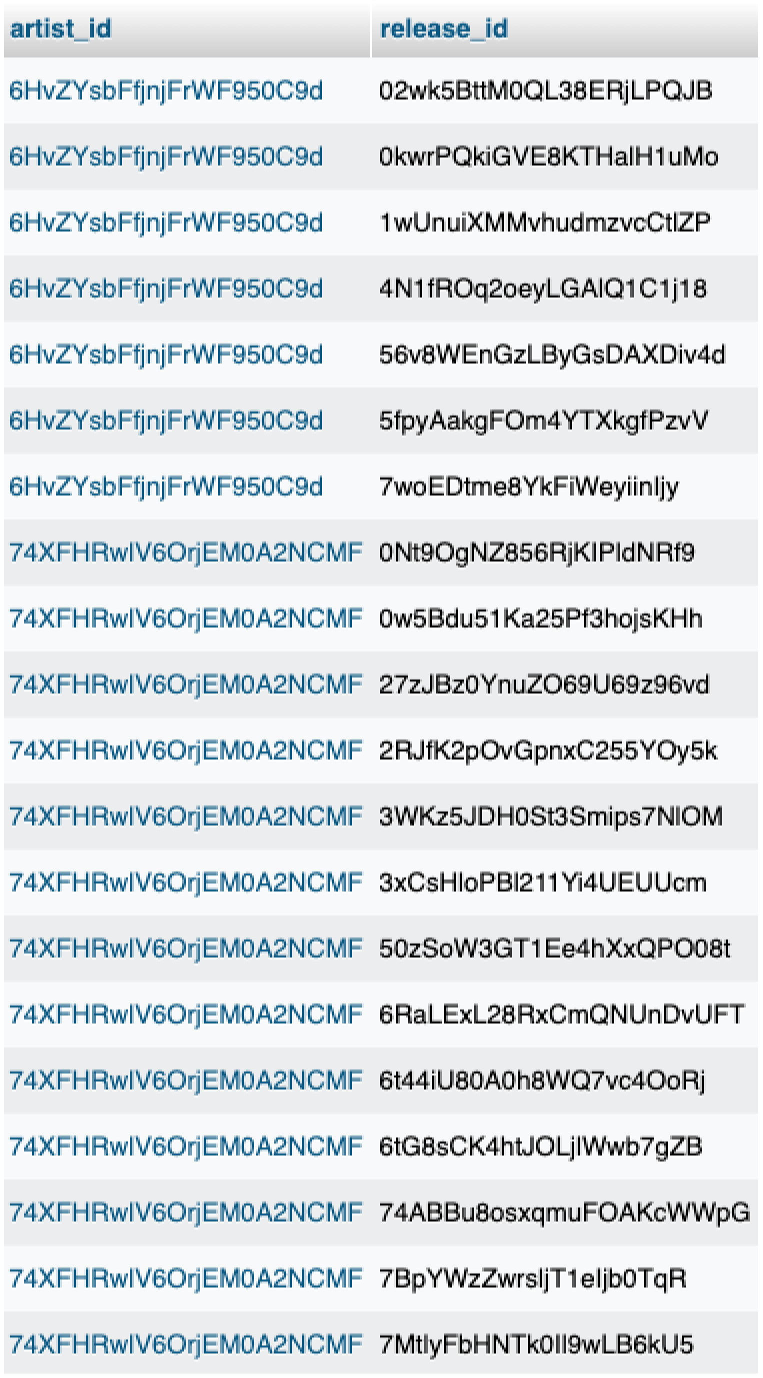
username	song_id
testuser	0w5Bdu51Ka25Pf3hojsKHh
testuser	2RJfK2pOvGpnxC255YOy5k
testuser	7BpYWzZwrsljT1eljb0TqR
testuser	74ABBu8osxqmuFOAKcWWpG
testuser	6t44iU80A0h8WQ7vc4OoRj
testuser	7MtlyFbHNTk0Il9wLB6kU5
testuser	3xCsHloPBl211Yi4UEUUcm
testuser	3WKz5JDH0St3Smips7NIOM
testuser	27zJBz0YnuZO69U69z96vd
testuser	50zSoW3GT1Ee4hXxQPO08t
testuser	6RaLExL28RxCmQNUnDvUFT
testuser	0Nt9OgNZ856RjKIPldNRf9
testuser	56v8WEnGzLByGsDAXDiv4d
testuser	02wk5BttM0QL38ERjLPQJB

releases

The releases table contains:
(artist_id, release_id)

An example sample of data may look like:
("74XFHRwlV6OrjEM0A2NCMF", "6tG8sCK4htJOLjlWwb7gZB")

This table is the same concept as the relation table between the album and the songs, but instead portrays it with the artist and their songs. When an artist releases a new song or album, this table will be populated with rows of the artist's ID and the songs that they create. This table's dataset is adequate and feasible because it shows the relation between artists and their songs with clarity, there will be many of these and the given dataset effectively visualizes this relationship.



