MELODIFY

Project by ,

Manasi Jog , UCE2023425
Anisha Kulal , UCE2023407
Vaishnavi Ahire , UCE2023406
Sejal jinde,UCE2023424

1.Software requirement specification

Introduction

Purpose :

The purpose of this document is to define the requirements for the **Melodify** database, a music streaming service that provides a database structure to store information about users, songs, artists, albums, genres, playlists, and user interactions (such as likes and playlist management). This database will support the back-end functionality for a streaming platform similar to Spotify, enabling efficient data retrieval and user experience.

Scope :

The database will facilitate user registration, song and artist data storage, playlist creation, and activity tracking. It will support all features required by the **Melodify** application, including playlist management, song likes, and streaming history. The database will be managed using relational database management principles to ensure data integrity, consistency, and scalability

System overview

The **Melodify** database includes following key entities :

- **User:** Information on users, including details required for access control and subscriptions.
- **Song:** Information about songs available in the library.
- **Artist:** Information about artists associated with songs.

- **Album:** Information about albums containing songs.
- **Genre:** Classification of songs and artists.
- **Playlist:** Custom playlists created by users.
- **User Activity:** Tracks user interactions with songs, including likes and playlist entries.

Functional Requirements

3.1 User Management

Registration and Login

- The system should store users with unique usernames and emails.
- Each user should have an associated subscription type.

Subscription Management

- Each user should have a `SubscriptionType` (e.g., Free, Premium).
- Subscription types will determine access levels for users.

3.2 Music Management

Song Management

- The system should store each song's unique details, such as title, duration, release date, album, and genre.
- Each song can belong to one album and be associated with multiple artists.

Artist Management

- The system should store each artist's details, including name, country, and genre.
- Each artist can be associated with multiple songs and albums.

Album Management

- The system should store albums with unique titles and release dates.
- Each album can contain multiple songs but is associated with only one artist.

Genre Management

- The system should store genre details.
- Each genre can be linked to multiple songs and artists.

3.3 Playlist Management

Playlist Creation and Management

- Users can create and name their playlists.
- Each playlist belongs to a single user and can contain multiple songs.
- The system should enforce unique playlist names per user.

3.4 User Activity Tracking

Likes Management

- Users can like multiple songs.
- The system should record user-song relationships for liked songs.

Playlist-Song Management

- The system should support the addition of songs to user playlists.
- Playlists can contain multiple songs, and songs can belong to multiple playlists.

3.5 Analytics

Popular Songs and Artists

- The system should store data that enables querying the most liked songs and popular artists based on user interaction.

Non Functional Requirements

4.1 Performance Requirements

- The database should be optimized for fast read/write operations, particularly for retrieving user playlists, liked songs, and artist/song data.
- Response time for database queries should be within 1 second for most operations.

4.2 Security Requirements

- User data (e.g., passwords) must be securely stored with encryption.
- Access to the database should be controlled with appropriate user roles and permissions.

4.3 Data Integrity Requirements

- Referential integrity must be maintained across tables (e.g., each song must reference a valid album).
- Triggers should be used to enforce consistency (e.g., minimum song duration).

4.4 Scalability Requirements

- The database must be able to handle an increasing number of users, playlists, and song entries without performance degradation.

Database Design

Tables and Attributes

1. **User**
   - UserID (PK)
   - Username
   - Email
   - Password

2. **Song**
   - SongID (PK)
   - Title
   - Duration
   - ReleaseDate
   - AlbumID (FK)
   - GenreID (FK)

3. **Artist**
   - ArtistID (PK)

- Name
- Country
- GenreID (FK)

4. **Album**
   - AlbumID (PK)
   - Title
   - ReleaseDate
   - ArtistID (FK)

5. **Genre**
   - GenreID (PK)
   - GenreName

6. **Playlist**
   - PlaylistID (PK)
   - UserID (FK)
   - PlaylistName
   - CreatedDate

7. **User_Likes_Song**
   - UserID (PK, FK)
   - SongID (PK, FK)

8. **Playlist_Contains_Song**
   - PlaylistID (PK, FK)
   - SongID (PK, FK)

Use Cases

6.1 User Registers for an Account

1. The user submits their registration details (Username, Email, Password).

2. The system validates the information and creates a new record in the `User` table.

6.2 User Creates a Playlist

1. The user specifies a name for a new playlist.

2. The system checks for name uniqueness for that user and creates a new entry in the `Playlist` table.

6.3 User Likes a Song

1. The user selects a song to like.

2. The system adds a new record to the `User_Likes_Song` table, linking the user to the song.

Triggers,Stored Procedures and Functions

7.1 Triggers

- Trigger on Playlist Creation to ensure unique playlist names per user.
- Trigger on Song Insert to update the album's total duration.

