ILLINOIS
Grainger College of Engineering

Computer Science 411:

Database Systems

PlaylistD

a Spotify group music sharing application

May 06, 2020

PREPARED FOR

Abdu Alawini, Ph.D, Teaching Assistant Professor
Aravind Sankar, Teaching Assistant
Various, Teaching Assistant(s)

PREPARED BY SIERRA (TEAM 24)

Ishani Desai, Bachelor of Computer Science student
Naiyin “Robert” Jin, Master of Computer Science student
JaeEun “Jenny” Lee, Bachelor of Computer Science student
Michael J Neal, Master of Civil Engineering student

Table of Contents

Briefly describe what the project accomplished.

PlaylistD allows users to pool songs together as a group and filter those songs based on optional parameters. The app allows for the user to add songs based on their Top 50 most listened to songs or manual input of a playlist. This data is archived in a MongoDB database where it is sorted by which group that user wishes to join. Then, information is queried from these entries, additional information is added (which is stored in a SQL database) and returned to the user. Export to Spotify is enabled.

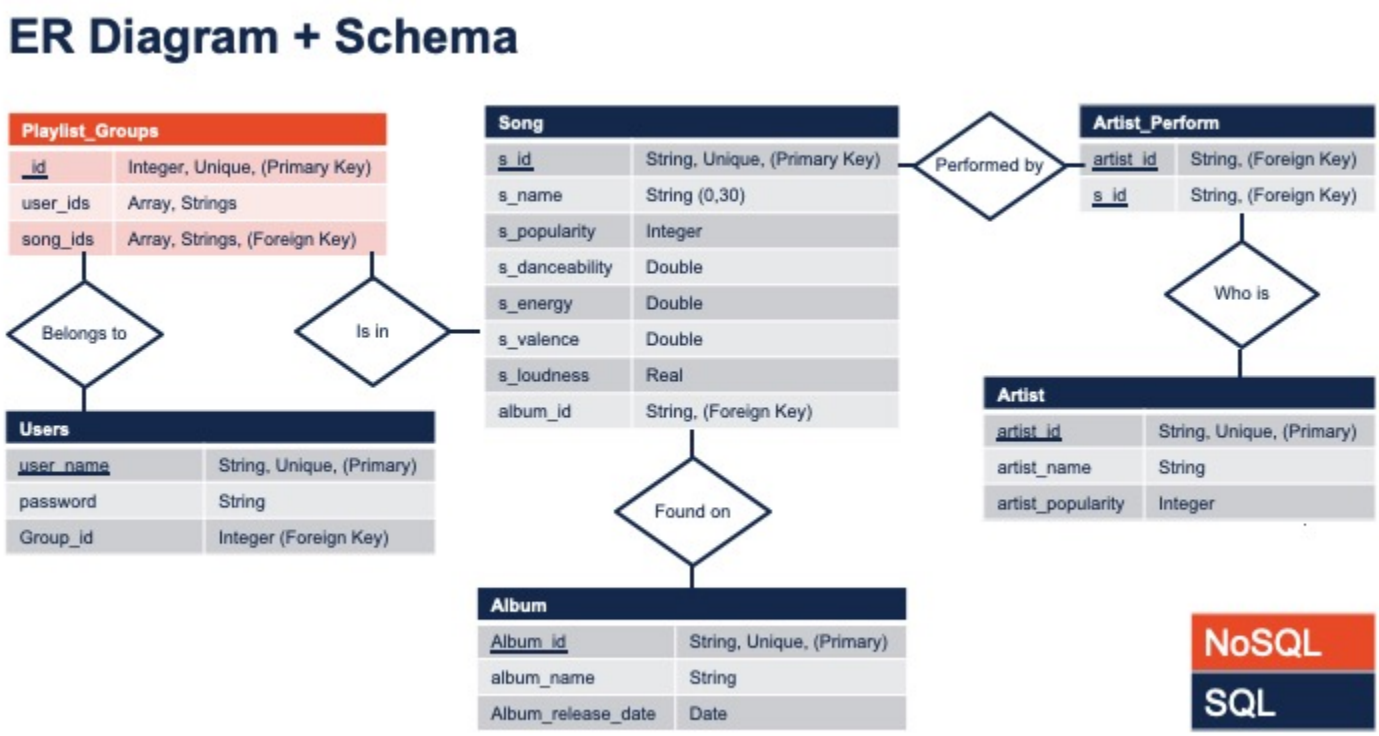
Discuss the usefulness of your project, i.e. what real problem you solved.