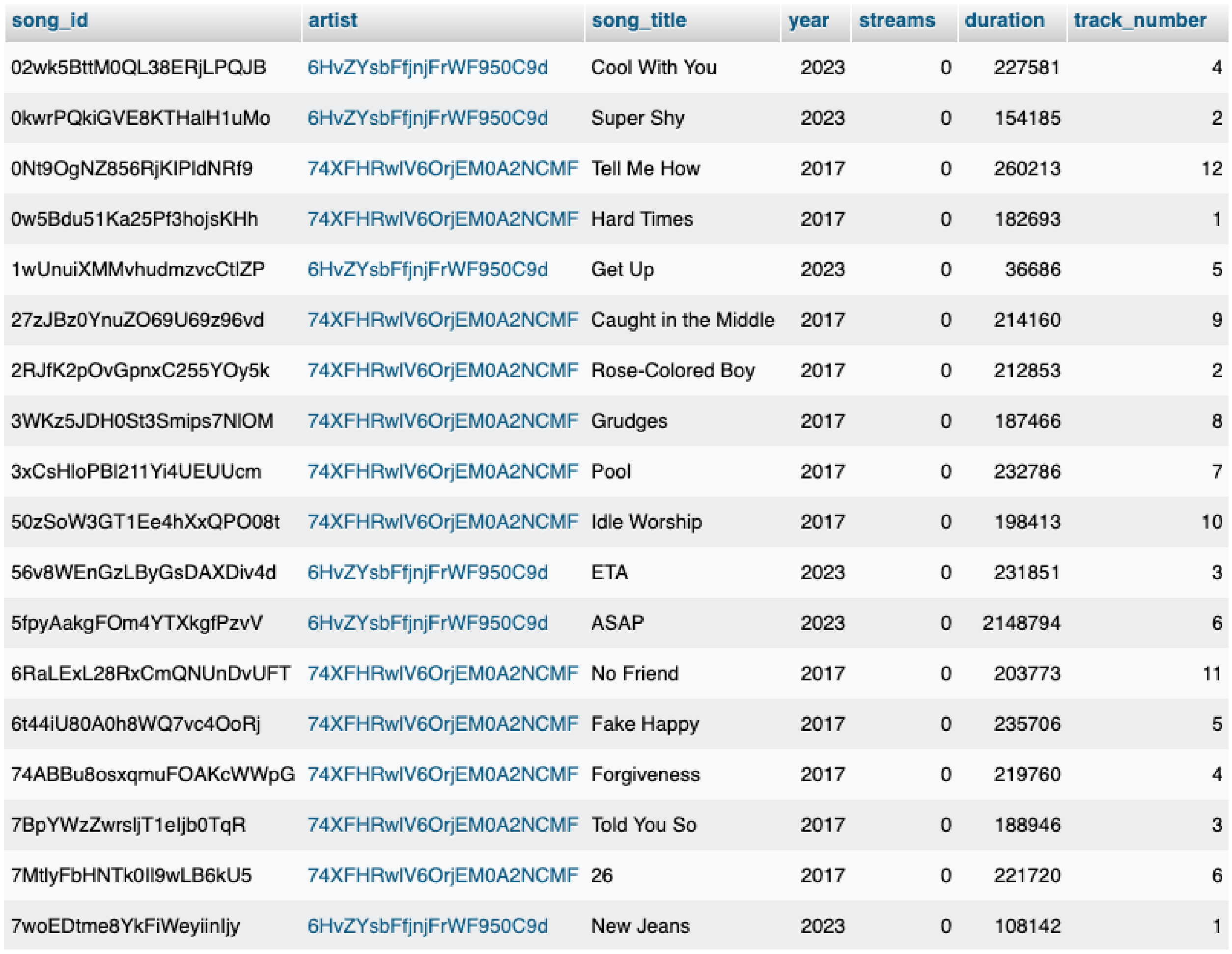
artist_id	release_id
6HvZYsbFfjnjFrWF950C9d	02wk5BttM0QL38ERjLPQJB
6HvZYsbFfjnjFrWF950C9d	0kwrPQkiGVE8KTHalH1uMo
6HvZYsbFfjnjFrWF950C9d	1wUnuiXMMvhudmzvcCtlZP
6HvZYsbFfjnjFrWF950C9d	4N1fROq2oeyLGAlQ1C1j18
6HvZYsbFfjnjFrWF950C9d	56v8WEnGzLByGsDAXDiv4d
6HvZYsbFfjnjFrWF950C9d	5fpyAakgFOm4YTXkgfPzvV
6HvZYsbFfjnjFrWF950C9d	7woEDtme8YkFiWeyiinljy
74XFHRwlV6OrjEM0A2NCMF	0Nt9OgNZ856RjKIPldNRf9
74XFHRwlV6OrjEM0A2NCMF	0w5Bdu51Ka25Pf3hojsKHh
74XFHRwlV6OrjEM0A2NCMF	27zJBz0YnuZO69U69z96vd
74XFHRwlV6OrjEM0A2NCMF	2RJfK2pOvGpnxC255YOy5k
74XFHRwlV6OrjEM0A2NCMF	3WKz5JDH0St3Smips7NlOM
74XFHRwlV6OrjEM0A2NCMF	3xCsHloPBl211Yi4UEUUcm
74XFHRwlV6OrjEM0A2NCMF	50zSoW3GT1Ee4hXxQPO08t
74XFHRwlV6OrjEM0A2NCMF	6RaLExL28RxCmQNUnDvUFT
74XFHRwlV6OrjEM0A2NCMF	6t44iU80A0h8WQ7vc4OoRj
74XFHRwlV6OrjEM0A2NCMF	6tG8sCK4htJOLjlWwb7gZB
74XFHRwlV6OrjEM0A2NCMF	74ABBu8osxqmuFOAKcWWpG
74XFHRwlV6OrjEM0A2NCMF	7BpYWzZwrsljT1eljb0TqR
74XFHRwlV6OrjEM0A2NCMF	7MtlyFbHNTk0Il9wLB6kU5

