Final Project Report

Course Name: Databases: Concepts and Usage
Course Code: COMP 3380

Submitted by:
Team 09

Team Members:
Sudipta Dip (dips/7900493)
Sukhmeet Singh Hora (horass/7884859)
Rishamdeep Singh (singhr50/7900942)

Submission Date: April 10th, 2024

Table of Contents

1. Project Introduction

In the ever-evolving landscape of the music industry, the power of data cannot be overstated. Our database project is a testament to this, offering a meticulously structured relational database model that captures the multifaceted aspects of albums, artists, and songs. With a rich array of queries at their disposal, analysts can delve deep into the data to extract meaningful insights. From identifying top artists by popularity and genre to analysing the evolution of an artist's popularity over time, our queries provide a multifaceted view of the music industry. Analysts can explore trends in album releases, track popularity which can help them to pinpoint the most influential artists or albums within a specific genre or time frame. Moreover, our database allows analysts to scrutinize individual songs and albums which enables them to gauge the acoustic attributes and duration patterns of albums, providing valuable insights into artistic styles and audience preferences. With all these essential queries, our database can help an analyst to also forecast future industry shifts. By harnessing the potential of our database, analysts can become pivotal players in steering the music industry towards success.

2. Data Summary

Our music-related database encompasses 9 primary files, including information on albums, artists, releases, songs, tracks, charts, popularity scores, lyrics, and acoustic features which was selected from the source [1]. Through rigorous cleaning and normalization processes, these initial files have been transformed into a total of 12 tables, each representing specific aspects of the music industry landscape. These tables are interconnected through unique identifiers, facilitating comprehensive analysis and exploration of relationships across the given participating factors.

2.1. Reason of Choosing or Selection

a. *Comprehensive Coverage*: The dataset includes albums, artists, and songs, offering a holistic view of the music industry.

b. *Complexity and Connectivity*: With over 10 tables and 1000 rows, the dataset provides intricate interconnections for thorough analysis.

c. *Data Quality*: Emphasis was placed on high data quality, minimizing blank entries and ensuring completeness for reliable analysis.

d. *Relevance and Interest*: The dataset was chosen to align with users' interests, ensuring engagement and enthusiasm for the project [2].

e. *Availability and Accessibility*: Sourced from public sources, the dataset ensures accessibility while adhering to copyright and licensing requirements [2].

2.2. Attributes

i. Albums

- *album_id* (Primary Key): Unique identifier for each album.
- *name*: Name of the album.
- *popularity*: Popularity score of the album.
- *total_tracks*: Total number of tracks in the album.
- *album_type*: Type of the album (e.g., album, single, compilation).

ii. Artists

- *artist_id* (Primary Key): Unique identifier for each artist.
- *name*: Name of the artist.
- *followers*: Number of followers of the artist.
- *popularity*: Popularity score of the artist.
- *artist_type*: Type of the artist (e.g., solo, band).
- *main_genre*: Main genre of the artist.

iii. Songs

- *song_id* (Primary Key): Unique identifier for each song.
- *name*: Name of the song.
- *popularity*: Popularity score of the song.
- *type*: Type of the song (e.g., solo, collaboration).
- *album_id* (Foreign Key): Identifier of the album the song belongs to.
- *track_no*: Track number of the song in the album.
- *release_year*: Year of song release.
- *release_month*: Month of song release.
- *release_date*: Exact date of song release.
- *duration_ms*: Duration of the song in milliseconds.
- *key*: Key of the song.
- *time_signature*: Time signature of the song.
- *acousticness*: Acousticness score of the song.
- *danceability*: Danceability score of the song.
- *energy*: Energy score of the song.
- *instrumentalness*: Instrumentalness score of the song.
- *liveness*: Liveness score of the song.
- *loudness*: Loudness of the song.
- *speechiness*: Speechiness score of the song.
- *valence*: Valence score of the song.
- *tempo*: Tempo of the song.
- *lyrics*: Lyrics of the song.

iv. Releases

- *artist_id* (Foreign Key): Identifier of the artist.
- *album_id* (Foreign Key): Identifier of the album.
- *release_year*: Year of release.
- *release_month*: Month of release.
- *release_date*: Exact date of release.

v. Writes

- *song_id* (Foreign Key): Identifier of the song.
- *artist_id* (Foreign Key): Identifier of the artist.

vi. ArtistGenre

- *artist_id* (Foreign Key): Identifier of the artist.
- *genre*: Genre associated with the artist.

vii. AlbumChart

- *sl_no_album_chart* (Primary Key): Sequential number for each record.
- *album_id* (Foreign Key): Identifier of the album.
- *rank_score*: Rank score of the album.
- *peak_position*: Peak position of the album on the chart.

- *week_year*: Year of the chart week.
- *week_month*: Month of the chart week.
- *week_date*: Date of the chart week.
- *weeks_on_chart_count*: Number of weeks the album stayed on the chart.

viii. ArtistChart

- *sl_no_artist_chart* (Primary Key): Sequential number for each record.
- *artist_id* (Foreign Key): Identifier of the artist.
- *rank_score*: Rank score of the artist.
- *peak_position*: Peak position of the artist on the chart.
- *week_year*: Year of the chart week.
- *week_month*: Month of the chart week.
- *week_date*: Date of the chart week.
- *weeks_on_chart_count*: Number of weeks the artist stayed on the chart.

ix. SongChart

- *sl_no_song_chart* (Primary Key): Sequential number for each record.
- *song_id* (Foreign Key): Identifier of the song.
- *rank_score*: Rank score of the song.
- *peak_position*: Peak position of the song on the chart.
- *week_year*: Year of the chart week.
- *week_month*: Month of the chart week.
- *week_date*: Date of the chart week.
- *weeks_on_chart_count*: Number of weeks the song stayed on the chart.

x. AlbumPop

- *sl_no_album_pop* (Primary Key): Sequential number for each record.
- *album_id* (Foreign Key): Identifier of the album.
- *year*: Year of the popularity score.
- *year_end_score*: Year-end popularity score of the album.

xi. ArtistPop

- *sl_no_artist_pop* (Primary Key): Sequential number for each record.
- *artist_id* (Foreign Key): Identifier of the artist.
- *year*: Year of the popularity score.
- *year_end_score*: Year-end popularity score of the artist.

xii. SongPop

- *sl_no_song_pop* (Primary Key): Sequential number for each record.
- *song_id* (Foreign Key): Identifier of the song.
- *year*: Year of the popularity score.
- *year_end_score*: Year-end popularity score of the song.

2.3. Cleaning, Normalization and Pre-Processing

The following steps had been taken to clean, normalize and pre-process the database to have a processed and optimal dataset.

1. **Remove Blank Rows**: Blank rows were identified and removed from each file to eliminate any unnecessary or incomplete data entries.

2. **Handle Null Values in Primary Key Columns**: Null values in primary key columns were addressed by either removing the affected rows or imputing appropriate values where applicable. This ensures the integrity of unique identifiers within the dataset.

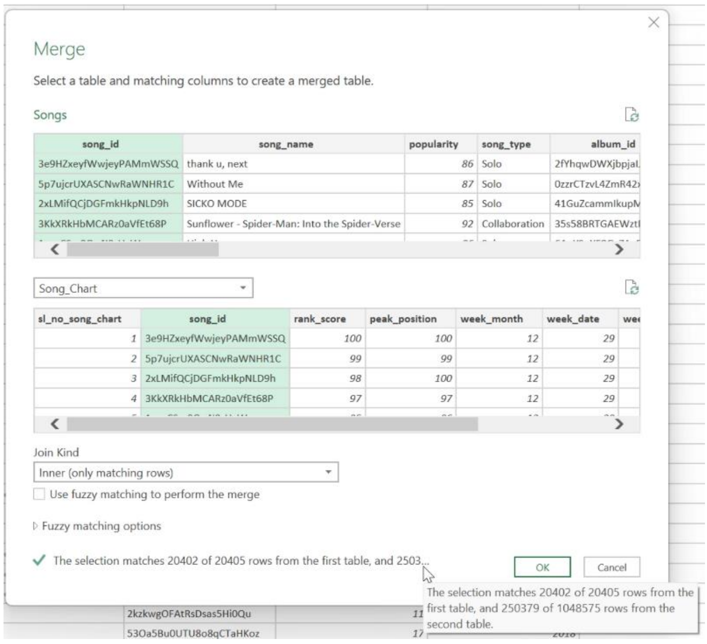
3. **Handle Duplicates**: Duplicate entries were identified and removed to avoid redundancy and maintain data accuracy.