This application allows for a user to more readily create playlists for group settings and listen to music that their friends are in to. The filters allow for an even more tailored playlist to suit the mood.

Discuss the data in your database

One section of data is stored in MongoDB. In this single collection, each document represents a group which has a UID, an array of user ID's, and an array of song ID's. This data is cloud-based, stored with MongoDB Atlas. The rest of the data is stored in SQL format. This information consists of four tables: album, artist, artist-perform, and song. These playlists, tied together with primary/foreign keys, provide more in-depth information about each song.

Include your ER Diagram and Schema



Briefly discuss from where you collected data and how you did it.

User data is collected one of two ways: automatic crawl and direct extraction, both from Spotify API. In the first scenario, a user indicates they want to use their Top 50 songs for the group song list. Back-end code opens a Spotify authorization page and after sign-in the information is extracted to the back-end. It is then stored into MongoDB under the indicated group and user ID. If the user inputs a playlist URI, a similar process occurs but no sign-in is required. Once a song ID is listed in MongoDB it will be queried in the Spotify API and relevant data is extracted (see section 3 of this report for how that information is tabled).

Discuss how you used a NoSQL database in your project.

NoSQL, specifically MongoDB, was used for groups, user IDs and song ID storage as it allows for flexibility and the storage of arrays. Each document in MongoDB has a varying number of user/song IDs, this would require additional storage resources to store in SQL as an additional column which states group ID for each entry would be required.

Discuss your design decisions related to storing your app data in relational vs. non-relational databases.

SQL storage has been useful for the predictable, formulaic data regarding songs. Using this storage has also allowed us to separate information such that each table is a manageable and specific set of entries. MongoDB was used as group ID and name, user and song ID storage. The user and song ID information is conveniently stored in an array, making querying simple. The NoSQL mechanism works well for the varying length of these arrays.

Clearly list the functionality of your application (feature specs).

Our application allows a user to automatically create, curate, and filter a body of songs with their friends. They can manually add songs using a playlist link or have songs automatically added directly from their top liked songs. There are three filter options- popularity, danceability, and energy. In each category a user can select no filter, low, or high. The web application then displays the song, artist, and album that fits the user criteria.

Explain one basic function.

Our web application allows users to create accounts and this information is stored in a SQL database. When a user logs in, this SQL database is referred to authorize entry.

Show the actual SQL and NoSQL code snippet.

SQL:

```
SELECT DISTINCT
  s.s_id as s_id,
  s.s_name as s_name,
  q.album_name as album_name,
  q.album_release_date as album_release_date,
  f.artist_names as artist_names
FROM
  Song s,
  (SELECT ap.s_id as s_id, GROUP_CONCAT(a.artist_name, ', ') as
artist_names
  FROM Artist a, Artist_perform ap
  WHERE a.artist_ID = ap.artist_ID
  GROUP BY ap.s_id) f,
  (SELECT
    c.s_id as s_id,
```

```
    d.album_name as album_name,
    d.album_release_date as album_release_date
  FROM Song c, Album d
  WHERE c.album_id = d.album_id) q
WHERE s.s_id = f.s_id
AND s.s_id = q.s_id
AND s.s_id IN {songs}
AND s.s_popularity >= {LP}
AND s.s_popularity <= {HP}
AND s.s_danceability >= {LD}
AND s.s_danceability <= {HD}
AND s.s_energy >= {LE}
AND s.s_energy <= {HE}
ORDER BY s_id
LIMIT 10
```

