a DBMS report on

‘**Music Database Management**’ on

Submitted by:-

Roll no.	Name	SAP ID
A085	Yashashri Joshi	70022200232
A087	Shreya Bhagwat	70022200234
A088	Afra Khan	70022200236

TABLE OF CONTENTS

SVKM's NMIMS
School of Technology Management & Engineering, Chandigarh
A.Y. 2023 - 24
Course: Database Management Systems
Project Report

Program	B.tech Computer Engineering	
Semester	4	
Name of the Project:	Music database management	
Details of Project Members		
Batch	Roll No.	Name
B1	A088. A087, A085	Afra K., Yashashri J., Shreya B.
Date of Submission: 02-04-2024		

Contribution of each project Members:

Roll No.	Name:	Contribution
A088	Afra Khan	SQL Queries
A085	Yashashri Joshi	Relational Model
A087	Shreya Bhagwat	ER Model

Storyline

In the bustling tech hub of Silicon Valley, a team of seasoned database architects and music aficionados converge to spearhead an ambitious project aimed at reshaping the digital music ecosystem. Under the leadership of Alex, a seasoned software engineer with a knack for disruptive innovation, they embark on the creation of SymphonyDB, a next-generation database system tailored explicitly for contemporary music streaming platforms. Fueled by their collective expertise in database design and a deep understanding of user behavior, the team meticulously engineers an architecture capable of seamlessly handling vast volumes of user data, artist metadata, and streaming analytics. From conceptualizing the Entity-Relationship (ER) model to fine-tuning relational schemas and optimizing query performance, every aspect of the project is approached with surgical precision. As they traverse the complexities of normalization and index optimization, they encounter technical hurdles and breakthroughs, each one contributing to the refinement of SymphonyDB's architecture. Armed with their ingenuity and a relentless pursuit of efficiency, they pave the way for a new era of music streaming, where data-driven insights and seamless user experiences converge to redefine the boundaries of digital audio consumption.

Components of Database Design

Entities and their attributes:

Users: user_id, name, email, password, dob

User_track: user_id, track_id, play_date

User_songs: user_id, track_id, play_date

Artist: Artist_id, name, genre

Tracks: track_id, album_id, name, duration, path

Albums: album_id, artist_id, name, released_date

Playlist: playlist_id, user_id, name

Playlist_tracks: playlist_id, track_id, order

Likes: user_id,track_id, like_date_time

Followers: user_id, artist_id

Permium_feature: premium_feature_id, name

Subscription_plan: Subscription_plan_id, name, price, description

Notification: notification_id, user_id, title, content, creation_date

Similarity: user_id, track_id, similarity_score

Relationships and Cardinality:

1. **One-to-Many relationship between Users and Playlists**: One user can have multiple playlists.

2. **One-to-Many relationship between Artists and Albums:** One artist can have multiple albums.

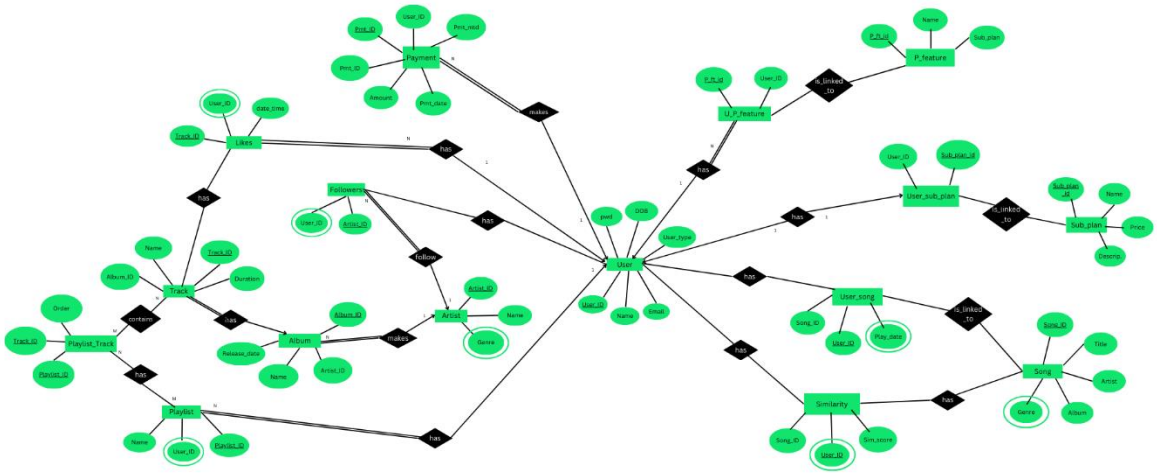
3. **One-to-Many relationship between Albums and Tracks**: One album can have multiple tracks.