7.2 Stored Procedures

- Create Playlist : A procedure to create a playlist with a list of song IDs for a user..

7.3 Functions

- Get Playlist Duration : A function to calculate the total duration of a playlist.

Constraints and Assumptions

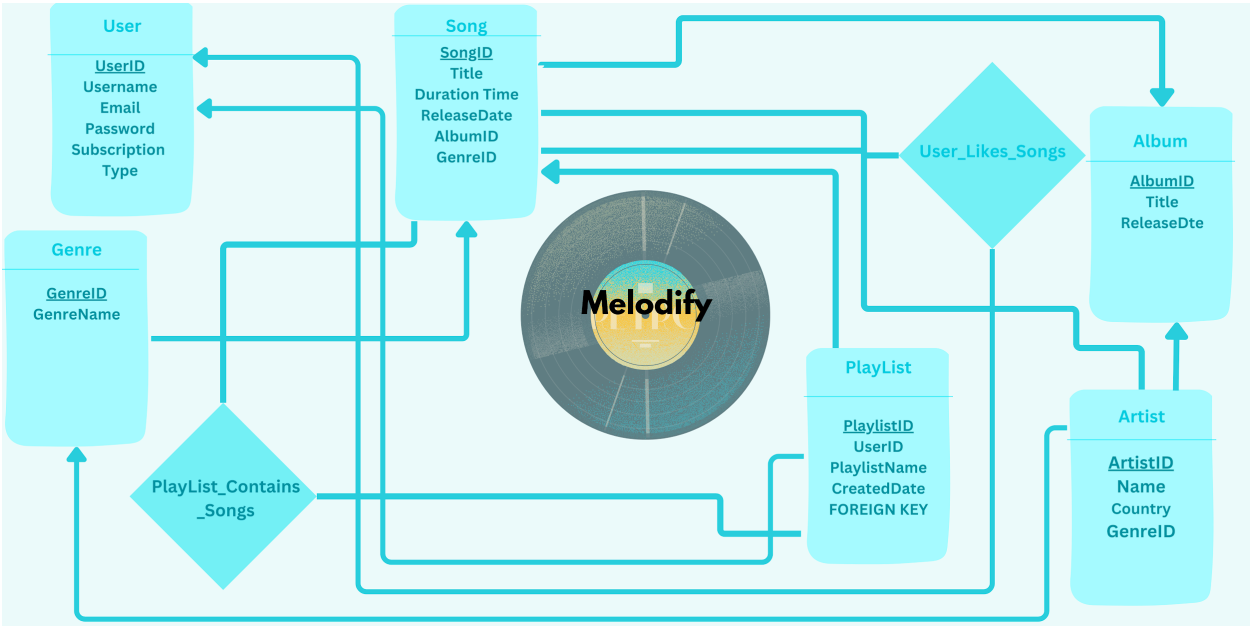
Constraints :

- Each song must have a minimum duration of 30 seconds.
- Each user must have a unique username and email.

Assumptions :

- Each song belongs to only one album.
- Each album is created by a single artist.
- The database can handle a high volume of read and write operations typical for music streaming.