NoSQL:

```
#Connection to MongoDB server
client = MongoClient('mongodb+srv://mjneal2:Bre302th%26@playlistd-
9nctl.mongodb.net/test?authSource=admin&replicaSet=Playlistd-shard-
0&readPreference=primary&appname=MongoDB%20Compass&ssl=true')
db = client['Playlistd']
pg = db.Playlist_Groups

# #Test connection
# print(db.list_collection_names())

#Extracting s_ID's from group and putting in SQL serviceable format
doc = pg.find_one({'_id': g_id})
songs_list = tuple(doc["song_ids"])
```

List and briefly explain the dataflow, i.e. the steps that occur between a user entering the data on the screen and the output that occurs

First, the user inputs their log-in information. This triggers a query to a SQL database which verifies credentials. After, a user inputs either a playlist URI or is prompted to sign into Spotify. From here, the Spotify API provides the request information- the song ID's are stored in MongoDB in the corresponding group ID-based document and all other track information is stored in various SQL tables. The program then queries the user input group ID, pulls the songs from MongoDB, stores them as a tuple, and uses that to query SQL for song information. Multiple tables are joined together in SQL and the information is returned to the script. Ultimately, this data, which now lists song names, artists, and other relevant information, is passed through our API and displayed on the front-end. An option to export to Spotify uses user input for username and playlist URI to send song_id's from front-end to Spotify API, automatically generating a playlist.

Explain your advanced function 2 (AF2) and why it's considered as advanced.

Our advanced function 1 is merging users' Spotify playlists (creating a "group playlist") based on certain criteria selected by the user(s) (energy, danceability, etc.), and creating a new playlist for them. This is advanced because it takes data from the Spotify API based on user-selected criteria, communicates with a MongoDB database to create/edit "groups", uses song IDs received from a MongoDB database to get more information about the songs from the SQL database, and runs sequel queries based on user input to get a specific list of songs for the user. Our AF1 finds songs that match all the group members' tastes based on either their top songs or a playlist of their choice, that also match the levels of energy, danceability, and popularity that they select. The front-end of the application stores the songs as a dictionary and displays the song information on the screen. The user can optionally create a playlist within their Spotify account with these songs, which the backend does by feeding the song IDs to the Spotify API.

Describe one technical challenge that the team encountered.

One technical challenge we faced was developing the front-end of the application. Writing the JavaScript code was challenging because there was a significant amount of code to write, and most of us had no experience with front-end development and/or JavaScript. We overcame the challenge of connecting our Python front-end to the back-end by using Flask. Initially setting it up was difficult because it was the first time any of us were using Flask. We were able to overcome not being familiar with JavaScript and Flask by finding online tutorials for how to make an API in Python using Flask, and how to call the API code and display it on the screen using React.

State if everything went according to the initial development plan and proposed specifications

Many things changed though the spirit of the project did not. We initially had ideas relating to creating recommendations but decided that Spotify did a good job of that already and that our algorithm for that wouldn't match up. We had some back and forth with where to use MongoDB and where to use SQL- initially considering exclusively using MongoDB. In terms of specifications, the user experience is much like we envisioned whereas the execution made practical adjustments.

Describe the final division of labor and how did you manage teamwork.

Robert Jin	Front-end, API, Spotify API
JaeEun Lee	Front-end, React
Ishani Desai	Back-end, MongoDB, Spotify API
Michael Neal	Back-end, MongoDB, SQL

Regular meetings were set, daily meetings closer to the project deadline. Half of the team focused on front-end, half on back-end. Robert orchestrated the bigger picture and ensured the ability to connect both works. Schedules were implemented but due to unknowns in the project, continual commun