4. **Standardize Data Formats**: Data formats were standardized across the dataset to ensure consistency and facilitate analysis. This includes standardizing date formats, text capitalization, and numerical representations.

5. **Normalize Data**: Data normalization techniques were applied to minimize redundancy and improve database efficiency. This involves organizing data into separate tables to reduce data duplication and improve data integrity.

6. **Resolve Data Integrity Issues**: Any inconsistencies or discrepancies in the dataset were addressed to ensure data integrity. This includes resolving inconsistencies in naming conventions, genre classifications, and other categorical data.

7. **Validate Referential Integrity**: Referential integrity constraints were enforced to ensure that relationships between tables are maintained and that foreign key references are valid. In this case, we performed inner joins to merge and process the tables where we had total participation from both sides of a relation (fig -01).


Merge

Select a table and matching columns to create a merged table.

Songs

song_id	song_name	popularity	song_type	album_id
3e9HZxeyfWwjeyPAMmWSSQ	thank u, next	86	Solo	2fYhqwDWXjbpjal
5p7ujcrUXASCNwRaWNHR1C	Without Me	87	Solo	0zzrCTzvL4ZmR42
2xLMifQCjDGFmkHkpNLD9h	SICKO MODE	85	Solo	41GuZcammIkupM
3KkXRkHbMCARz0aVfEt68P	Sunflower - Spider-Man: Into the Spider-Verse	92	Collaboration	35s58BRTGAEWztl

Song_Chart

sl_no_song_chart	song_id	rank_score	peak_position	week_month	week_date	we
1	3e9HZxeyfWwjeyPAMmWSSQ	100	100	12	29	
2	5p7ujcrUXASCNwRaWNHR1C	99	99	12	29	
3	2xLMifQCjDGFmkHkpNLD9h	98	100	12	29	
4	3KkXRkHbMCARz0aVfEt68P	97	97	12	29	

Join Kind

Inner (only matching rows)

☐ Use fuzzy matching to perform the merge

Fuzzy matching options

✓ The selection matches 20402 of 20405 rows from the first table, and 2503...

The selection matches 20402 of 20405 rows from the first table, and 250379 of 1048575 rows from the second table.

2kzkwgOFAtRsDsas5Hi0Qu 11

53Oa5Bu0UTU8o8qCTaHKoz 17

Fig-01: Ensuring Referential Integrity through Power Query's Merge (Join)

Note that, for the cleaning and pre-processing operations, we mostly used **DAX formula**, **Power Query** and **Power BI** transformations.

Original Source Files [1]	***Total Columns***	***Total Entries***		***Database Tables***	***Total Attributes***	***Total Entries***
Albums	8	26519		Albums	5	26518
Artists	8	11518		Artists	6	5825
Releases	4	26522		ArtistGenre	2	26990
Songs	7	20405		Releases	5	19100
Lyrics	2	20404		Songs	22	20405
Tracks	5	20405	→ Cleaning, Normalization and Pre-processing →	Writes	2	16436
AlbumChart	5	471706		AlbumChart	8	469349
ArtistChart	5	530379		ArtistChart	8	402067
SongChart	5	250392		SongChart	8	250379
AlbumPop	4	39469		AlbumPop	4	39464
ArtistPop	4	36997		ArtistPop	4	26064
SongPop	4	25194		SongPop	4	25193
AcousticFeatures	14	20405				

2.4. ER Diagram

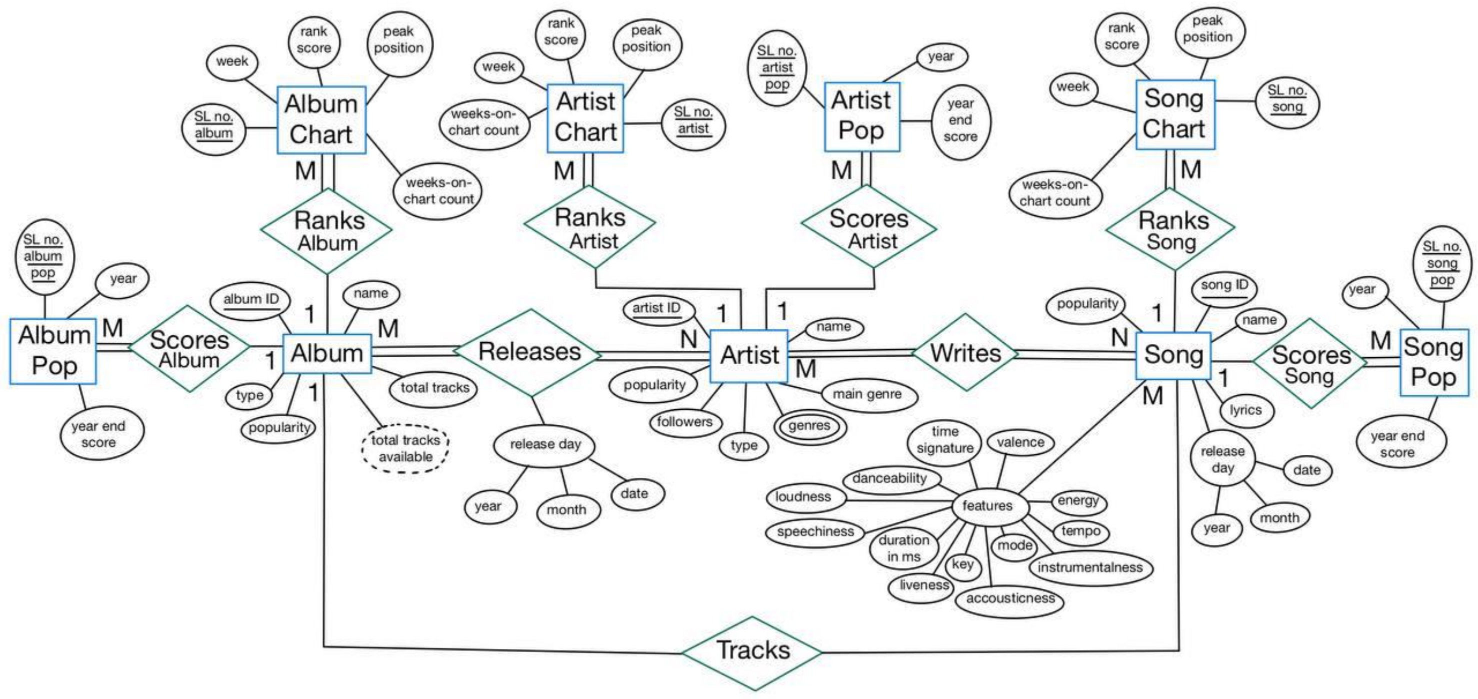


Fig-02: Database ER Diagram

2.5. Normalized Relational Model

1. Albums(**album_id**, name, popularity, total_tracks, album_type)
2. Artists(**artist_id**, name, followers, popularity, artist_type, main_genre)
3. Songs(**song_id**,song_name,popularity,song_type,album_id,track_no,release_year,release_month,release_date,duration_ms,key,time_signature,acousticness,danceablity,energy,instrumentalness,liveness,loudness,speechiness,valence,tempo,lyrics)
4. Releases(**artist_id**,**album_id**,release_year, release_month, release_date)
5. Writes(**song_id**,**artist_id**)
6. ArtistGenre(**artist_id**,**genre**)
7. AlbumChart(**sl_no_album_chart**,album_id,rank_score,peak_position,week_year,week_month,week_date,weeks_on_chart_count)
8. ArtistChart(**sl_no_artist_chart**,artist_id,rank_score,peak_position,week_year,week_month,week_date,weeks_on_chart_count)
9. SongChart(**sl_no_song_chart**,song_id,rank_score,peak_position,week_year,week_month,week_date,weeks_on_chart_count)
10. AlbumPop(**sl_no_album_pop**,album_id,year,year_end_score)
11. ArtistPop(**sl_no_artist_pop**,artist_id,year,year_end_score)
12. SongPop(**sl_no_song_pop**,song_id,year,year_end_score)