2.Entity Relationship Diagram :



3. Tables Made from ERD

USER TABLE

```
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL UNIQUE,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Password VARCHAR(100) NOT NULL
```

```
);
```

SONG TABLE

```
CREATE TABLE Song (
    SongID INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Duration INT NOT NULL CHECK (Duration > 0), -- duration in seconds
    ReleaseDate DATE,
    AlbumID INT,
    GenreID INT,
    FOREIGN KEY (AlbumID) REFERENCES Album(AlbumID),
    FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
```

ARTIST TABLE

```
CREATE TABLE Artist (
    ArtistID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Country VARCHAR(50),
    GenreID INT,
    FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
```

ALBUM TABLE

```
CREATE TABLE Album (
    AlbumID INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    ReleaseDate DATE,
    ArtistID INT NOT NULL,
    FOREIGN KEY (ArtistID) REFERENCES Artist(ArtistID)
);
```

GENRE TABLE

```
CREATE TABLE Genre (
    GenreID INT PRIMARY KEY,
    GenreName VARCHAR(50) NOT NULL UNIQUE
);
```

PLAYLIST TABLE

```
CREATE TABLE Playlist (
    PlaylistID INT PRIMARY KEY,
    UserID INT NOT NULL,
    PlaylistName VARCHAR(100) NOT NULL,
    CreatedDate DATE DEFAULT CURRENT_DATE(),
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

User_Likes_Song Tables

```
CREATE TABLE User_Likes_Song (
    UserID INT,
    SongID INT,
    PRIMARY KEY (UserID, SongID),
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (SongID) REFERENCES Song(SongID)
);
```

Playlist_Contains_Song Table

```
CREATE TABLE Playlist_Contains_Song (
    PlaylistID INT,
    SongID INT,
    PRIMARY KEY (PlaylistID, SongID),
    FOREIGN KEY (PlaylistID) REFERENCES Playlist(PlaylistID),
    FOREIGN KEY (SongID) REFERENCES Song(SongID)
);
```

4.Normalization up to 3NF :

1. **User Table**

Original Attributes : `UserID (PK), Username, Email, Password, SubscriptionType`

- 1NF : All values are atomic (e.g., Username and Email are unique fields and not lists).
- 2NF : Since `UserID` is the primary key, all non-key attributes (Username, Email, Password, and SubscriptionType) fully depend on `UserID`.
- 3NF : There are no transitive dependencies, as each attribute relates directly to the primary key `UserID`.

2. **Song Table**

Original Attributes : `SongID (PK), Title, Duration, ReleaseDate, AlbumID, GenreID`

- 1NF : All values are atomic.
- 2NF : Since `SongID` is the primary key, all other attributes fully depend on it.
- 3NF : There are no transitive dependencies in the table, as each attribute relates directly to `SongID`.

3. **Artist Table**

Original Attributes : `ArtistID (PK), Name, Country, GenreID`

- 1NF : All values are atomic.
- 2NF : Since `ArtistID` is the primary key, all other attributes fully depend on it.
- 3NF : There are no transitive dependencies in this table.

4. **Album Table**

Original Attributes : `AlbumID (PK), Title, ReleaseDate`

- 1NF : All values are atomic.
- 2NF : Since `AlbumID` is the primary key, all other attributes depend fully on it.
- 3NF : There are no transitive dependencies.

5. Genre Table

Original Attributes : `GenreID (PK), GenreName`

- 1NF : All values are atomic.
- 2NF : Since `GenreID` is the primary key, all other attributes depend fully on it.
- 3NF : There are no transitive dependencies.

6. Playlist Table

Original Attributes : `PlaylistID (PK), UserID (FK), PlaylistName, CreatedDate`

- 1NF : All values are atomic.
- 2NF : Since `PlaylistID` is the primary key, all attributes depend on it directly.
- 3NF : There are no transitive dependencies.

7. User_Likes_Song Table

This table represents a many-to-many relationship between `User` and `Song`.

Attributes : `UserID (PK, FK), SongID (PK, FK)`

- 1NF : All values are atomic.
- 2NF : Both `UserID` and `SongID` together form the composite primary key, and there are no additional non-key attributes.
- 3NF : No transitive dependencies as there are only foreign key references.

8. Playlist_Contains_Song Table

This table represents a many-to-many relationship between `Playlist` and `Song`.

Attributes : `PlaylistID (PK, FK), SongID (PK, FK)`

- 1NF : All values are atomic.
- 2NF : Both `PlaylistID` and `SongID` together form the composite primary key, and there are no additional non-key attributes.
- 3NF : No transitive dependencies as there are only foreign key references.

5. Table commands of the Normalized tables

CREATE TABLE User (

