Top Spotify Tracks Database

Jason Caballes
jasoncab, SeaQwel (Team 9)
CSE 560 University at Buffalo
Buffalo (NY), USA
jasoncab@buffalo.edu

Nick Cruz
nacruz, SeaQwel (Team 9)
CSE 560 University at Buffalo
Buffalo (NY), USA
nacruz@buffalo.edu

Utkarsh Mathur
umathur, SeaQwel (Team 9)
CSE 560 University at Buffalo
Buffalo (NY), USA
umathur@buffalo.edu

Abstract—

I. INTRODUCTION

A. *Task 1: Problem Statement*

Our database system utilizes the Spotify API [**?**] to store the platform's Top 50 tracks on a daily basis. The database works as a record of tracks that are currently charting across the globe. Because the system collects records daily, we prefer a database over a simple Excel file to organize the data. Furthermore, we may only be concerned with metadata related to a track, but not the track itself. A database implementation allows us to separate attributes like artist data, album data, etc. into a more organized, legible format.

B. *Spotify Web API [?]*

In 2017, Spotify made its Web API publicly available to aid developers and researchers by providing a wide range of functionalities. Using it, one can easily search for Spotify content; retrieve spotify metadata for artists, albums, shows, and tracks; control and interact with the playback to play and resume, seek to a position, or retrieve queue. Since its launch, Spotify API has been used by developers and researchers alike.

C. *Target Users*

The primary users of this database are any music enthusiast, workers in the music industry, or any Spotify user that would like to gain more insight about songs that Spotify deems to believe are the top 50 in the world! Workers in the music industry can utilize the data provided to understand how well artists are doing over time. Workers can then understand which artists to invest in. Music enthusiasts can understand the fanbase of their favorite artists. The artist themselves can learn about their fanbase and who they typically work with when creating music! This database would most likely be used in the backend of a web or mobile application in conjunction with Spotify's application.

II. DATABASE DESIGN

A. *Tasks 2 & 4 - Creating Database*

We store metadata related to the daily Top 50 tracks in each country which is pulled using the Spotify Web API [**?**]. Each of these playlist's URLs is contained in the contries_top_fifty.txt file (refer to the source code). Our script sends a request to the spotify API and pulls each song's information for each of the playlist. Any entity, i.e. (Artist, Playlist, Album, Track) is an object in the form of a python dictionary. The scripts parses the data so that we can obtain tuples to populate our tables. The one issue that arised is that these objects would create non-atomic values and therefore violate 2NF. For example, an album could have multiple artists featured in it, therefore we needed to create tuples for each artist in the album. Our script will create atomic tuples for each object that has an attribute containing multiple values. Once tuples are created, we use embedded SQL to execute our insert queries.

scripts.data_pull_functions.py is a python script that contains a function to pull the data for each relation we have in our database. To upload our data we use scripts.insert_data.py. Each function in scripts.insert_data.py uses the data created in the functions of scripts.data_pull_functions.py as an argument to upload the data to our database.

B. *Task 3, 6, 7 - Database Design & Functional Dependencies*

Our database intends to store metadata of daily Top 50 songs in each of 67 countries where Spotify is available. This means that our database would have 3350 tracks at max however the properties of these songs can prove extremely useful to researchers and music enthusiasts so designing an efficient database is required. In order to do so, we have created a database with 10 table which are interconnected through foreign keys as shown in the E/R Diagram (Fig. 1). Each of the table is in Boyce-Codd Normal Form (BCNF) owing to the Functional Dependencies associated with each table as shown below.

a. *AlbumArtists Table:*

This table stores the information about albums and the contributing artists. Since all the albums and artists have unique IDs the table only stores these IDs.