song

The song table contains:
(song_id, artist, song_title, year, streams, duration, track_number)

An example sample of data may look like:
("0w5Bdu51Ka25Pf3hojsKHh", "74XFHRwlV6OrjEM0A2NCMF", "Hard Times", 2017, 0, 182693, 1)

The given table is populated with a sample dataset of songs and their fields. As more songs are accessed by users, more songs will be recorded into this table. This dataset is adequate and feasible because it accurately stores the parameters of all songs searched and selected by users, given that this table has a lot of columns, it can be queried for many reasons and can fulfill many other relations/functional requirements.



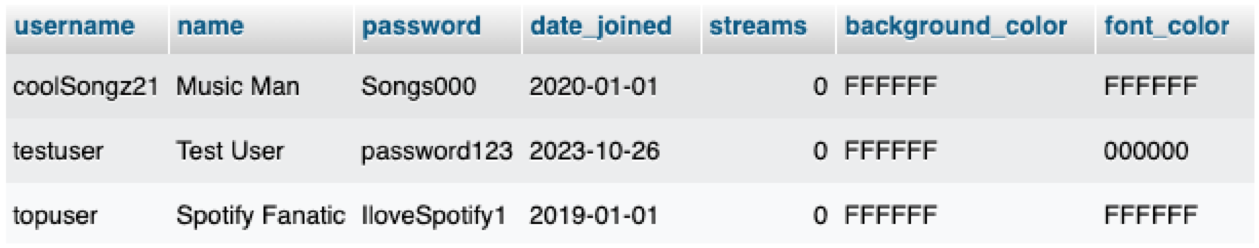
song_id	artist	song_title	year	streams	duration	track_number
02wk5BttM0QL38ERjLPQJB	6HvZYsbFfjnjFrWF950C9d	Cool With You	2023	0	227581	4
0kwrPQkiGVE8KTHalH1uMo	6HvZYsbFfjnjFrWF950C9d	Super Shy	2023	0	154185	2
0Nt9OgNZ856RjKIPIdNRf9	74XFHRwlV6OrjEM0A2NCMF	Tell Me How	2017	0	260213	12
0w5Bdu51Ka25Pf3hojsKHh	74XFHRwlV6OrjEM0A2NCMF	Hard Times	2017	0	182693	1
1wUnuiXMMvhudmzvcCtlZP	6HvZYsbFfjnjFrWF950C9d	Get Up	2023	0	36686	5
27zJBz0YnuZO69U69z96vd	74XFHRwlV6OrjEM0A2NCMF	Caught in the Middle	2017	0	214160	9
2RJfK2pOvGpnxC255YOy5k	74XFHRwlV6OrjEM0A2NCMF	Rose-Colored Boy	2017	0	212853	2
3WKz5JDH0St3Smips7NlOM	74XFHRwlV6OrjEM0A2NCMF	Grudges	2017	0	187466	8
3xCsHloPBl211Yi4UEUUcm	74XFHRwlV6OrjEM0A2NCMF	Pool	2017	0	232786	7
50zSoW3GT1Ee4hXxQPO08t	74XFHRwlV6OrjEM0A2NCMF	Idle Worship	2017	0	198413	10
56v8WEnGzLByGsDAXDiv4d	6HvZYsbFfjnjFrWF950C9d	ETA	2023	0	231851	3
5fpyAakgFOm4YTXkgfPzvV	6HvZYsbFfjnjFrWF950C9d	ASAP	2023	0	2148794	6
6RaLExL28RxCmQNUnDvUFT	74XFHRwlV6OrjEM0A2NCMF	No Friend	2017	0	203773	11
6t44iU80A0h8WQ7vc4OoRj	74XFHRwlV6OrjEM0A2NCMF	Fake Happy	2017	0	235706	5
74ABBu8osxqmuFOAKcWWpG	74XFHRwlV6OrjEM0A2NCMF	Forgiveness	2017	0	219760	4
7BpYWzZwrsljT1eljb0TqR	74XFHRwlV6OrjEM0A2NCMF	Told You So	2017	0	188946	3
7MtlyFbHNTk0Il9wLB6kU5	74XFHRwlV6OrjEM0A2NCMF	26	2017	0	221720	6
7woEDtme8YkFiWeyiinljy	6HvZYsbFfjnjFrWF950C9d	New Jeans	2023	0	108142	1

user

The user table contains:
(username, name, password, date_joined, streams, background_color, font_color)

An example sample of data may look like:
("testuser", "Test User", "password123", "2023-10-26", 0, "FFFFFF", "000000")