```
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL UNIQUE,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Password VARCHAR(100) NOT NULL,
    SubscriptionType VARCHAR(20) CHECK (SubscriptionType IN ('Free', 'Premium', 'Family'))
);
CREATE TABLE Song (
    SongID INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Duration INT NOT NULL CHECK (Duration > 0),
    ReleaseDate DATE,
    AlbumID INT,
    GenreID INT,
    FOREIGN KEY (AlbumID) REFERENCES Album(AlbumID),
    FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
CREATE TABLE Artist (
    ArtistID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Country VARCHAR(50),
    GenreID INT,
    FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
CREATE TABLE Album (
    AlbumID INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    ReleaseDate DATE
);
CREATE TABLE Genre (
    GenreID INT PRIMARY KEY,
```

```
    GenreName VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE Playlist (
    PlaylistID INT PRIMARY KEY,
    UserID INT NOT NULL,
    PlaylistName VARCHAR(100) NOT NULL,
    CreatedDate DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
CREATE TABLE User_Likes_Song (
    UserID INT,
    SongID INT,
    PRIMARY KEY (UserID, SongID),
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (SongID) REFERENCES Song(SongID)
);
   CREATE TABLE Playlist_Contains_Song (
    PlaylistID INT,
    SongID INT,
    PRIMARY KEY (PlaylistID, SongID),
    FOREIGN KEY (PlaylistID) REFERENCES Playlist(PlaylistID),
    FOREIGN KEY (SongID) REFERENCES Song(SongID)
);
```

6.Queries covering all topics

1. **Using `BETWEEN`**

- Question : Find all songs released between January 1, 2012, and December 31, 2013.

- Query :

```
SELECT Title, ReleaseDate
FROM Song
WHERE ReleaseDate BETWEEN ‘2012-01-01' AND '2013-12-31'';
```

- Output : List of song titles and release dates.

```
+------------+-------------+
| Title      | ReleaseDate |
+------------+-------------+
| Tum Hi Ho  | 2013-04-08  |
| Wake Me Up | 2013-06-17  |
+------------+-------------+
```

2. **Using `LIKE`**

- Question : Retrieve all artist names that start with the letter "A".
- Query :

```
SELECT Name
FROM Artist
WHERE Name LIKE 'A%';
```

- Output : List of artist names starting with "A".

```
+--------------+
| Name         |
+--------------+
| A. R. Rahman |
| Arijit Singh |
| Adele        |
| Asha Bhosle  |
+--------------+
```

3. **Queries on Dates**

- Question : Find all playlists created in the year 2024.
- Query :

```
SELECT PlaylistName, CreatedDate
FROM Playlist
WHERE YEAR
```

- Output : List of playlist names and creation dates for 2024.

PlaylistName	CreatedDate
Morning Motivation	2024-01-01
Bollywood Hits	2024-01-02
Workout Jams	2024-01-03
Classical Relaxation	2024-01-04
Rock Anthems	2024-01-05
Party Time	2024-01-06
Top Pop Songs	2024-01-07
Jazz Vibes	2024-01-08
Sufi Soothers	2024-01-09
Romantic Ballads	2024-01-10
Hip-Hop Beats	2024-01-11
Soulful Sunday	2024-01-12
EDM Hits	2024-01-13
Travel Tunes	2024-01-14
Country Classics	2024-01-15
Retro Bollywood	2024-01-16
Lofi Chill	2024-01-17
Focus and Study	2024-01-18
Reggae Roots	2024-01-19
Evening Melodies	2024-01-20
Metal Madness	2024-01-21

Latin Fiesta	2024-01-22
Opera Essentials	2024-01-23
Dance Party	2024-01-24
Indie Mix	2024-01-25
Vibes	2024-11-12
Old Songs	2024-11-12
+----------+----------+

4. **Using `ORDER BY`**

- Question : List all songs in ascending order by their release date.
- Query :