2.6. Data Entity Model

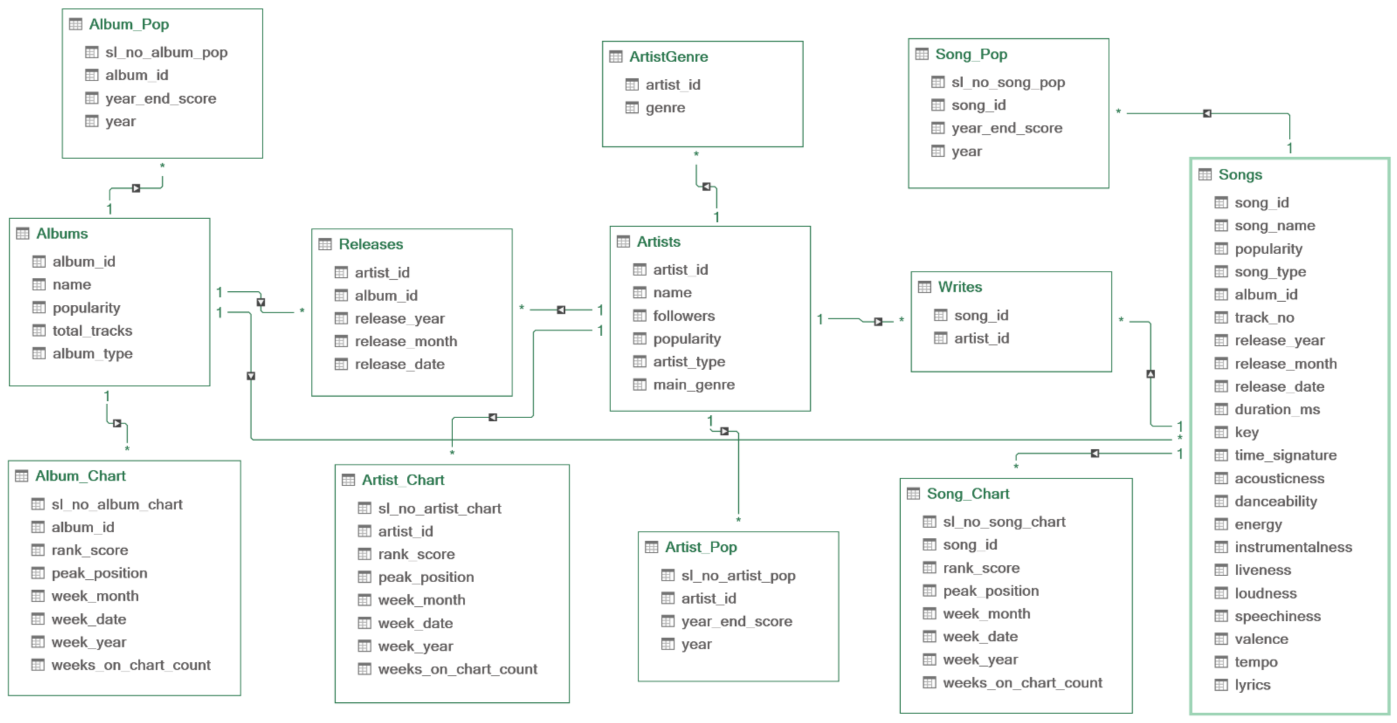


Fig-03: Data Entity Model by Power Query Connection

3. Discussion of the Data Model

3.1. Table Breakdown Reasons

The data model was already broken down into multiple tables to reflect the complex relationships and attributes within the music industry. Each table represents a distinct entity (e.g., albums, artists, songs), allowing for efficient data organization and retrieval [2]. However, we required further break-down of the data to ensure data integrity and to reduce redundancy.

a. *Remove unnecessary attributes*
The attribute *billboard* was mostly restating the values of the name attribute in songs and albums which was removed. Again, the artists table had *image_url* of their images; many of the artists had *null* values for this attribute and the available images also do not serve any analytical purpose, hence, we removed it. There were some boolean attributes like *explicit*, *is_pop* etc. which were also eliminated as they can also be determined by some corresponding scoring attributes.

b. *Handle multi-value attribute*
Artists table had a multi-value attribute named *genres* which stored all the related genres for an artist. For handling this concern, we made a separate table named *ArtistGenre* and used Power Query delimiter and unpivot operation to assign multiple genres to the same artist at different table entries. We also had to breakdown all the *release_date* attributes into *date*, *month* and *year* for ensuring proper and convenient comparison from a query.

c. *Reference Table in a many-to-many relationship*
We had to add a table named *Writes* which only stores the foreign keys from Songs and Artists to ensure no data duplication in this many-to-many relationship.

3.2. Difficult Choices

During the design process, some challenging decisions were faced regarding some tricky participation and cardinality ratios.

a. *Cardinality of Songs, Albums, Artists with their corresponding Chart and Pop tables*
In the AlbumChart and AlbumPop table, there are many entries for a single Album. However, in the beginning, we did not have any auto-generated keys and so ended up with a many-to-many relationship. However, while reflecting on feedback later, we realized the significance of a serial keys as the primary key of these tables and landed into an appropriate one-to-many relationship. The same situation happened with ArtistChart, ArtistPop, SongChart and SongPop tables.

b. *Participation ratio between Songs and Albums*
We processed our database in a way such that an Artist cannot exist in the database without releasing an album or writing a song. Hence, we had a total participation between '*Albums and Artists*', and '*Songs and Artists*'. We also assumed to have a total participation between Albums and Songs considering the fact that a song is always a part of an album. However, as we progressed through our project, we realized that there can be songs without an album and a total participation eliminates a good number of records from our database. Therefore, we made the relationship as a partial participation with consideration that an album can still be in the database with missing its songs and a song can be written without assigning it to an album or with a missing album entry. This change made our database more consistent and prevented losing most of its data.

3.3. Relational Database Fit

The data model cleanly fit into the relational database paradigm, as it effectively captures the relationships between entities through primary and foreign key constraints. This relational structure enables efficient querying and manipulation of the data. This was possible due to the actions mentioned in section 3.1-2.

3.4. Regrets and Changes

While the initial model was carefully designed, there may be some aspects that could have been improved upon in hindsight. For instance,

a. *Albums with missing Songs*
All the songs of an existing album should have also existed in the database; due to not having this privilege, we could not make the *total_track* attribute derived.

b. *Two types of Release Day*
No song should have been in the database without its corresponding Album; due to not having this privilege, we ended up having two types of release day one for when an artist releases an album and another for when an artist writes a song. We did not forcefully make a total participation in this regard to ensure more consistency and data availability as mentioned in section 3.2(b). However, our model needed to be changed from our initial version as we shifted from a total participation to a partial participation between Songs and Album.

We also designed the model to have all many-to-many relationship of Songs, Albums, Artists with their corresponding Chart and Pop tables. We had to change these one-to-many relationships as per the reasons mentioned in section 3.2(a). Apart from these, we did not have major changes to our initial database design.

3.5. Alternative Modelling

a. *Primary Key Choice*
Using "*id*" and "*week*" together as the primary key in Chart tables is a viable alternative. This approach ensures uniqueness within each week. Also, employing "*id*" and "*year*" together as the primary key in Pop tables could be another option. This ensures uniqueness within each year and simplifies querying, especially for yearly trends and analysis. However, these approaches may increase complexity in querying and indexing due to composite keys.

b. *Storing Images*
Storing images associated with albums or artists could enhance the user experience as we had some image URLs provided in the original database [1]. However, it may require additional storage space and considerations for image file formats and retrieval methods.

c. *Lyrics Storage*
Storing lyrics in a separate table linked to songs or storing them as *JSON* strings could provide flexibility for longer texts and facilitate searching and analysis based on lyrical content. However, this approach may introduce complexities in maintaining data consistency and synchronization.

d. *Handling Many Attributes in Songs*
Managing many attributes in the Songs table might lead to redundancy and data integrity issues. A possible solution could involve normalizing the data by splitting attributes into separate tables, such as one for audio features and another for metadata. We could also transfer the release day to the relationship table Writes.

Given the work completed and the project's objectives and based on the pros and cons of the alternative actions mentioned above, we would still choose the current model as it effectively captures the essential aspects of the music industry and meets the requirements for analysis and exploration. However, we could handle the Songs table in a better way following some options mentioned in part (d) of this section 3.5.

4. Discussion of the Database

For our project, we utilized a **Microsoft SQL Server** database management system (DBMS) hosted on the **Uranium** platform of University of Manitoba. The database is accessed through **Aviary** Linux system of University of Manitoba, which serves as the development and execution environment for our queries and scripts.

The database connectivity is established using the JDBC (Java Database Connectivity) driver provided by Microsoft SQL Server. Specifically, we incorporated the *mssql-jdbc-11.2.0.jre11.jar* file, which corresponds to the version of the JDBC driver used for connecting to the SQL Server database.

To facilitate secure authentication and connection to the database, we stored the necessary user credentials (username and password) of one of our team members in a configuration file named *auth.cfg*. This file ensures that only authorized users can access the database and execute queries unless the file has been shared unknowingly.

The choice of Microsoft SQL Server was a project requirement for us. Additionally, the integration of Aviary and Uranium platforms provided a seamless environment for database development, testing, and deployment.