AlbumID $\rightarrow$ ArtistID

b. *Albums Table:*

This table stores the information regarding albums of the tracks in top 50 playlists. The table comprises of 4 attributes, i.e., AlbumID (unique ID for each album), AlbumName, TotalTracks (Total number of tracks in the album regardless their appearance in our database), and DateReleased (The date this album was released in YYYY-MM-DD format).
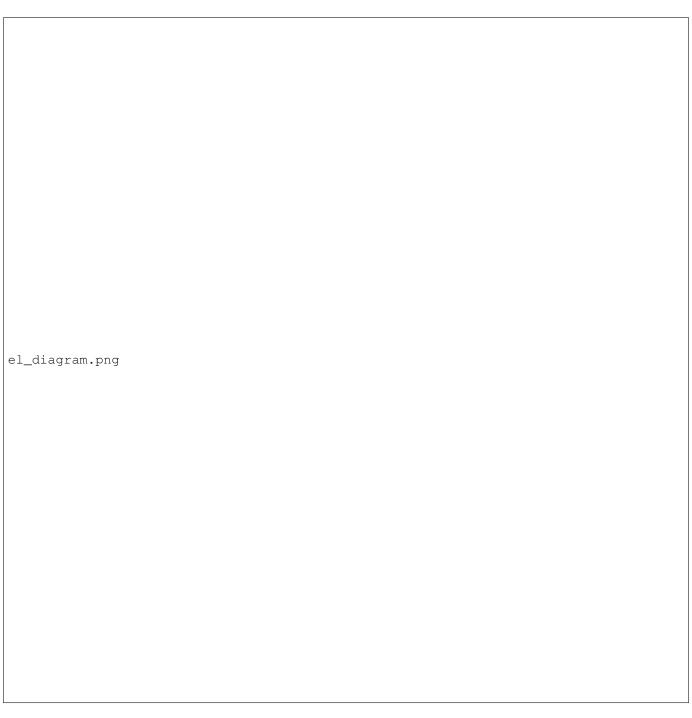
el_diagram.png

Fig. 1. ER Diagram for our database

AlbumID → (AlbumName, TotalTracks, DateReleased)

c. *Artists* Table:

This table store information regarding artists features in the top 50 charts across the 67 countries. The table schema has 4 attributes - ArtistID (unique of each artist), ArtistName, NumberOfFollowers, and Popularity (a statistic designed by Spotify measuring the popularity of artists on the scale of 0-100).

ArtistID → (ArtistName, NumberOfFollowers, Popularity)

d. *GenreArtists* Table:

This table stores the details regarding the genres of songs produced by artists by populating two atributes - ArtistID and Genre.

ArtistID → Genre

e. *KeySignatures* Table:

This table stores key signature information and it is one of the 2 tables that is pre-populated. Each song has a base key around which it's music is constructed. As there is a fixed set of keys in western music notation, it is convenient to design this table.

KeyID → KeySignature

f. *MarketCodes* Table:

This table stores information countries regarding by assigning each Country a unique CountryCode. Fixing these CountryCodes reduced the time required to populate the table and hence this table is also pre-populated like **KeySignatures** table.

CountryCodes → Country

g. *TrackAvailableMarkets* Table:

This table stores the information of countries in which a particular track is available to stream on Spotify.

TrackID → CountryCodes

h. *Playlists* Table:

This table is one of the most crucial tables for our database as it stores the information of playlists and their tracks. It has 5 attributes - Name (Playlist Name), TrackID (unique ID for tracks), TrackName (Name of the song), AlbumID (ID of the album of the track), and AlbumName (Name of the album of the track). A pair attributes [Name and TrackID] forms the primary key as it is convenient to define all the playlists and their contents.

(Name, TrackID) → (TrackName, AlbumID, AlbumName)

i. *TrackArtists* Table:

This table stores the information of tracks and their corresponding by populating two attributes - Track ID (unique ID for each song) and ArtistID (unique ID assigned to each Artist).

TrackID → ArtistID

j. *Tracks* Table:

This table is the main table where store all the metadata associated with a tracks in out playlist. This table has 14 attributes - TrackID (unique ID assigned to each song in the playlist), TrackName (name of the song), AlbumID (unique ID for the album of the song), Duration (the length of the track in miliseconds), Explicit (boolean value indicating global availability of the track across markets), CurrentPopularity (the popularity of the song on the current day), Accousticness (a metric generated by Spotify to measure the extent of use of accoustic instruments in the song), Dancebility (a metric generated by Spotify to measure the preference of song to dance upon), Intrumentalness (a metric generated by Spotify to measure the extent of use of instruments in the song), KeySignature (stores the information regarding the base key of the song), Mode, Tempo, TimeSignature, and Valence.