```
SELECT Title, ReleaseDate
FROM Song
ORDER BY ReleaseDate ASC;
```

- Output : List of song titles and release dates in ascending order.

+----------+----------+
| Title | ReleaseDate |
+----------+----------+
Symphony No. 5	1808-12-22
What a Wonderful World	1967-09-01
Aaj Jaane Ki Zid Na Karo	1970-06-15
Take Me Home, Country Roads	1971-04-12
Imagine	1971-10-11
Chura Liya Hai Tumne	1973-02-01
Bohemian Rhapsody	1975-10-31
Hotel California	1977-02-22
Stayin' Alive	1977-11-13
Highway to Hell	1979-07-27
Thriller	1982-11-30
Billie Jean	1983-01-02

```
| Chaiyya Chaiyya              | 1998-08-21  |
| Smooth                 | 1999-06-29  |
| Taal Se Taal Mila            | 1999-10-15  |
| Lose Yourself                | 2002-10-28  |
| Sufi Qawwali                 | 2003-08-08  |
| Ye Jo Des Hai Tera           | 2004-12-24  |
| Suno Aisha             | 2010-04-30  |
| Rolling in the Deep          | 2010-11-29  |
| Zindagi Na Milegi Dobara  | 2011-06-15  |
| Tum Hi Ho              | 2013-04-08  |
| Wake Me Up                   | 2013-06-17  |
| Shape of You                 | 2017-01-06  |
| Despacito              | 2017-01-13  |
+----------------------------+-------------+
```

5. **Using `GROUP BY`**

- Question : Count the number of songs in each genre.
- Query :

```
SELECT GenreID, COUNT( ) AS SongCount
FROM Song
GROUP BY GenreID;
```

- Output : List of genre IDs with the number of songs in each.

```
+---------+-----------+
| GenreID | SongCount |
+---------+-----------+
|       1 |         3 |
|       2 |         2 |
|       3 |         1 |
|       4 |         1 |
|       5 |         1 |
```

```
|        6 |      1 |
|        7 |      2 |
|        10 |     1 |
|        13 |     2 |
|        14 |     1 |
|        15 |     1 |
|        24 |     7 |
|        25 |     2 |
+---------+-----------+
```

6. **Join with Cartesian Product**

- Question : Retrieve all combinations of user names and playlist names.
- Query :

Select UserName, PlaylistName from User natural join Playlist natural join Playlist_Contains_Song;

- Output : List of all combinations of usernames and playlists.

```
+--------------+---------------------+
| UserName    | PlaylistName          |
+--------------+---------------------+
| Aarav Sharma | Morning Motivation   |
| Manasi Jog   | Old Songs            |
| Ishaan Verma | Bollywood Hits       |
| Vihaan Patel | Workout Jams         |
| Aditya Mehta | Classical Relaxation |
| Ishaan Verma | Bollywood Hits       |
| Aditya Mehta | Classical Relaxation |
| Vihaan Patel | Workout Jams         |
| Sai Kumar    | Rock Anthems         |
| Sai Kumar    | Rock Anthems         |
| Aditya Mehta | Classical Relaxation |
```

```
| Aarav Sharma | Morning Motivation   |
| Vihaan Patel | Workout Jams         |
| Ishaan Verma | Bollywood Hits       |
| Ishaan Verma | Bollywood Hits       |
| Aarav Sharma | Morning Motivation   |
| Aarav Sharma | Morning Motivation   |
| Sai Kumar    | Rock Anthems         |
| Sai Kumar    | Rock Anthems         |
| Aditya Mehta | Classical Relaxation |
| Aditya Mehta | Classical Relaxation |
| Vihaan Patel | Workout Jams         |
| Vihaan Patel | Workout Jams         |
| Aarav Sharma | Morning Motivation   |
| Vihaan Patel | Workout Jams         |
+--------------+----------------------+
```

7. **Inner Join**

- Question : List songs along with their album titles.
- Query :

```
SELECT S.Title AS SongTitle, A.Title AS AlbumTitle
FROM Song S
INNER JOIN Album A ON S.AlbumID = A.AlbumID;
```

- Output : List of song titles and their corresponding album titles.