The database population process involved reading data from .csv files and generating SQL insert commands. This task was executed efficiently using batch processing with the *addBatch()* method to accumulate SQL commands and *executeBatch()* method to execute them in bulk. This approach streamlined the population process by minimizing database interactions and optimizing performance. Additionally, the batch execution ensured consistency and reliability in data insertion, enhancing the overall robustness of the database population procedure.

5. Description of Interface

5.1. Command Line UI

User interface of a program that operates primarily through text commands entered by the user via a command-line interface (CLI), rather than a graphical user interface (GUI).

Interface Functionality

i. Command Line Interface made using java.
ii. The user can enter the commands to interact with the database.
iii. h command to see the menu of valid commands.
iv. The user gets the results displayed in a tabular form for a better view and progress bar only for the repopulate command (takes time to complete execution).
v. The interface is made easy to use with allowing the user to get customized results with multiple inputs and get a result displayed in a tabular form.

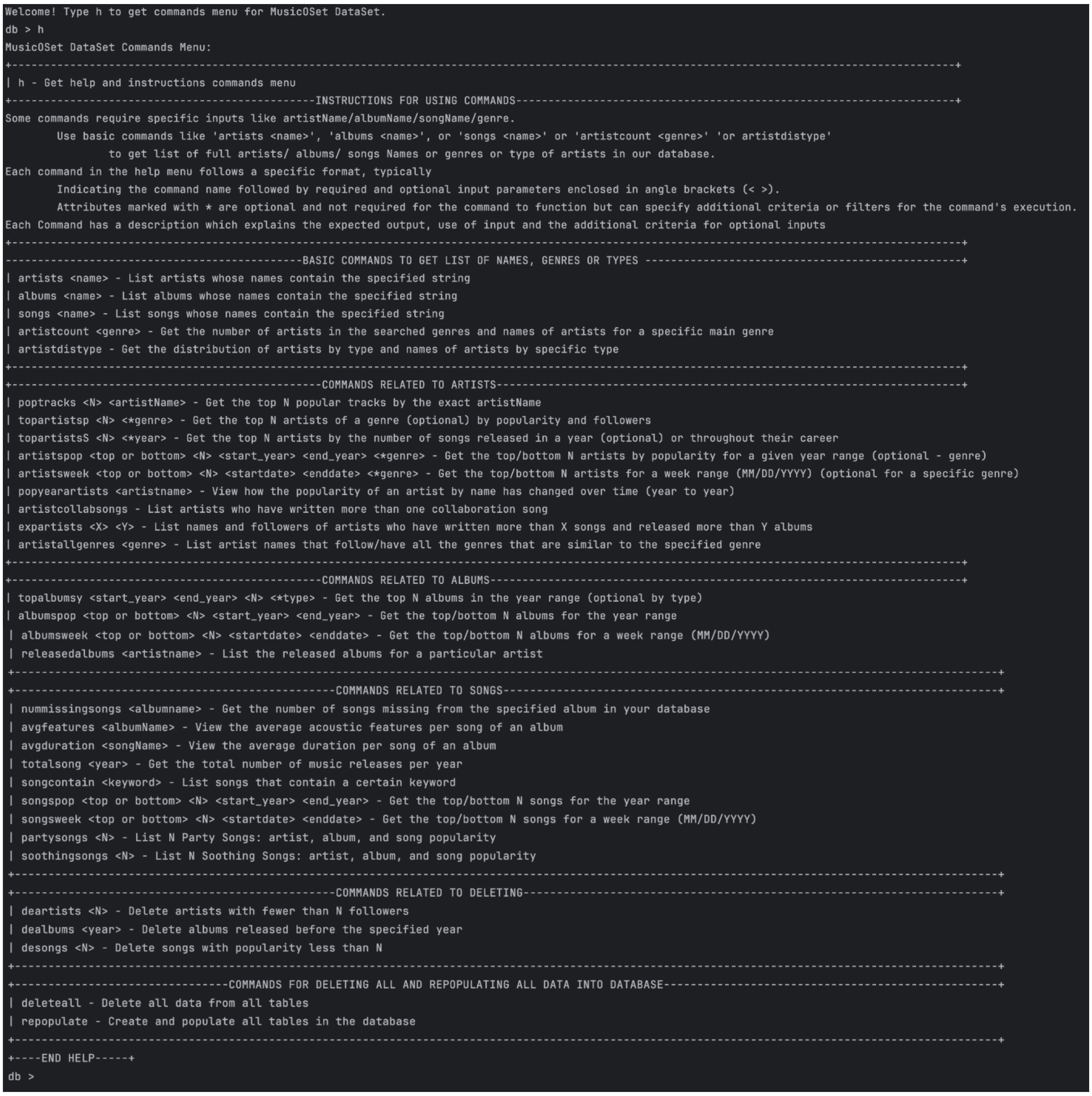
5.2. Help Menu

When the user runs the program, they can press the command h to view the help menu. This menu provides information about the various commands available within the program, along with explanations of their functionalities and the syntax required to use them.

The user must run the ***h** - Get help and instructions commands menu* command to display a menu of commands with the description of commands.

Instructions to Use the Commands

i. Some commands require specific inputs like artistName/albumName/songName/genre.
ii. Use basic commands like 'artists', 'albums', or 'songs ' or 'artistcount ' 'or artistdistype' to get list of full artists/albums/songs Names or genres or type of artists in our database.
iii. Each command in the help menu follows a specific format, typically indicating the command name followed by required and optional input parameters enclosed in angle brackets (< >).
iv. Attributes marked with * are optional and not required for the command to function but can specify additional criteria or filters for the command's execution.
v. Each Command has a description which explains the expected output, use of input and the additional criteria for optional inputs.