4. **Many-to-Many relationship between Playlists and Tracks**: One playlist can have multiple tracks, and one track can be part of multiple playlists. The relationship is managed by the Playlist_Tracks table.

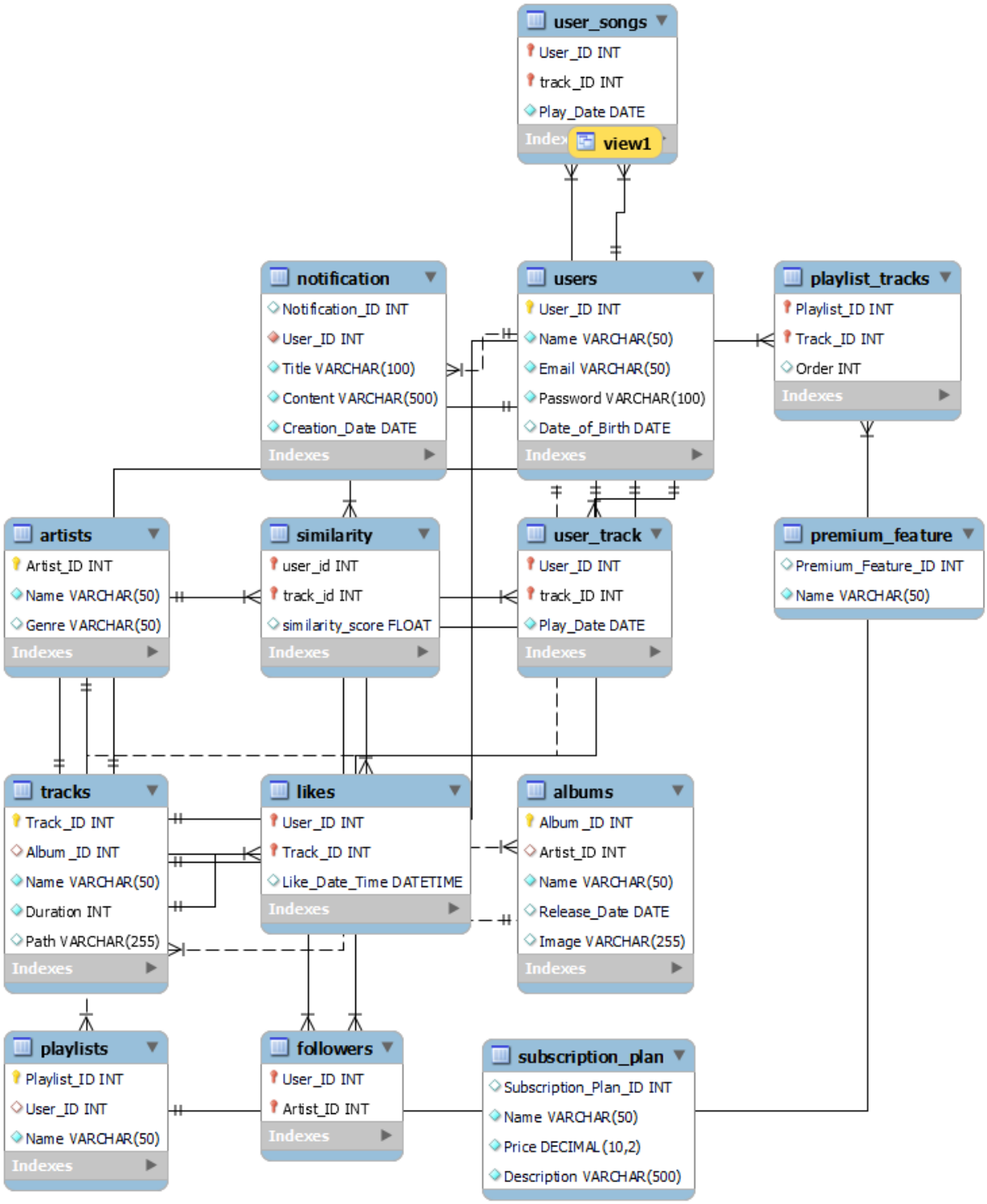
5. **Many-to-Many relationship between Users and Artists**: One user can follow multiple artists, and one artist can be followed by multiple users. The relationship is managed by the Followers table.