TrackID → (TrackName, AlbumID, Duration, Explicit, CurrentPopularity, Accousticness, Danceability, Instrumentalness, KeySignature, Mode, Temp0, TimeSignature, Valence)

III. DATABASE ANALYSIS

A. *Task 5, 8, & 9 - Large Dataset Test Queries*

We created a variety of queries that may arise in real-world applications. These queries involve a variety of clauses using where conditions, joins, aggregation functions on groups, partitions, ranking, and ordering. These queries exhibit the use-cases of our database and how researchers can leverage our database for further analysis.

1: **Find artists with minimum number of followers in each genre**

```
SELECT genre,
MIN(nbr_of_followers) min_followers
FROM genreartists
NATURAL JOIN artists
GROUP BY genre
```

2: **Find the number of countries charted for each track**

```
SELECT playlists.track_id,'
playlists.track_name,
COUNT(playlists.name) AS num_countries
FROM playlists
JOIN tracks ON playlists.track_id=tracks.track_id
GROUP BY (playlists.track_id,playlists.track_name)
ORDER BY num_countries DESC
```

3: Find all tracks created in 2023

```sql
SELECT * FROM tracks
JOIN albums ON tracks.album_id=albums.album_id
WHERE LEFT(ALBUMS.date_released, 4) = '2023'
```

4: Find the most popular artist for an album for every year

```sql
WITH data AS(
SELECT AA.ALBUM_ID,AA.artist_id,
artists.artist_name, albums.album_name,
CAST(LEFT(albums.date_released, 4) AS INT)
AS year_released, artists.popularity,
MAX(Artists.popularity)
OVER (PARTITION BY aa.album_id) AS
max_popularity_in_album
FROM albumartists AS AA
JOIN Albums ON AA.album_id=Albums.album_id
JOIN Artists ON AA.artist_id=Artists.artist_id)
SELECT DISTINCT album_name,
artist_name,popularity,
year_released
FROM data
WHERE popularity = max_popularity_in_album
ORDER BY year_released DESC, album_name;
```

5: Ranks artists by popularity for every year

```sql
SELECT DISTINCT artist_name,
year, popularity,
DENSE_RANK()
OVER(PARTITION BY YEAR
ORDER BY popularity DESC)
AS popularity_rank
FROM (SELECT *,
LEFT(ALBUMS.date_released, 4) AS year
FROM AlbumArtists
JOIN Albums
ON AlbumArtists.album_id=Albums.album_id
JOIN Artists ON
AlbumArtists.artist_id=Artists.artist_id)
AS album_join_artist
ORDER BY year DESC, popularity_RANK
```

6: Find all available tracks in the United States

```sql
SELECT Tracks.*
FROM Tracks
JOIN trackavailablemarkets AS markets
ON markets.track_id = Tracks.track_id
WHERE markets.country_code =
(SELECT country_code
FROM marketcodes
WHERE country = 'United_States_of_America')
ORDER BY Tracks.track_id;
```

7: Get follower count, popularity, album count, track count, genre count for each artist

```sql
WITH genres AS(
SELECT DISTINCT
artist_id,
COUNT(genre)
OVER (partition by artist_id) AS genre_count
FROM genreartists),
albums AS
(SELECT DISTINCT
artist_id,COUNT(album_id)
OVER (partition by artist_id)
AS album_count FROM albumartists),
tracks AS(
SELECT DISTINCT
artist_id ,COUNT(track_id)
OVER (partition by artist_id)
AS track_count FROM trackartists)
SELECT artists.*,album_count,
track_count,genre_count
FROM ARTISTS
JOIN albums ON
albums.artist_id = artists.artist_id
JOIN tracks ON
tracks.artist_id = artists.artist_id
JOIN genres ON
genres.artist_id = artists.artist_id
ORDER BY track_count DESC
```

8: Find the average duration of tracks for each year

```sql
WITH get_releases AS (
SELECT tracks.*,
albums.date_released
FROM tracks JOIN albums ON
albums.album_id = tracks.album_id
)
SELECT LEFT(date_released, 4)
AS year_released,
ROUND(AVG(duration_ms), 2) AS
avg_track_duration_ms
FROM get_releases
GROUP BY LEFT(date_released, 4)
ORDER BY LEFT(date_released, 4);
```

B. **Task 10** - *Query Execution Analysis*

IV. CONCLUSION

ACKNOWLEDGMENT

REFERENCES

[1] Spotify, "Web API — Spotify for Developers," developer.spotify.com.
https://developer.spotify.com/documentation/web-api