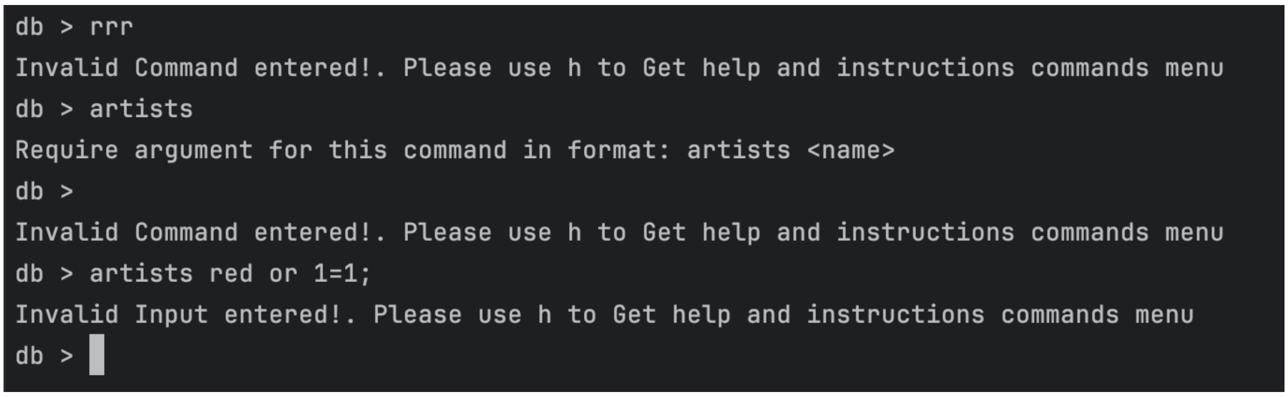
```
Welcome! Type h to get commands menu for MusicOSet DataSet.
db > h
MusicOSet DataSet Commands Menu:
+--------------------------------------------------------------------------------------------------------------------------------------------------+
| h - Get help and instructions commands menu
+-----------------------------------------------INSTRUCTIONS FOR USING COMMANDS--------------------------------------------------------------------+
Some commands require specific inputs like artistName/albumName/songName/genre.
        Use basic commands like 'artists <name>', 'albums <name>', or 'songs <name>' or 'artistcount <genre>' 'or artistdistype'
                to get list of full artists/ albums/ songs Names or genres or type of artists in our database.
Each command in the help menu follows a specific format, typically
        Indicating the command name followed by required and optional input parameters enclosed in angle brackets (< >).
        Attributes marked with * are optional and not required for the command to function but can specify additional criteria or filters for the command's execution.
Each Command has a description which explains the expected output, use of input and the additional criteria for optional inputs
+--------------------------------------------------------------------------------------------------------------------------------------------------+
---------------------------------------------BASIC COMMANDS TO GET LIST OF NAMES, GENRES OR TYPES -------------------------------------------------+
| artists <name> - List artists whose names contain the specified string
| albums <name> - List albums whose names contain the specified string
| songs <name> - List songs whose names contain the specified string
| artistcount <genre> - Get the number of artists in the searched genres and names of artists for a specific main genre
| artistdistype - Get the distribution of artists by type and names of artists by specific type
+--------------------------------------------------------------------------------------------------------------------------------------------------+
+------------------------------------------------COMMANDS RELATED TO ARTISTS----------------------------------------------------------------------+
| poptracks <N> <artistName> - Get the top N popular tracks by the exact artistName
| topartistsp <N> <*genre> - Get the top N artists of a genre (optional) by popularity and followers
| topartistsS <N> <*year> - Get the top N artists by the number of songs released in a year (optional) or throughout their career
| artistspop <top or bottom> <N> <start_year> <end_year> <*genre> - Get the top/bottom N artists by popularity for a given year range (optional - genre)
| artistsweek <top or bottom> <N> <startdate> <enddate> <*genre> - Get the top/bottom N artists for a week range (MM/DD/YYYY) (optional for a specific genre)
| popyearartists <artistname> - View how the popularity of an artist by name has changed over time (year to year)
| artistcollabsongs - List artists who have written more than one collaboration song
| expartists <X> <Y> - List names and followers of artists who have written more than X songs and released more than Y albums
| artistallgenres <genre> - List artist names that follow/have all the genres that are similar to the specified genre
+--------------------------------------------------------------------------------------------------------------------------------------------------+
+------------------------------------------------COMMANDS RELATED TO ALBUMS-----------------------------------------------------------------------+
| topalbumsy <start_year> <end_year> <N> <*type> - Get the top N albums in the year range (optional by type)
| albumspop <top or bottom> <N> <start_year> <end_year> - Get the top/bottom N albums for the year range
| albumsweek <top or bottom> <N> <startdate> <enddate> - Get the top/bottom N albums for a week range (MM/DD/YYYY)
| releasedalbums <artistname> - List the released albums for a particular artist
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
+------------------------------------------------COMMANDS RELATED TO SONGS------------------------------------------------------------------------------+
| nummissingsongs <albumname> - Get the number of songs missing from the specified album in your database
| avgfeatures <albumName> - View the average acoustic features per song of an album
| avgduration <songName> - View the average duration per song of an album
| totalsong <year> - Get the total number of music releases per year
| songcontain <keyword> - List songs that contain a certain keyword
| songspop <top or bottom> <N> <start_year> <end_year> - Get the top/bottom N songs for the year range
| songsweek <top or bottom> <N> <startdate> <enddate> - Get the top/bottom N songs for a week range (MM/DD/YYYY)
| partysongs <N> - List N Party Songs: artist, album, and song popularity
| soothingsongs <N> - List N Soothing Songs: artist, album, and song popularity
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
+------------------------------------------------COMMANDS RELATED TO DELETING---------------------------------------------------------------------------+
| deartists <N> - Delete artists with fewer than N followers
| dealbums <year> - Delete albums released before the specified year
| desongs <N> - Delete songs with popularity less than N
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
+--------------------------------COMMANDS FOR DELETING ALL AND REPOPULATING ALL DATA INTO DATABASE------------------------------------------------------+
| deleteall - Delete all data from all tables
| repopulate - Create and populate all tables in the database
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
+----END HELP-----+
db >
```

Fig-04: Help Menu in Command Line UI

5.3. Handling Invalid User Input and Preventing SQL Injection

a) *Handling invalid input*
The user gets an error message Invalid Input or Invalid Command is asked to re-enter another command when the user either enters a command that does not exist or a command with incomplete required inputs or input is not of the valid required type.

b) *Handling SQL injection*
The user gets an error message and is asked to re-enter another command when the user tries to inject SQL commands as input. This is checked by seeing if the user input contains any of the following strings "--", ";", "' ", "'''", ",", ")", ">", "=", "union select", "or select", "intersect select", "except ", "*".



```
db > rrr
Invalid Command entered!. Please use h to Get help and instructions commands menu
db > artists
Require argument for this command in format: artists <name>
db >
Invalid Command entered!. Please use h to Get help and instructions commands menu
db > artists red or 1=1;
Invalid Input entered!. Please use h to Get help and instructions commands menu
db >
```

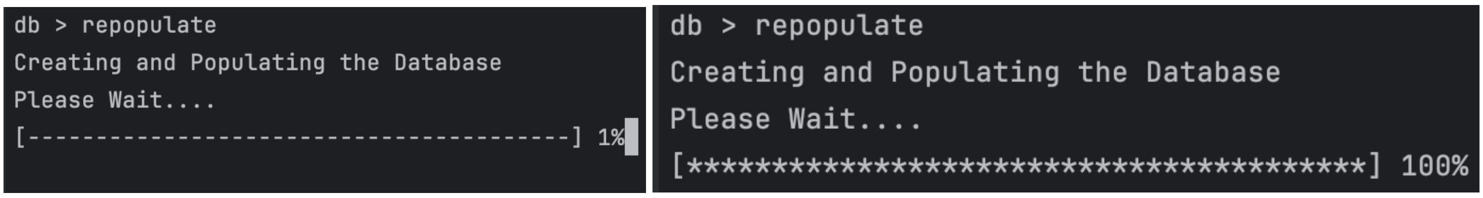
Fig-05: Examples of Handling Invalid User Input and
Preventing Injection in Command Line UI

5.4. Data Repopulation

Command to repopulate data is ***repopulate** - Create and populate all tables in the database*.

This command calls the *repopulate()* function in the code that first calls *setup()* and then calls the functions to read the .csv file in Database Files folder, add the queries to the batch and execute the batch.

Progress bar is shown to users to visually indicate the ongoing process of refreshing data, ensuring a user-friendly experience by providing clear feedback that the command is being executed.



```
db > repopulate
Creating and Populating the Database
Please Wait....
[---------------------------------------] 1%
```

```
db > repopulate
Creating and Populating the Database
Please Wait....
[***************************************] 100%
```

Fig-06: Progression for Data Repopulation in Command Line UI

6. List of Interesting Queries

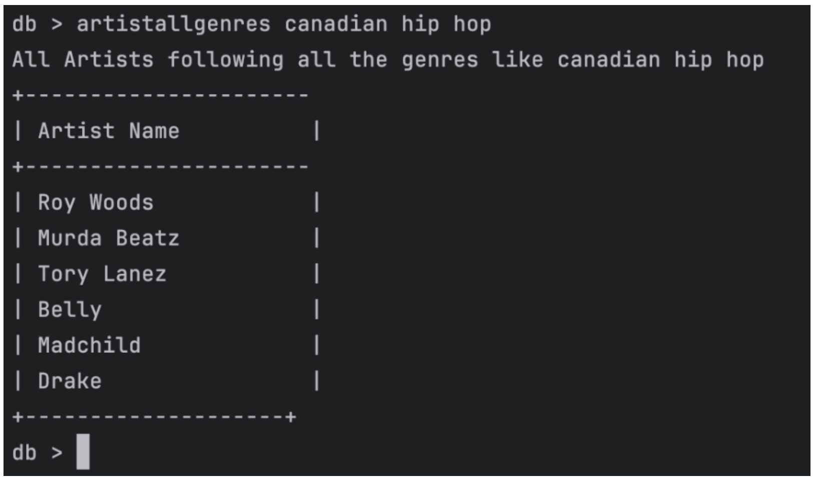
The Following queries are few of the interesting queries for our project.

6.1. artistallgenres

This query list the names of all the artists that follow all the genres which are like X using wild-card approach for searching. The query is interesting for user to know the artists who follow all varieties for a certain type of genre.

The command must be in format ***artistallgenres <genre>*** where genre is the user input.