6. **Many-to-Many relationship between Users and Tracks**: One user can like multiple tracks, and one track can be liked by multiple users. The relationship is managed by the Likes table.

ER Model


Payment
Pmt_ID
User_ID
Pmt_mod
Amount
Pmt_date
Make
Likes
User_ID
date_time
Track_ID
has
Followers
User_ID
Artist_ID
has
follow
Track
Track_ID
Name
Album_ID
Duration
contains
has
Playlist_Track
Order
Track_ID
Playlist_ID
has
Playlist
Name
User_ID
Playlist_ID
has
Album
Album_ID
Release_date
Name
Artist_ID
makes
Artist
Artist_ID
Name
Genre
User
pwd
DOB
Real_type
User_ID
Name
Email
has
U_P_feature
P_ft_id
User_ID
has
P_feature
P_ft_id
Name
Sub_plan
is_linked_to
has
User_sub_plan
User_ID
Sub_plan_id
is_linked_to
Sub_plan
Sub_plan_id
Name
Price
Descrip
has
User_song
Song_ID
User_ID
Play_date
is_linked_to
has
Similarity
Song_ID
User_ID
Sim_score
has
Song
Song_ID
Title
Artist
Genre
Album

Relational Model



Normalization

- **Users Table**: Already in 3NF.
- **Artists Table**: Already in 3NF.

- **Albums Table**: Artist_ID needs to be not null because every album should be associated with an artist.

- **Tracks Table**: Album_ID needs to be not null because every track should be associated with an album.

- **Playlists Table**: Already in 3NF.
- Playlist_Tracks Table: This is a junction table for a many-to-many relationship and is already in 3NF.

- **Followers Table**:This is a junction table for a many-to-many relationship and is already in 3NF.

- **Likes Table**:This is a junction table for a many-to-many relationship and is already in 3NF.

SQL Queries

#Retrieve the name of an artist and the name of all tracks in an album:

```
SELECT Artists.Name AS Artist_Name, Albums.Name AS Album_Name, Tracks.Name AS Track_Name

FROM Artists

JOIN Albums ON Artists.Artist_ID = Albums.Artist_ID

JOIN Tracks ON Albums.Album_ID = Tracks.Album_ID;
```

#Retrieve the name of a user, the name of a playlist, and all tracks in the playlist:

```
SELECT Users.Name AS User_Name, Playlists.Name AS Playlists_Name, Tracks.Name AS Track_Name

FROM Users

JOIN playlists ON users.User_ID = playlists.User_ID

JOIN Playlist_Tracks ON playlists.Playlist_ID = Playlist_Tracks.Playlist_ID

JOIN tracks ON Playlist_Tracks.Track_ID = tracks.Track_ID;
```