```
+----------------------------+--------------------+
| SongTitle                  | AlbumTitle         |
+----------------------------+--------------------+
| Tum Hi Ho                  | Bollywood Bash     |
| Chaiyya Chaiyya            | Bollywood Classics |
| Shape of You               | EDM Hits           |
| Bohemian Rhapsody          | Rock Legends       |
```

```
| What a Wonderful World   | Jazz Vibes              |
| Taal Se Taal Mila        | Bollywood Bash          |
| Symphony No. 5           | Classical Masters       |
| Wake Me Up               | EDM Hits                |
| Lose Yourself            | Hip-Hop Essentials      |
| Smooth                   | Blues Masters           |
| Imagine                  | Rock Legends            |
| Despacito                | Latin Fiesta            |
| Ye Jo Des Hai Tera       | Bollywood Classics      |
| Thriller                 | Disco Fever             |
| Highway to Hell          | Rock Legends            |
| Chura Liya Hai Tumne     | Bollywood Bash          |
| Take Me Home, Country Roads | Country Gold            |
| Hotel California         | Rock Legends            |
| Stayin' Alive            | Disco Fever             |
| Rolling in the Deep      | Soulful Sounds          |
| Sufi Qawwali             | Sufi Soul               |
| Billie Jean              | Disco Fever             |
| Suno Aisha               | Bollywood Bash          |
| Aaj Jaane Ki Zid Na Karo | Bollywood Bash          |
| Zindagi Na Milegi Dobara | Bollywood Bash          |
+--------------------------+-------------------+
```

8. Left Outer Join

- Question : Retrieve all albums and any associated songs (if available).
- Query :

```
SELECT A.Title AS AlbumTitle, S.Title AS SongTitle
FROM Album A
LEFT JOIN Song S ON A.AlbumID = S.AlbumID;
```

- Output : List of album titles with associated song titles (or NULL if no song).

AlbumTitle	SongTitle
Bollywood Classics	Chaiyya Chaiyya
Bollywood Classics	Ye Jo Des Hai Tera
Rock Legends	Bohemian Rhapsody
Rock Legends	Imagine
Rock Legends	Highway to Hell
Rock Legends	Hotel California
Jazz Vibes	What a Wonderful World
Classical Masters	Symphony No. 5
EDM Hits	Shape of You
EDM Hits	Wake Me Up
Hip-Hop Essentials	Lose Yourself
Soulful Sounds	Rolling in the Deep
Folklore	NULL
Country Gold	Take Me Home, Country Roads
Reggae Roots	NULL
Blues Masters	Smooth
Latin Fiesta	Despacito
Disco Fever	Thriller
Disco Fever	Stayin' Alive
Disco Fever	Billie Jean
Punk Rock Anthems	NULL
Metal Mania	NULL
Gospel Greats	NULL
Opera Night	NULL
Techno Beats	NULL
Trance Journey	NULL

```
| House Party          | NULL                    |
| Dance Floor Hits     | NULL                    |
| K-Pop Superstars     | NULL                    |
| Bollywood Bash       | Tum Hi Ho               |
| Bollywood Bash       | Taal Se Taal Mila       |
| Bollywood Bash       | Chura Liya Hai Tumne    |
| Bollywood Bash       | Suno Aisha              |
| Bollywood Bash       | Aaj Jaane Ki Zid Na Karo |
| Bollywood Bash       | Zindagi Na Milegi Dobara |
| Sufi Soul            | Sufi Qawwali            |
| Indie Vibes          | NULL                    |
```

9. Right Outer Join

- Question : Retrieve all songs and any associated albums.
- Query :

```
SELECT S.Title AS SongTitle, A.Title AS AlbumTitle
FROM Song S
RIGHT JOIN Album A ON S.AlbumID = A.AlbumID;
```

- Output : List of song titles with corresponding album titles (or NULL if no album).

