Database Design for Spotify

Saloni Sarbhai
Department of Engineering Science University at Buffalo
salonisa@buffalo.edu
50495156

Shri Gayathri Chandrasekar
Department of Engineering Science University at Buffalo
shrigaya@buffalo.edu
50495167

Sree Lekshmi Prasannan
Department of Engineering Science University at Buffalo
sprasann@buffalo.edu
50495144

Introduction:

In a world where technology and music are inseparably connected, our initiative intends to combine user information with Spotify playlists, providing a unique insight into their music choices, subscriptions, and more. This approach is intended to transform music suggestions and curated playlists by adapting them to users' preferences in genres, artists, albums, and more. Throughout this paper, we will look at the system's architecture, techniques, and the significant impact it has on improving the Spotify experience. Join us as we mix data science and music to change the way we discover and enjoy music on Spotify.

Problem Statement:

As a leading music streaming network, Spotify handles huge volumes of data, such as song details, artist profiles, user preferences, and playback history. Simpler tools, such as Excel, may have sufficed at first. However, as the platform's reach and offers developed, the limitations of Excel became clear: concerns with data size restrictions, difficulties preserving data integrity, and difficulties handling real-time concurrent access. As a result, a more robust database solution is required.

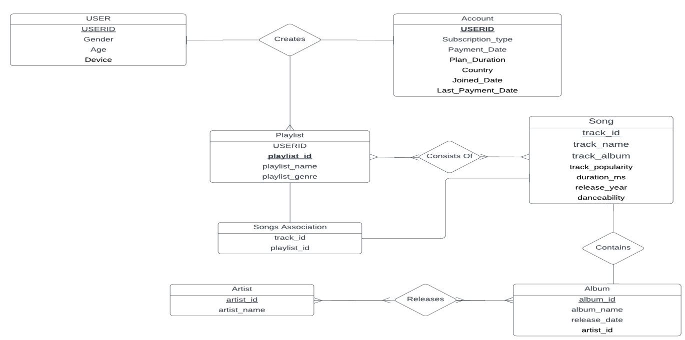
How can we ensure effective, scalable, and reliable data handling given the huge and intricate nature of Spotify's music data? Why is it critical for a music streaming behemoth like Spotify to transition from Excel to a dedicated database system?

Significance of the Problem:

Data influences user experience for a platform like Spotify. Inadequate data management can result in bad song recommendations, inaccurate artist royalties, and even system outages. Maintaining user trust and platform reliability requires seamless access to accurate data.

Real Life Scenario: Spotify launches a global campaign to promote indie artists. Data analysts require real-time access to millions of songs' streams, likes, and user feedback. Using Excel, this becomes a time-consuming operation rife with errors and delays. Spotify, on the other hand, can immediately discover trending indie music, push them to suitable users, and track the campaign's effectiveness in real-time, ensuring that new artists receive the attention they deserve.

ER Diagram:



Relational Schema:

User (USERID, Gender, Age, Device)
Account (USERID, Subscription_Type, Plan_Duration, Country, Joined_Date, Last_Payment_Date)
Playlist (USERID, playlist_id, playlist_name, playlist_genre)
Songs Association (track_id, playlist_id)
Song (track_id, track_name, track_album, track_popularity, duration_ms, release_year, danceability)

Album (album_id, album_name, release_date, artist_id)
Artist (artist_id, artist_name)

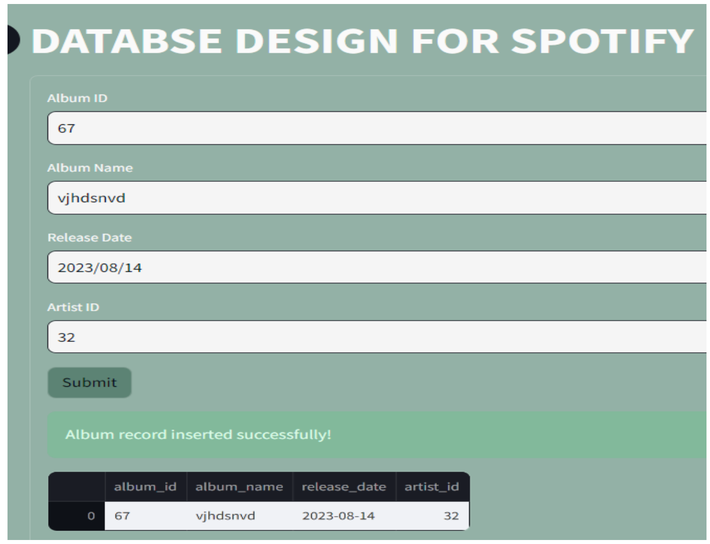
Issues Faced:

Normalization Issues: We ensured that the database is normalized to reduce redundancy and maintain data integrity was challenging. Checked if the tables were in BCNF and if not converted them into one.
Relationships Between Tables: Establishing and maintaining relationships between tables, such as foreign key constraints, was complex. It's crucial to ensure that data consistency is maintained when performing operations like inserts, updates, and deletes. To combine the track_id and playlist_id to get track_id as the primary key in the Song Table we created a separate table called Song Association.
Maintenance and Updates: The Song Association table consists of track_id and playlist_id and doesn't have any unique value. So it is difficult to update the table as we need to update the value everywhere.
Data Validation: Validating user inputs and ensuring that the data entered into the tables adhere to the defined data types and constraints is crucial. Incomplete or inconsistent data can lead to errors and affect the reliability of your database.
Integration Challenges: Integrating the database with the streamlit app was a bit challenging.
User Experience: Designing a database that supports an optimal user experience involves considering how data will be queried and displayed in the application. Balancing the need for comprehensive data with the requirement for quick and efficient queries is often challenging.

SQL Queries:

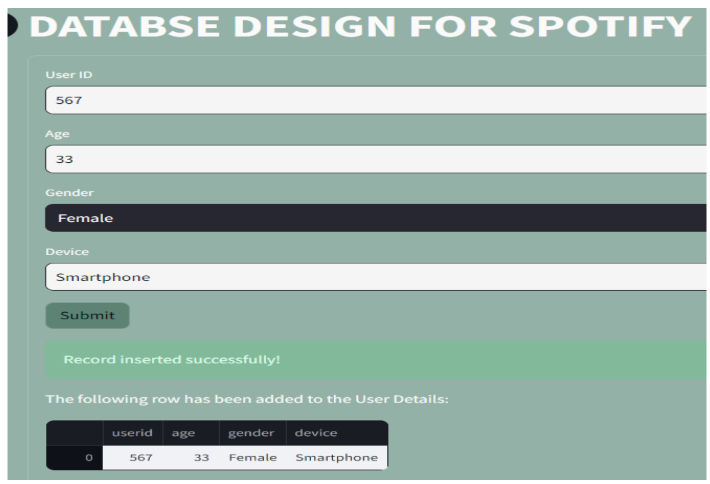
Insert Query1:

```
INSERT INTO album_table (Album_ID, Album_Name, Release_Date, Artist_ID)
VALUES (%s, %s, %s, %s)
```



Insert Query2:

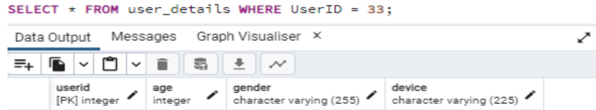
```
INSERT INTO user_details (UserID, Age, Gender, Device) VALUES (%s, %s, %s,%s)
```



Delete Queries:

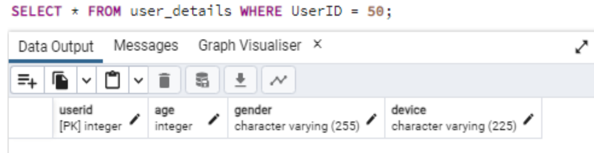
```
DELETE FROM SONG_PLAYLIST_ASSOCIATION_TABLE
WHERE Playlist_ID IN (SELECT Playlist_ID FROM playlist_table WHERE USER_ID = 33);
DELETE FROM playlist_table WHERE USER_ID = 33;
DELETE FROM account_details WHERE UserID = 33;
DELETE FROM user_details WHERE UserID = 33;

SELECT * FROM user_details WHERE UserID = 33;
```




```
DELETE FROM SONG_PLAYLIST_ASSOCIATION_TABLE
WHERE Playlist_ID IN (SELECT Playlist_ID FROM playlist_table WHERE USER_ID = 50);
DELETE FROM playlist_table WHERE USER_ID = 50;
DELETE FROM account_details WHERE UserID = 50;
DELETE FROM user_details WHERE UserID = 50;

SELECT * FROM user_details WHERE UserID = 50;
```



Update Query1:

```
UPDATE user_details SET Age = %s WHERE UserID = %s'
```

DATABSE DESIGN FOR SPOTIFY

User ID

23

New Age

67

Submit

User age updated successfully!