This table is populated with a sample dataset with 3 rows with 3 distinct users. It shows the formatting of user information when a new one is created. When a new user is created, this table will be given another row. This dataset is feasible and accurate because it represents a scenario where the first 3 users have created an account with our application. It shows the accuracy and the scalability of this table as the data is easily understandable and can be increased as more users are created.



username	name	password	date_joined	streams	background_color	font_color
coolSongz21	Music Man	Songs000	2020-01-01	0	FFFFFF	FFFFFF
testuser	Test User	password123	2023-10-26	0	FFFFFF	000000
topuser	Spotify Fanatic	IloveSpotify1	2019-01-01	0	FFFFFF	FFFFFF

4. SQL Commands

In order to collaborate on a central database, we had one of our group members grant all privileges of their user, <computingid>, to <computingid_a>, <computingid_b>, and <computingid_c> with the following commands:
GRANT ALL ON <computingid>.* TO '<computingid_a>' @'%';
GRANT ALL PRIVILEGES ON <computingid>.* TO '<computingid_a>' @'%';
As mentioned in section 2, the SQL commands for table creation were saved to a SQL file and run against the database. The SQL file is accessible on GitHub for further reference.
Additionally, a collection of SQL commands that we foresaw being used for certain operations when running the application were developed. Such operations included login, adding/removing songs and albums, creating/deleting playlists, following/unfollowing artists, etc. These were all written with placeholder variable names in the `WHERE` clause of the query and saved to a file which was then pushed to our code repository. Placeholder names were used for parameterization purposes and for ease of implementation when we start developing application code.
Similarly, advanced SQL queries were also developed, which make use of constraints and triggers. Such commands ensure consistency of data formatting as new rows are added to tables. These were saved to a different file and also pushed to GitHub.

5. SQL File

In addition to the data insertions made via SQL, as well as the SQL commands documented in the previous section, a `.sql` file was exported from phpMyAdmin detailing all of the commands run against the database. This file is included with this report in our submission.

Honor Pledge

On our honor as students, we have neither given nor received unauthorized aid on this assignment.

Katie Kim (kjk5drb),
Andrew Shin (ajs5qgm),
Gabriella Woo (gdw5bv),
Winston Zhang (wyz5rge)

27. October, 2023