```
+---------------------------+-------------------+
| SongTitle                 | AlbumTitle        |
+---------------------------+-------------------+
| Chaiyya Chaiyya           | Bollywood Classics |
| Ye Jo Des Hai Tera        | Bollywood Classics |
| Bohemian Rhapsody         | Rock Legends       |
| Imagine                   | Rock Legends       |
| Highway to Hell           | Rock Legends       |
| Hotel California          | Rock Legends       |
```

What a Wonderful World	Jazz Vibes
Symphony No. 5	Classical Masters
Shape of You	EDM Hits
Wake Me Up	EDM Hits
Lose Yourself	Hip-Hop Essentials
Rolling in the Deep	Soulful Sounds
NULL	Folklore
Take Me Home, Country Roads	Country Gold
NULL	Reggae Roots
Smooth	Blues Masters
Despacito	Latin Fiesta
Thriller	Disco Fever
Stayin' Alive	Disco Fever
Billie Jean	Disco Fever
NULL	Punk Rock Anthems
NULL	Metal Mania
NULL	Gospel Greats
NULL	Opera Night
NULL	Techno Beats
NULL	Trance Journey
NULL	House Party
NULL	Dance Floor Hits
NULL	K-Pop Superstars
Tum Hi Ho	Bollywood Bash
Taal Se Taal Mila	Bollywood Bash
Chura Liya Hai Tumne	Bollywood Bash
Suno Aisha	Bollywood Bash
Aaj Jaane Ki Zid Na Karo	Bollywood Bash
Zindagi Na Milegi Dobara	Bollywood Bash
Sufi Qawwali	Sufi Soul

| NULL | Indie Vibes |

10. Subquery

- Question : Find the longest song in the database.
- Query :

```
SELECT Title, Duration
FROM Song
WHERE Duration = (SELECT MAX(Duration) FROM Song);
```

- Output : The title and duration of the longest song.

```
+--------------+----------+
| Title        | Duration |
+--------------+----------+
| Sufi Qawwali | 00:07:30 |
+--------------+----------+
```

11. View

- Question : Create a view to list each user's playlist count.
- Query :

```
CREATE VIEW UserPlaylistCount AS
SELECT UserID, COUNT(PlaylistID) AS PlaylistCount
FROM Playlist
GROUP BY UserID;
```

- Output : A view showing UserID with their respective playlist count.

```
SELECT  FROM UserPlaylistCount;
```

```
+--------+---------------+
| UserID | PlaylistCount |
+--------+---------------+
|      1 |             1 |
|      2 |             1 |
```

3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1
25	1
26	2

12. Index

- Question : Add an index on the `Title` column in the `Song` table to improve search performance.
- Query :

```
CREATE INDEX idx_song_title ON Song (Title);
```

- Output : Index created on the `Title` column of `Song` table.

13. Trigger

- Question : Create a trigger to automatically update a duration of album whenever a song is added.
- Query :

```
CREATE TRIGGER update_album_duration
AFTER INSERT ON Song
FOR EACH ROW
BEGIN
        UPDATE Album
        SET TotalDuration = TotalDuration + NEW.Duration
        WHERE AlbumID = NEW.AlbumID;
END //
```

- Output : Trigger that updates duration

14. Function

- Question : Write a function to get playlist duration..
- Query :

```
CREATE FUNCTION (playlist_id INT)
RETURNS INT DETERMINISTIC
BEGIN
        DECLARE total_duration INT;

        SELECT SUM(Song.Duration) INTO total_duration
        FROM Playlist_Contains_Song
        INNER JOIN Song ON Playlist_Contains_Song.SongID = Song.SongID
        WHERE Playlist_Contains_Song.PlaylistID = playlist_id;
```

```
        RETURN IFNULL(total_duration, 0);
END //
```

- Output : Function that gives duration

```
SELECT PlaylistName, get_playlist_duration(PlaylistID) FROM Playlist WHERE PlaylistID = 28;
+--------------+----------------------------------+
| PlaylistName | get_playlist_duration(PlaylistID) |
+--------------+----------------------------------+
| Old Songs    |                              422 |
+--------------+----------------------------------+
```

15. Using `COUNT` with `GROUP BY`

- Question : Find the total number of songs liked by each user.
- Query :

```
SELECT UserID, COUNT(SongID) AS TotalLikes
FROM User_Likes_Song
GROUP BY UserID;
```

- Output : User IDs and the total number of songs they liked.