X is user input here.



```
db > artistallgenres canadian hip hop
All Artists following all the genres like canadian hip hop
+----------------------
| Artist Name         |
+----------------------
| Roy Woods           |
| Murda Beatz         |
| Tory Lanez          |
| Belly               |
| Madchild            |
| Drake               |
+--------------------+
db >
```

Fig-07: Executing *artistallgenres* command in Command Line UI

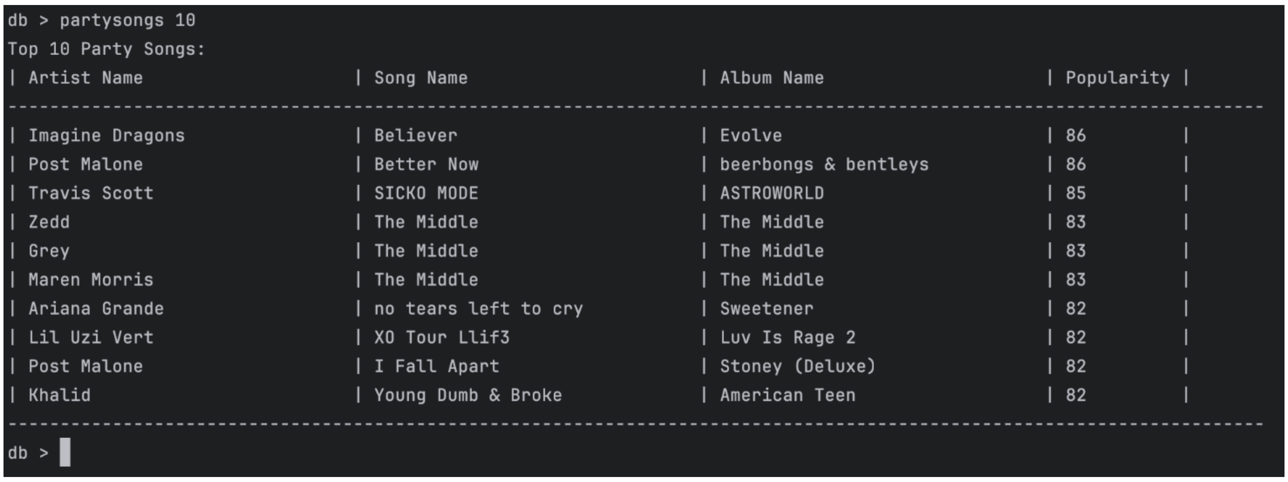
6.2. partysongs

The analyst can ask to display the N number of party Songs, their writer and the album name they were released in. Party songs usually have:

i. High danceability
ii. High energy
iii. High tempo
iv. Low acousticness
v. Positive valence
vi. Moderate to high loudness
vii. Low instrumentalness

Here we are assuming that party songs have danceability > 0.5, energy > 0.5, tempo > 100, acousticness < 0.5, valence > 0, loudness < -5 and instrumentalness < 0.01.
The user enters the number of songs to display in the list which is N.

Command Format: ***partysongs <N>*** - List N Party Songs: artist, album, and song popularity.



```
db > partysongs 10
Top 10 Party Songs:
| Artist Name                     | Song Name                       | Album Name                      | Popularity |
------------------------------------------------------------------------------------------------------------------------
| Imagine Dragons                 | Believer                        | Evolve                          | 86         |
| Post Malone                     | Better Now                      | beerbongs & bentleys            | 86         |
| Travis Scott                    | SICKO MODE                      | ASTROWORLD                      | 85         |
| Zedd                            | The Middle                      | The Middle                      | 83         |
| Grey                            | The Middle                      | The Middle                      | 83         |
| Maren Morris                    | The Middle                      | The Middle                      | 83         |
| Ariana Grande                   | no tears left to cry            | Sweetener                       | 82         |
| Lil Uzi Vert                    | XO Tour Llif3                   | Luv Is Rage 2                   | 82         |
| Post Malone                     | I Fall Apart                    | Stoney (Deluxe)                 | 82         |
| Khalid                          | Young Dumb & Broke              | American Teen                   | 82         |
------------------------------------------------------------------------------------------------------------------------
db >
```

Fig-08: Executing *partysongs* command in Command Line UI

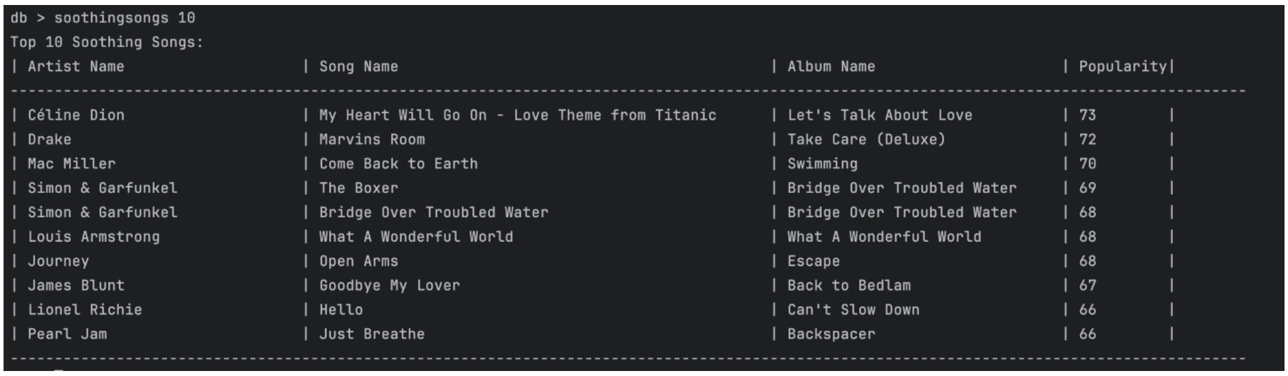
6.3. soothingsongs

The analyst can ask to display the N number of soothing Songs, their writer and the album name they were released in. Part songs usually have:

i. Low danceability
ii. Low tempo
iii. High acousticness
iv. Low energy
v. Positive valence
vi. Low loudness
vii. Low instrumentalness

Here we are assuming that party songs have danceability < 0.5, energy < 0.5, temp < < 120, acousticness > 0.5, valence > 0, loudness > -5, instrumentalness < 0.01).
The user enters the number of songs to list. Let user has entered N.

Command Format: ***soothingsongs <N>*** - List N Soothing Songs: artist, album, and song popularity.



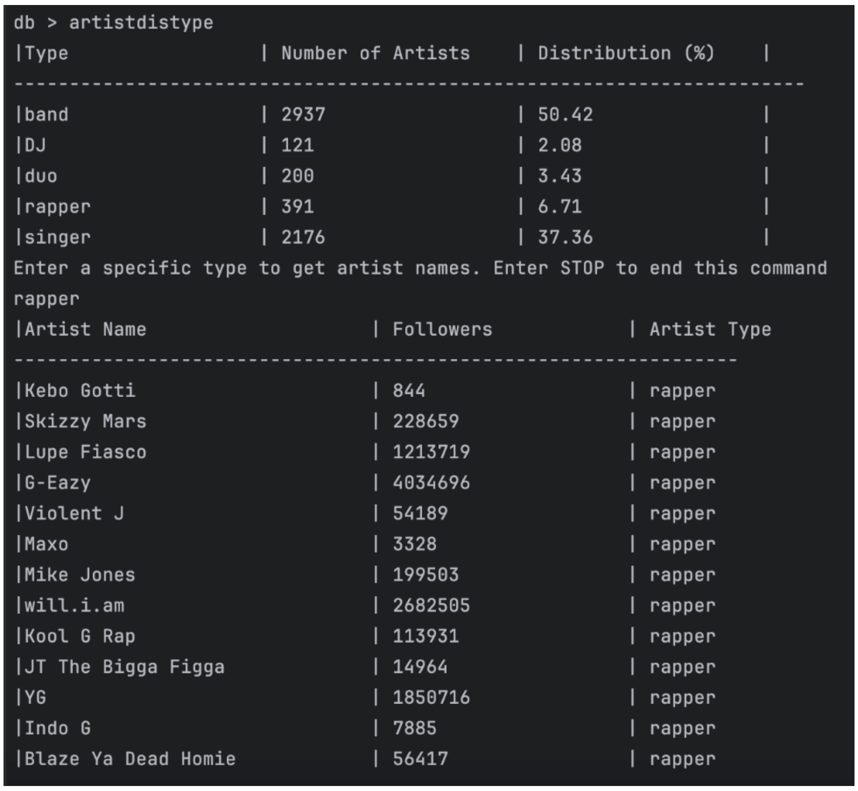
```
db > soothingsongs 10
Top 10 Soothing Songs:
| Artist Name                       | Song Name                                                 | Album Name                         | Popularity|
-------------------------------------------------------------------------------------------------------------------------------------------------
| Céline Dion                       | My Heart Will Go On - Love Theme from Titanic             | Let's Talk About Love              | 73        |
| Drake                             | Marvins Room                                              | Take Care (Deluxe)                 | 72        |
| Mac Miller                        | Come Back to Earth                                        | Swimming                           | 70        |
| Simon & Garfunkel                 | The Boxer                                                 | Bridge Over Troubled Water         | 69        |
| Simon & Garfunkel                 | Bridge Over Troubled Water                                | Bridge Over Troubled Water         | 68        |
| Louis Armstrong                   | What A Wonderful World                                    | What A Wonderful World             | 68        |
| Journey                           | Open Arms                                                 | Escape                             | 68        |
| James Blunt                       | Goodbye My Lover                                          | Back to Bedlam                     | 67        |
| Lionel Richie                     | Hello                                                     | Can't Slow Down                    | 66        |
| Pearl Jam                         | Just Breathe                                              | Backspacer                         | 66        |
-------------------------------------------------------------------------------------------------------------------------------------------------
```

Fig-09: Executing *soothingsongs* command in Command Line UI

6.4. artistdistype

The analyst can ask to see the distribution of artists according to their type, then to display the name, followers and popularity of a specific type of artist. The user can enter the type of artist from the displayed distribution list. Let the user type X on the follow-up command entry.