#Retrieve the name of a user and all tracks they have liked:

```
SELECT Users.Name AS User_Name, Tracks.Name AS Track_Name

FROM Users

JOIN Likes ON Users.User_ID = Likes.User_ID

JOIN Tracks ON Likes.Track_ID = Tracks.Track_ID;
```

#Retrieve the name of a user and all tracks in a playlist, but only for playlists that contain more than 10 tracks:

```
SELECT Users.Name AS User_Name, Playlists.Name AS Playlist_Name, COUNT(Tracks.Track_ID) AS Track_Count

FROM Users

JOIN Playlists ON Users.User_ID = Playlists.User_ID

JOIN Playlist_Tracks ON playlists.Playlist_ID = Playlist_Tracks.Playlist_ID

JOIN tracks ON Playlist_Tracks.Track_ID = tracks.Track_ID

GROUP BY users.Name, playlists.Name

HAVING COUNT(tracks.Track_ID) > 10;
```

#Retrieve the name of a user and all artists they are following, but only for users who are following more than 5 artists:

```
SELECT users.Name AS User_Name, COUNT(Artists.Artist_ID) AS Artist_Count

FROM users

JOIN followers ON users.User_ID = followers.User_ID

JOIN Artists ON followers.Artist_ID = Artists.Artist_ID

GROUP BY users.Name

HAVING COUNT(Artists.Artist_ID) > 1;
```

#top 5 most liked tracks:

```
SELECT t.Track_ID, t.Name AS Track_Name, COUNT(l.Track_ID) AS
Likes_Count

FROM Tracks t

LEFT JOIN Likes l ON t.Track_ID = l.Track_ID

GROUP BY t.Track_ID

ORDER BY Likes_Count DESC

LIMIT 5;


#total duration of all tracks in each playlist:

SELECT pt.Playlist_ID, p.Name AS Playlist_Name, SUM(t.Duration) AS
Total_Duration

FROM Playlist_Tracks pt

JOIN Tracks t ON pt.Track_ID = t.Track_ID

JOIN Playlists p ON pt.Playlist_ID = p.Playlist_ID

GROUP BY pt.Playlist_ID;


#artists with the most followers:

SELECT ar.Artist_ID, ar.Name AS Artist_Name, COUNT(f.User_ID) AS
Follower_Count

FROM Artists ar

LEFT JOIN Followers f ON ar.Artist_ID = f.Artist_ID

GROUP BY ar.Artist_ID

ORDER BY Follower_Count DESC

LIMIT 3;
```

#playlists that contain tracks from multiple genres:

```
SELECT pt.Playlist_ID, p.Name AS Playlist_Name
FROM Playlist_Tracks pt
JOIN Tracks t ON pt.Track_ID = t.Track_ID
JOIN Albums al ON t.Album_ID = al.Album_ID
JOIN Artists ar ON al.Artist_ID = ar.Artist_ID
JOIN Playlists p ON pt.Playlist_ID = p.Playlist_ID
GROUP BY pt.Playlist_ID, p.Name
HAVING COUNT(DISTINCT ar.Genre) > 1;
```

#playlists that contain tracks from same genres

```
SELECT pt.Playlist_ID, p.Name AS Playlist_Name
FROM Playlist_Tracks pt
JOIN Tracks t ON pt.Track_ID = t.Track_ID
JOIN Albums al ON t.Album_ID = al.Album_ID
JOIN Artists ar ON al.Artist_ID = ar.Artist_ID
JOIN Playlists p ON pt.Playlist_ID = p.Playlist_ID
GROUP BY pt.Playlist_ID, p.Name
HAVING COUNT(DISTINCT ar.Genre) > 1;
```