The updated user details are as follows:

	userid	age	gender	device
0	23	67	Female	Smart TV

Update Query2:

```
'UPDATE artist SET Artist_Name = %s WHERE Artist_ID = %s
```

DATABSE DESIGN FOR SPOTIFY

Artist ID

98

New Artist Name

Sree

Submit

Artist name updated successfully!

	artist_name	artist_id
0	Sree	98

Select queries:

1. Retrieve all the playlists that a specific user owns, you can use the following query:

```
SELECT * FROM playlist_table WHERE USER_ID = '23'
```

	playlist_id [PK] character varying (255)	playlist_name character varying (255)	playlist_genre character varying (50)	user_id integer
1	37i9dQZF1DWXrVH01e3PIE	Trapperz Argentina	rap	23
2	2YPP7fiYu5pIcp2yyHvw4A	Neo Soul	r&b	23

DATABSE DESIGN FOR SPOTIFY

Enter User ID:

23

Search Playlists

	playlist_id	playlist_name	playlist_genre	user_id
0	37i9dQZF1DWXrVH01e3PIE	Trapperz Argentina	rap	23
1	2YPP7fiYu5pIcp2yyHvw4A	Neo Soul	r&b	23

2. Retrieve the total number of songs in each playlist along with the playlist details.

```
SELECT p.Playlist_ID, p.Playlist_Name, COUNT(sp.Track_ID) as Total_Songs
FROM playlist_table p
JOIN SONG_PLAYLIST_ASSOCIATION_TABLE sp ON p.Playlist_ID = sp.Playlist_ID
GROUP BY p.Playlist_ID;
```

	playlist_id [PK] character varying (255)	playlist_name character varying (255)
1	3YouF0u7waJnolytf9JCXf	Hard Rock Workout
2	4mciQwEuqaUMwIWKcDMF...	The New (Jack Swing) Testament
3	0HD4Pc1PK8fsyKQq9e2U2v	INDIE POP! TUNES
4	4xJULuV0P5PLcMe3xP8Pgj	Southern soul & hip hop
5	37i9dQZF1DWZjqjZMudx9T	Mansión Reggaetón

DATABSE DESIGN FOR SPOTIFY

Show Total Songs in Playlists

	playlist_id	playlist_name	total_songs
0	3YouF0u7waJnolytf9JCXf	Hard Rock Workout	95
1	4mciQwEuqaUMwIWKcDMFW0	The New (Jack Swing) Testament	17
2	0HD4Pc1PK8fsyKQq9e2U2v	INDIE POP! TUNES	46
3	4xJULuV0P5PLcMe3xP8Pgj	Southern soul & hip hop	100
4	37i9dQZF1DWZjqjZMudx9T	Mansión Reggaetón	27
5	6mXh8CUBMBsBUu88a4eAQV	Pop / Dance	77

3. Retrieve the top 5 most popular songs with their details.

```
SELECT * FROM song_details_table ORDER BY TRACK_POP DESC LIMIT 5;
```

	track_id [PK] character varying (255)	track_name character varying (255)	track_album_id character varying (255)	track_pop integer	duration_ms integer	release_year integer	danceability double precision
1	2XU0oxnq2qxCpomAAuJY8K	Dance Monkey	0UywfDKYlyiu1b38DRrzYD	100	209438	2019	0.824
2	696DnlkuDOXcMAnKlTgXXK	ROXANNE	6HJDrXs0hpebaRFKA1sF90	99	163636	2019	0.621
3	7k4t7uLgtOxPwTpFmtJNTY	Tusa	7mKevNHhVnZER3BLgI8O...	98	200960	2019	0.803
4	2b8fOow8UzyDFAE27YhOZM	Memories	3nR9B40hYLKLcR0Eph3Goc	98	189486	2019	0.764
5	0sf12qNH5qcw8qpgymFOqD	Blinding Lights	2ZfHkwHuoAZrlz7RMj0PDz	98	201573	2019	0.513