Command Format: ***artistdistype*** - Get the distribution of artists by type and names of artists by specific type.



```
db > artistdistype
|Type                 | Number of Artists    | Distribution (%)     |
----------------------------------------------------------------------
|band                 | 2937                 | 50.42                |
|DJ                   | 121                  | 2.08                 |
|duo                  | 200                  | 3.43                 |
|rapper               | 391                  | 6.71                 |
|singer               | 2176                 | 37.36                |
Enter a specific type to get artist names. Enter STOP to end this command
rapper
|Artist Name                    | Followers              | Artist Type
----------------------------------------------------------------
|Kebo Gotti                     | 844                    | rapper
|Skizzy Mars                    | 228659                 | rapper
|Lupe Fiasco                    | 1213719                | rapper
|G-Eazy                         | 4034696                | rapper
|Violent J                      | 54189                  | rapper
|Maxo                           | 3328                   | rapper
|Mike Jones                     | 199503                 | rapper
|will.i.am                      | 2682505                | rapper
|Kool G Rap                     | 113931                 | rapper
|JT The Bigga Figga             | 14964                  | rapper
|YG                             | 1850716                | rapper
|Indo G                         | 7885                   | rapper
|Blaze Ya Dead Homie            | 56417                  | rapper
```

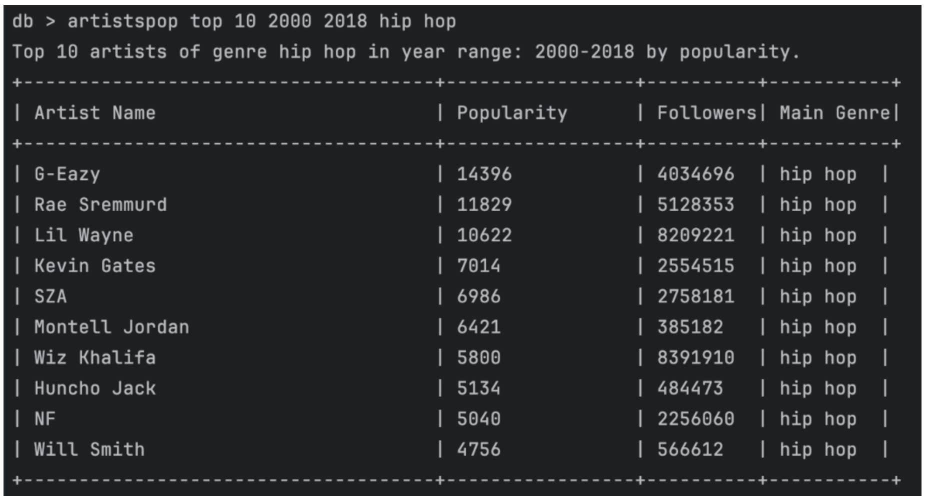
Fig-10: Executing *artistdistype* command in Command Line UI

6.5. artistspop

The analyst can ask to list N number of artists in ascending order (i.e. top) or descending order (i.e. bottom) of year end score in Artist Pop. If they specify a genre, the order will be made on only the artists that have the requested genre in their genres attribute. Otherwise, the order will be made on all the artists in the database.

Command Format: ***artistspop <top or bottom> <N> <start_year> <end_year> <*genre>*** - Get the top/bottom N artists by popularity for a given year range (optional - genre).

We have similar command for albums (*albumspop <top or bottom> <N> <start_year> <end_year> - Get the top/bottom N albums for the year range*) and songs (*songspop <top or bottom> <N> <start_year> <end_year> - Get the top/bottom N songs for the year range*).



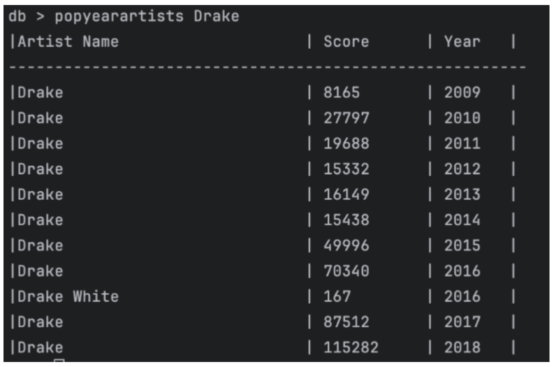
```
db > artistspop top 10 2000 2018 hip hop
Top 10 artists of genre hip hop in year range: 2000-2018 by popularity.
+-------------------------------------+-----------------+----------+----------+
| Artist Name                         | Popularity      | Followers| Main Genre|
+-------------------------------------+-----------------+----------+----------+
| G-Eazy                              | 14396           | 4034696  | hip hop  |
| Rae Sremmurd                        | 11829           | 5128353  | hip hop  |
| Lil Wayne                           | 10622           | 8209221  | hip hop  |
| Kevin Gates                         | 7014            | 2554515  | hip hop  |
| SZA                                 | 6986            | 2758181  | hip hop  |
| Montell Jordan                      | 6421            | 385182   | hip hop  |
| Wiz Khalifa                         | 5800            | 8391910  | hip hop  |
| Huncho Jack                         | 5134            | 484473   | hip hop  |
| NF                                  | 5040            | 2256060  | hip hop  |
| Will Smith                          | 4756            | 566612   | hip hop  |
+-------------------------------------+-----------------+----------+----------+
```

Fig-11: Executing *artistspop* command in Command Line UI

6.6. popyearartists

The analyst can ask how the popularity of the artists has changed year after year. The user can enter the specific artist's name. Let the user typed name be X.

Command Format: ***popyearartists <artistname>*** - View how the popularity of an artist by name has changed over time (year to year).



```
db > popyearartists Drake
|Artist Name                          | Score       | Year    |
-----------------------------------------------------------------
|Drake                                | 8165        | 2009    |
|Drake                                | 27797       | 2010    |
|Drake                                | 19688       | 2011    |
|Drake                                | 15332       | 2012    |
|Drake                                | 16149       | 2013    |
|Drake                                | 15438       | 2014    |
|Drake                                | 49996       | 2015    |
|Drake                                | 70340       | 2016    |
|Drake White                          | 167         | 2016    |
|Drake                                | 87512       | 2017    |
|Drake                                | 115282      | 2018    |
```

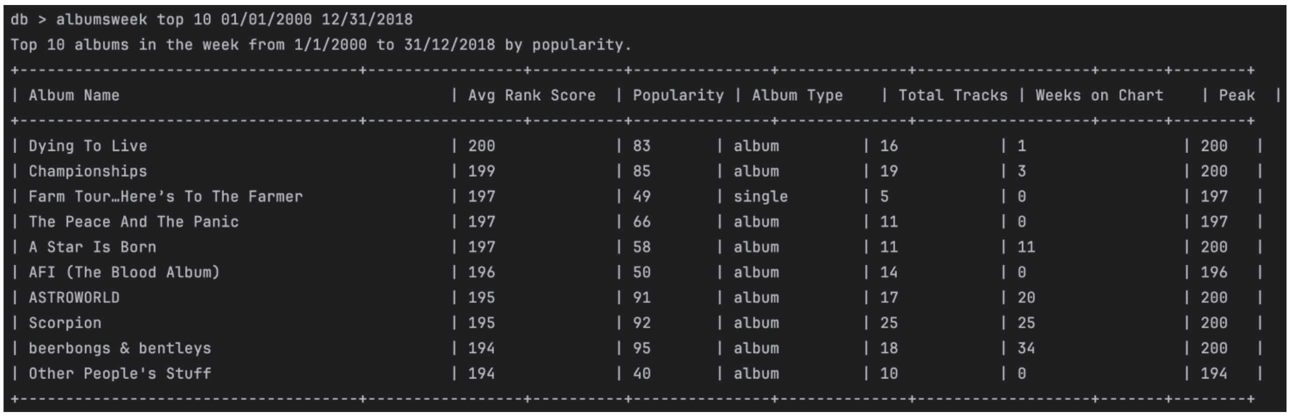
Fig-12: Executing *popyearartists* command in Command Line UI

6.7. albumsweek

The analyst can ask to list N number of albums in ascending order (i.e. top) or descending order (i.e. bottom) of the average of rank score in Album Chart that qualifies for the requested week range.