#average duration of tracks in each album:

```
SELECT a.Album_ID, a.Name AS Album_Name, AVG(t.Duration) AS Average_Duration

FROM Albums a

JOIN Tracks t ON a.Album_ID = t.Album_ID

GROUP BY a.Album_ID, a.Name;


#users who have not created playlists or liked any tracks:

SELECT u.User_ID, u.Name AS User_Name

FROM Users u

LEFT JOIN Playlists p ON u.User_ID = p.User_ID

LEFT JOIN Likes l ON u.User_ID = l.User_ID

WHERE p.User_ID IS NULL AND l.User_ID IS NULL;


#the most recent release from each artist:

SELECT ar.Artist_ID, ar.Name AS Artist_Name, MAX(al.Release_Date) AS Most_Recent_Release_Date

FROM Artists ar

JOIN Albums al ON ar.Artist_ID = al.Artist_ID

GROUP BY ar.Artist_ID, ar.Name;


#users who have liked all tracks from a specific album:

SELECT l.User_ID, u.Name AS User_Name

FROM Likes l
```

```
JOIN Tracks t ON l.Track_ID = t.Track_ID

JOIN Albums al ON t.Album_ID = al.Album_ID

JOIN Users u ON l.User_ID = u.User_ID

WHERE al.Album_ID = 1

GROUP BY l.User_ID, u.Name

HAVING COUNT(DISTINCT t.Track_ID) = (SELECT COUNT(*) FROM
Tracks WHERE Album_ID = 1);


#artists with the most tracks in the system:

SELECT ar.Artist_ID, ar.Name AS Artist_Name, COUNT(t.Track_ID) AS
Track_Count

FROM Artists ar

JOIN Albums al ON ar.Artist_ID = al.Artist_ID

JOIN Tracks t ON al.Album_ID = t.Album_ID

GROUP BY ar.Artist_ID, ar.Name

ORDER BY Track_Count DESC;
```

Learnings Achieved

Database Design: Understanding how to design a relational database schema to model complex data structures such as a music streaming service, including tables for users, artists, albums, tracks, playlists, likes, and followers.

SQL Skills: Gaining proficiency in writing complex SQL queries to retrieve, manipulate, and analyze data stored in the database. This includes using various SQL clauses such as SELECT, JOIN, GROUP BY, HAVING, and ORDER BY to perform operations like filtering, aggregating, and sorting data.

Data Modeling: Learning how to effectively model relationships between different entities in a database using primary and foreign keys, as well as understanding normalization principles to ensure data integrity and minimize redundancy.

Practical Application: Applying database and SQL concepts to a real-world project scenario, which provides hands-on experience and helps reinforce theoretical knowledge.

Problem-Solving: Overcoming challenges encountered during the project, such as debugging queries, optimizing database performance, and handling data inconsistencies, which enhances problem-solving skills.

Collaboration: Working collaboratively with team members, such as developers, designers, and stakeholders, to understand requirements, integrate components, and ensure the project meets objectives, fostering teamwork and communication skills.

Documentation: Documenting the database schema, query logic, and project specifications, which improves clarity, facilitates knowledge sharing, and serves as a valuable resource for future reference.

Challenges faced

Determining multivalued attributes: A lot of time was spent in determining which attributes possess multiple values.

Deciding the primary keys of the entities: Many a times we found ourselves confused in deciding which attribute needs to be the primary key.

Normalization: Creating FD’s from the relational model and then applying normalization was a tough task.

Conclusion

In conclusion, the development of a comprehensive database system for a music streaming platform represents a significant endeavor in harnessing technology to enhance the user experience and streamline operations. Through the creation of an Entity-Relationship (ER) model, its translation into a relational model, and subsequent normalization processes, we have established a solid foundation for efficient data management and retrieval. The formulation and execution of essential queries further demonstrate the database's capability to extract valuable insights and support various functionalities crucial for the platform's success. As technology continues to evolve and user preferences shift, ongoing optimization and adaptation of the database will be imperative to ensure scalability, performance, and user satisfaction. This project underscores the importance of robust database design principles in powering modern digital services, paving the way for continued innovation and advancement in the realm of music streaming and beyond.

References

https://medium.com/towards-data-engineering/design-the-database-for-a-system-like-spotify-95ffd1fb5927