DATABSE DESIGN FOR SPOTIFY

Show Top 5 Popular Songs

	track_id	track_name	track_album_id	track_pop	duration_ms	release_year	danceability
0	2XU0oxnq2qxCpomAAuJY8K	Dance Monkey	0UywfDKYlyiu1b38DRrzYD	100	209438	2019	0.824000
1	696DnlkuDOXcMAnKlTgXXK	ROXANNE	6HJDrXs0hpebaRFKA1sF90	99	163636	2019	0.621000
2	0sf12qNH5qcw8qpgymFOqD	Blinding Lights	2ZfHkwHuoAZrlz7RMj0PDz	98	201573	2019	0.513000
3	7k4t7uLgtOxPwTpFmtJNTY	Tusa	7mKevNHhVnZER3BLgI8O4F	98	200960	2019	0.803000
4	2b8fOow8UzyDFAE27YhOZM	Memories	3nR9B40hYLKLcR0Eph3Goc	98	189486	2019	0.764000
5	21jGcNKet2qwijlDFuPiPb	Circles	4g1ZRSobMefqF6nelkgibi	98	215280	2019	0.695000

4. Retrieve the average age of users for each subscription type.

```sql
SELECT a.Subscription_Type, AVG(u.Age) as Average_Age
FROM account_details a
JOIN user_details u ON a.UserID = u.UserID
GROUP BY a.Subscription_Type;
```

	subscription_type character varying (255)	average_age numeric
1	Basic	39.3707865168539326
2	Premium	38.0923076923076923
3	Standard	38.2500000000000000

DATABSE DESIGN FOR SPOTIFY

Show Average Age by Subscription

	subscription_type	average_age
0	Basic	39.3707865168539326
1	Premium	38.0923076923076923
2	Standard	38.2500000000000000

5. Retrieve the artists who have albums released in the last year and the total number of songs in each album.

```sql
SELECT ar.Artist_Name, al.Album_ID, COUNT(sd.Track_ID) as Total_Songs
FROM artist ar
JOIN album_table al ON ar.Artist_ID = al.Artist_ID
JOIN song_details_table sd ON al.Album_ID = sd.Track_Album_ID
WHERE al.Release_Date >= CURRENT_DATE - INTERVAL '5 year'
GROUP BY ar.Artist_Name, al.Album_ID;
```

	artist_name character varying (255)	album_id character varying (255)	total_songs bigint
1	Kalli	6W1oIR6Hh8K4gWwmeMe...	1
2	CLRFL	0Z3TDNVXjeTKm8KvE0F6...	1
3	Critic City	20dJlmVbsCHRdqd3lNZ3Sa	1
4	NGHTMRE	1jd4dq1xm8me7AvD2EHW...	1
5	Gregory Porter	045wnruBIjG8IPdPbJHSVh	1

DATABSE DESIGN FOR SPOTIFY

Show Artists with Recent Albums

	artist_name	album_id	total_songs
0	Gregory Porter	045wnruBIjG8IPdPbJHSVh	1
1	Finesse	0hijGWEK6AInJJjCcb5rwl	1
2	Boys Get Hurt	30jTmRcGjXt2XI7C7wZ1jS	1
3	Dan Bravo	3xyk2MJAkzgJRRrEk4k1wR	1
4	LEVZ	1hijwSTok4QONlScQpTNQZ	1
5	Os Barões Da Pisadinha	7dIuql1amYWijiQsIMps1l	1

6. Retrieve Users and Their Subscription Status:

```sql
SELECT u.UserID, u.Gender, u.Age, a.Subscription_Type
    FROM user_details u
    JOIN account_details a ON u.UserID = a.UserID;
```

	userid integer	gender character varying (255)	age integer	subscription_type character varying (255)
1	1	Male	28	Basic
2	2	Female	35	Premium
3	3	Male	42	Standard
4	4	Female	51	Standard
5	5	Male	33	Basic

DATABSE DESIGN FOR SPOTIFY

Show Users and Subscription Status

	userid	gender	age	subscription_type
0	1	Male	28	Basic
1	2	Female	35	Premium
2	3	Male	42	Standard
3	4	Female	51	Standard
4	5	Male	33	Basic

7. Identify Artists with Most Albums Released:

```sql
SELECT ar.Artist_Name, COUNT(al.Album_ID) as Total_Albums
FROM artist ar
JOIN album_table al ON ar.Artist_ID = al.Artist_ID
GROUP BY ar.Artist_Name
ORDER BY Total_Albums DESC;
```

	artist_name character varying (255)	total_albums bigint
1	Martin Garrix	72
2	Dimitri Vegas & Like Mike	64
3	Hardwell	61
4	David Guetta	51
5	The Chainsmokers	51