Command Format: ***albumsweek <top or bottom> <N> <startdate> <enddate>*** - Get the top/bottom N albums for a week range (MM/DD/YYYY).

We have two similar queries for artists (*artistsweek <top or bottom> <N> <startdate> <enddate> <*genre> - Get the top/bottom N artists for a week range (MM/DD/YYYY) (optional for a specific genre)*) and songs (*songsweek <top or bottom> <N> <startdate> <enddate> - Get the top/bottom N songs for a week range (MM/DD/YYYY)*).



```
db > albumsweek top 10 01/01/2000 12/31/2018
Top 10 albums in the week from 1/1/2000 to 31/12/2018 by popularity.
+------------------------------------------+-------------------+----------+-----------------+----------------+---------------------+-------+
| Album Name                               | Avg Rank Score  | Popularity | Album Type    | Total Tracks | Weeks on Chart    | Peak  |
+------------------------------------------+-------------------+----------+-----------------+----------------+---------------------+-------+
| Dying To Live                            | 200               | 83         | album           | 16             | 1                     | 200   |
| Championships                            | 199               | 85         | album           | 19             | 3                     | 200   |
| Farm Tour…Here's To The Farmer           | 197               | 49         | single          | 5              | 0                     | 197   |
| The Peace And The Panic                  | 197               | 66         | album           | 11             | 0                     | 197   |
| A Star Is Born                           | 197               | 58         | album           | 11             | 11                    | 200   |
| AFI (The Blood Album)                    | 196               | 50         | album           | 14             | 0                     | 196   |
| ASTROWORLD                               | 195               | 91         | album           | 17             | 20                    | 200   |
| Scorpion                                 | 195               | 92         | album           | 25             | 25                    | 200   |
| beerbongs & bentleys                     | 194               | 95         | album           | 18             | 34                    | 200   |
| Other People's Stuff                     | 194               | 40         | album           | 10             | 0                     | 194   |
+------------------------------------------+-------------------+----------+-----------------+----------------+---------------------+-------+
```

Fig-13: Executing *albumsweek* command in Command Line UI

7. Concluding Remarks

7.1. Relational Database System for the Dataset and Exploring Alternate Systems

For the organization and management of our dataset, which encompasses a variety of entities including songs, artists, albums, and charts, we have determined that a relational database management system (RDBMS), specifically MySQL, is most suitable. This decision is grounded in the inherently structured nature of our data, where entities are not only distinct but also interconnected through various relationships. For instance, songs are linked to artists and albums, while albums correlate with artists and potentially charts, necessitating a system that can efficiently manage and query these intricate relationships.

Relational databases, such as MySQL, excel in handling structured data and supporting complex queries that involve multiple entities and their relationships. This capability is particularly beneficial for our dataset, enabling us to perform sophisticated data analysis. For example, we can easily query the database to find all songs by a particular artist that appeared on specific charts within a given timeframe, or to retrieve all albums that contain songs that have achieved a certain ranking.

While alternative database systems like NoSQL or graph databases present certain advantages, such as scalability and flexibility for unstructured data or the efficient mapping of relationships, respectively, they were not deemed optimal for our specific requirements. NoSQL databases lack the inherent structure and complex querying capabilities for relational data. Similarly, while graph databases are excellent for mapping and exploring relationships between data points is not suitable for our dataset because we needed a database that could handle complex relational queries. Therefore, employing MySQL as our database solution offers a balance of efficiency, structure, and the ability to handle complex queries involving multiple relationships between entities. This approach ensures that we can maintain data integrity, support detailed data analysis.

7.2. Feasibility of Queries in Alternate Database System and Potential on Different or More Queries

The feasibility and ease of recreating the interesting queries mentioned, which involve complex relationships between songs, artists, albums, and charts, would vary significantly across different database systems. Each type of database relational, NoSQL, and graph has its own set of strengths and weaknesses that influence how data can be structured, stored, and queried. NoSQL databases allow for more flexible data models, which can support dynamic queries on unstructured or semi-structured data. However, recreating complex relational queries might be less straightforward.

Graph databases enable a different set of queries, particularly those involving deep relationships, patterns, or paths between data points. If we were to use a graph database, we could potentially make a query that would return a recommendation of songs depending on the input set of songs but on the other hand it would be very hard to do queries where we require the joining of large volumes of entries which is more straightforward in a relational DBMS.

7.3. Potentials as a Teaching Tool for COMP 3380 Course and its Future Students

A database featuring songs, artists, albums, and charts would serve as an excellent teaching tool by providing a rich context for practical exercises across various aspects of database design and management. Students could engage in data modelling to create an efficient, normalized schema that represents complex relationships within the music industry, honing their skills in identifying key relational database components. Writing SQL queries to analyse music trends would teach students proficiency in SQL. Assignments could explore transaction management, simulating real-world scenarios of updating charts and handling concurrent database accesses, thus emphasizing the importance of data integrity and consistency. Furthermore, students could learn about performance optimization through query tuning and index design. This multifaceted approach not only enriches students' theoretical knowledge but also equips them with practical skills crucial for navigating the complexities of database systems.

7.4. Conclusion

For the structured and relational nature of our music-related dataset, which includes entities like songs, artists, albums, and charts, a relational database management system, specifically MySQL, has been identified as the most fitting solution. This choice was motivated by the need to efficiently manage and query the complex interconnections among these entities, where MySQL's strengths in handling structured data and supporting complex relational queries come to the forefront. Despite the potential advantages of alternative systems like NoSQL and graph databases these were not deemed optimal due to their limitations in handling the kind of complex relational queries our dataset requires. The detailed data analysis and integrity maintenance facilitated by MySQL underline its suitability for our needs. Furthermore, the dataset's structure offers a rich educational resource for database design and management courses, providing students with hands-on experience in data modelling, SQL querying, and understanding transaction management and performance optimization. This comprehensive approach not only enhances theoretical knowledge but also imparts practical skills essential for navigating database systems, making it an exemplary teaching tool for courses like COMP 3380.

Appendix

Sukhmeet's Contribution

Sukhmeet played a critical role in the development and optimization of the project code to interact with the database (interface), functionality and ease of use and querying capabilities, making the program robust against invalid input and sql injection. He was instrumental in the normalization process, ensuring the database structure was efficient and logically organized, which is vital for maintaining data integrity and optimizing query performance. Additionally, Sukhmeet contributed significantly to the creation of the Entity-Relationship (ER) diagram, a crucial step in visualizing the database schema and understanding the relationships between the data entities, converting into a relational model and normalizing it. He wrote the instructions on how to create and populate the database and run your program in the readme file and prepared the final project submissions. His work on the query and interface aspects facilitated smooth interaction with the database, allowing for efficient data retrieval and manipulation.

Sudipta's Contribution

Sudipta focused on the essential tasks of data cleaning and formulating complex queries, a foundational work that guaranteed the data's quality and usability within the database. His efforts in data cleaning were paramount in ensuring accuracy and consistency across the dataset, which, in turn, enhanced the reliability of query results. Beyond this, Sudipta was involved in developing the ER diagram, providing a clear blueprint of the database structure and relationships. His contribution to the project's documentation further enriched the resource pool, offering detailed insights into the database schema, query examples, and guidelines for future users, thereby ensuring the project's longevity and ease of use.

Rishamdeep's Contribution

Rishamdeep's contributions were pivotal to the foundational aspects and overall integrity of the project. He commenced his involvement by identifying and securing the dataset, a crucial step that determined the project's scope and potential. Following this, Rishamdeep undertook the critical task of designing the database, a process that involved detailed planning to ensure the structure would efficiently support the dataset's complexity and the intended queries. His role also extended to populating the database, where he used batches to transfer data into the newly created allowing efficient data transfer to the database while also ensuring accuracy and consistency in the dataset's representation within the database. He played a central role in crafting the ER diagram and applied normalization principles to the database, a critical step for reducing redundancy and enhancing the integrity and efficiency of the database operations.

References

[1] .csv Tables: musicoset_metadata.zip (Contains textual and numeric information about songs, artists, and albums), musicoset_popularity.zip (Contains nine tables of musical popularity information) and musicoset_songfeatures.zip (Contains lyrics and acoustic fingerprints of the songs collected). Available: [Online] https://marianaossilva.github.io/DSW2019/index.html#tables.

[2] MusicOSet, An Enhanced Music Dataset for Music Data Mining. Available: [Online] https://marianaossilva.github.io/DSW2019/index.html.