DATABSE DESIGN FOR SPOTIFY

Show Artists with Most Albums

	artist_name	total_albums
0	Martin Garrix	72
1	Dimitri Vegas & Like Mike	64
2	Hardwell	61
3	David Guetta	51
4	The Chainsmokers	51
5	R3HAB	47

8. Identify the Most Subscribed Country:

```sql
SELECT Country, COUNT(*) as Total_Subscriptions
FROM account_details
GROUP BY Country
ORDER BY Total_Subscriptions DESC
LIMIT 1;
```

	country character varying (255)	total_subscriptions bigint
1	Canada	23

DATABSE DESIGN FOR SPOTIFY

Show Most Subscribed Country

	country	total_subscriptions
0	Canada	23

9. Find the Latest Joined Users Without a Subscription:

```
SELECT u.UserID, u.Age, u.Gender
FROM user_details u
LEFT JOIN account_details a ON u.UserID = a.UserID
WHERE a.UserID IS NULL
ORDER BY a.Joined_Date DESC;
```

	userid [PK] integer	age integer	gender character varying (255)
1	224	30	Female

DATABSE DESIGN FOR SPOTIFY

Show Latest Joined Users Without Subscription

	userid	age	gender
0	224	30	Female

10. List Users and Their Favorite Genre (based on most added songs):

```
SELECT u.UserID, MAX(p.Playlist_Genre) as Favorite_Genre
FROM user_details u
JOIN playlist_table p ON u.UserID = p.USER_ID
JOIN SONG_PLAYLIST_ASSOCIATION_TABLE s ON p.Playlist_ID = s.Playlist_ID
GROUP BY u.UserID;
```

	userid [PK] integer	favorite_genre text
1	18	rock
2	64	rock
3	110	latin
4	178	rock
5	55	rap

DATABSE DESIGN FOR SPOTIFY

Show Users and Favorite Genre

	userid	favorite_genre
0	18	rock
1	64	rock
2	110	latin
3	178	rock
4	55	rap
5	27	rock

Query Execution Analysis:

Problematic Query 1: Retrieve the total number of songs in each playlist along with the playlist details.

Optimize query performance by addressing sequential scans on song_playlist_association_table and playlist_table through index creation on playlist_id columns. Consider switching to INNER JOIN for improved efficiency and evaluate potential gains from materialized views, especially for large datasets. Utilize EXPLAIN ANALYZE to pinpoint specific bottlenecks and fine-tune optimizations accordingly.

```
-- Explain
-- Query 2
EXPLAIN SELECT p.Playlist_ID, p.Playlist_Name, COUNT(sp.Track_ID) as Total_Songs
FROM playlist_table p
JOIN SONG_PLAYLIST_ASSOCIATION_TABLE sp ON p.Playlist_ID = sp.Playlist_ID
GROUP BY p.Playlist_ID;
```

	QUERY PLAN text
1	HashAggregate (cost=525.57..526.17 rows=60 width=1040)
2	Group Key: p.playlist_id
3	-> Hash Join (cost=11.35..475.17 rows=10080 width=1150)
4	Hash Cond: ((sp.playlist_id)::text = (p.playlist_id)::text)
5	-> Seq Scan on song_playlist_association_table sp (cost=0.00..436.80 rows=10080 width=2...
6	-> Hash (cost=10.60..10.60 rows=60 width=1032)
7	-> Seq Scan on playlist_table p (cost=0.00..10.60 rows=60 width=1032)

Problematic Query 2: Retrieve the artists who have albums released in the last year and the total number of songs in each album.

Query inefficiencies are indicated by sequential scans on song_details_table and album_table, highlighting the need for indexes on track_album_id and album_id. The hash join may pose a bottleneck, suggesting potential gains from converting to an INNER JOIN. The HashAggregate step, grouping by ar.artist_name and al.album_id, may be resource-intensive, warranting consideration of indexing. To address these issues, implement indexes, consider INNER JOIN optimization, and fine-tune with EXPLAIN ANALYZE for performance analysis.

```
-- Query 5
EXPLAIN SELECT ar.Artist_Name, al.Album_ID, COUNT(sd.Track_ID) as Total_Songs
    FROM artist ar
    JOIN album_table al ON ar.Artist_ID = al.Artist_ID
    JOIN song_details_table sd ON al.Album_ID = sd.Track_Album_ID
    WHERE al.Release_Date >= CURRENT_DATE - INTERVAL '5 year'
    GROUP BY ar.Artist_Name, al.Album_ID;
```

	QUERY PLAN text
1	HashAggregate (cost=824.30..831.23 rows=693 width=1040)
2	Group Key: ar.artist_name, al.album_id
3	-> Hash Join (cost=375.00..819.10 rows=693 width=1548)
4	Hash Cond: (al.artist_id = ar.artist_id)
5	-> Hash Join (cost=248.76..691.03 rows=693 width=1036)
6	Hash Cond: ((sd.track_album_id)::text = (al.album_id)::text)
7	-> Seq Scan on song_details_table sd (cost=0.00..436.80 rows=2080 width=10...
8	-> Hash (cost=242.46..242.46 rows=504 width=520)
9	-> Seq Scan on album_table al (cost=0.00..242.46 rows=504 width=520)
10	Filter: (release_date >= (CURRENT_DATE - '5 years'::interval))
11	-> Hash (cost=109.44..109.44 rows=1344 width=520)
12	-> Seq Scan on artist ar (cost=0.00..109.44 rows=1344 width=520)

Problematic Query 3: List Users and Their Favorite Genre (based on most added songs):

The third query's execution plan reveals potential inefficiencies with sequential scans on song_playlist_association_table and playlist_table, indicating a need for indexes on playlist_id. The hash join might be a bottleneck, suggesting gains from converting to an INNER JOIN. The HashAggregate step, grouping by u.userid, may be resource-intensive, warranting consideration of indexing. To address these issues, implement indexes, consider INNER JOIN optimization, and fine-tune with EXPLAIN ANALYZE for performance analysis.

```
EXPLAIN SELECT u.UserID, MAX(p.Playlist_Genre) as Favorite_Genre
    FROM user_details u
    JOIN playlist_table p ON u.UserID = p.USER_ID
    JOIN SONG_PLAYLIST_ASSOCIATION_TABLE s ON p.Playlist_ID = s.Playlist_ID
    GROUP BY u.UserID;
```

	QUERY PLAN text
1	HashAggregate (cost=565.23..566.03 rows=80 width=36)
2	Group Key: u.userid
3	-> Hash Join (cost=23.15..514.83 rows=10080 width=122)
4	Hash Cond: (p.user_id = u.userid)
5	-> Hash Join (cost=11.35..475.17 rows=10080 width=122)
6	Hash Cond: ((s.playlist_id)::text = (p.playlist_id)::text)
7	-> Seq Scan on song_playlist_association_table s (cost=0.00..436.80 rows=10080 width=1...
8	-> Hash (cost=10.60..10.60 rows=60 width=638)
9	-> Seq Scan on playlist_table p (cost=0.00..10.60 rows=60 width=638)
10	-> Hash (cost=10.80..10.80 rows=80 width=4)
11	-> Seq Scan on user_details u (cost=0.00..10.80 rows=80 width=4)

Future Scope:

Podcast Integration: Expand the database to support podcasts and audiobooks, catering to a broader range of audio content preferences.
Enhanced Recommendation System: Invest in machine learning algorithms to improve the recommendation system, providing more accurate and personalized music suggestions to users.
Social Features: Introduce social elements such as friend connections, shared playlists, and collaborative playlist creation to enhance user engagement and community interaction.
Offline Mode and Cross-Device Syncing: Implement offline mode, allowing users to download playlists for offline listening, and ensure seamless syncing across different devices.
User Feedback and Ratings: Incorporate a user feedback system, enabling users to rate songs and playlists, providing valuable data for refining recommendations and enhancing user experience.

Conclusion:

In summary, the database architecture devised for Spotify establishes a strong foundation for an enriched music streaming platform. The meticulously organized relational schema adeptly handles user information, subscription details, playlists, and music content. Moving forward, enhancing the user experience could involve integrating podcasts, fine-tuning the recommendation system, and introducing social functionalities. Addressing challenges like maintaining data security and privacy remains an ongoing priority. Altogether, the project shows potential in molding Spotify into a versatile and user-focused platform, ready to evolve in tandem with industry shifts and user preferences.

References:

- Music Recommendation System using Spotify Dataset | Kaggle
- Database Design for Spotify.. In this series of Database Designs, I… | by Ayush Dixit | Towards Data Engineering | Medium