```
+--------+------------+
| UserID | TotalLikes |
+--------+------------+
|      1 |          1 |
|      2 |          1 |
|      3 |          1 |
|      4 |          1 |
|      5 |          1 |
|      6 |          1 |
|      7 |          1 |
|      8 |          1 |
|      9 |          1 |
```

```
|        10 |            1 |
|        11 |            1 |
|        12 |            1 |
|        13 |            1 |
|        14 |            1 |
|        15 |            1 |
|        16 |            1 |
|        17 |            1 |
|        18 |            1 |
|        19 |            1 |
|        20 |            1 |
|        21 |            1 |
|        22 |            1 |
|        23 |            1 |
|        24 |            1 |
|        25 |            1 |
|        26 |            2 |
+--------+------------+
```

16. `HAVING` with Aggregate Function

- Question : Retrieve users who have created more than two playlists.
- Query :

```
SELECT UserID, COUNT(PlaylistID) AS PlaylistCount
FROM Playlist
GROUP BY UserID
HAVING COUNT(PlaylistID) > 2;
```

- Output : List of UserIDs with more than two playlists.

```
+--------+---------------+
| UserID | PlaylistCount |
+--------+---------------+
```

```
|        26 |              2 |
+--------+---------------+
```

17. Join with `LIKE`

- Question : Find all users who created playlists with names containing "Chill".
- Query :

```
SELECT U.Username, P.PlaylistName
FROM User U
INNER JOIN Playlist P ON U.UserID = P.UserID
WHERE P.PlaylistName LIKE '%Chill%';
```

- Output : List of users with playlists containing "Chill".

```
+---------------+--------------+
| Username    | PlaylistName |
+---------------+--------------+
| Pranav Thakur | Lofi Chill   |
+---------------+--------------+
```

18. Subquery in `WHERE` Clause

- Question : Find all songs that are in playlists created by users with whose name starts with A.
- Query :

```
SELECT DISTINCT Song.Title FROM Song NATURAL JOIN Playlist
WHERE UserID IN (SELECT UserID FROM User Where Username LIKE "A%
");
```

- Output : List of song titles in playlists created by users with whose name starts with A.

```
+----------------------------+
| Title                      |
```

```
+----------------------------+
| Aaj Jaane Ki Zid Na Karo    |
| Billie Jean                 |
| Bohemian Rhapsody           |
| Chaiyya Chaiyya             |
| Chura Liya Hai Tumne        |
| Despacito                   |
| Highway to Hell             |
| Hotel California            |
| Imagine                     |
| Lose Yourself               |
| Rolling in the Deep         |
| Shape of You                |
| Smooth                      |
| Stayin' Alive               |
| Sufi Qawwali                |
| Suno Aisha                  |
| Symphony No. 5              |
| Taal Se Taal Mila           |
| Take Me Home, Country Roads |
| Thriller                    |
| Tum Hi Ho                   |
| Wake Me Up                  |
| What a Wonderful World      |
| Ye Jo Des Hai Tera          |
| Zindagi Na Milegi Dobara    |
+----------------------------+
```

19. Using `DATE` Functions

- Question : Find all playlists created in the past 30 days.

- Query :

```
SELECT PlaylistName, CreatedDate
FROM Playlist
WHERE CreatedDate >= CURDATE() - INTERVAL 30 DAY;
```

- Output : List of recent playlists created in the last 30 days.

```
+--------------+-------------+
| PlaylistName | CreatedDate |
+--------------+-------------+
| Vibes        | 2024-11-12  |
| Old Songs    | 2024-11-12  |
+--------------+-------------+
```

20. Using `JOIN` with `ORDER BY`

- Question : List all songs along with album names, ordered by album names alphabetically

- Query :

```
SELECT Song.Title, Album.Title from Song JOIN Album where
Song.AlbumID = Album.AlbumID ORDER BY Album.Title;
```

- Output : List of song titles and album names sorted alphabetically by artist name.

```
+----------------------------+--------------------+
| Title                      | Title              |
+----------------------------+--------------------+
| Smooth                     | Blues Masters      |
| Zindagi Na Milegi Dobara   | Bollywood Bash     |
| Aaj Jaane Ki Zid Na Karo   | Bollywood Bash     |
| Suno Aisha                 | Bollywood Bash     |
| Chura Liya Hai Tumne       | Bollywood Bash     |
| Taal Se Taal Mila          | Bollywood Bash     |
| Tum Hi Ho                  | Bollywood Bash     |
```

```
| Ye Jo Des Hai Tera          | Bollywood Classics |
| Chaiyya Chaiyya             | Bollywood Classics |
| Symphony No. 5              | Classical Masters  |
| Take Me Home, Country Roads | Country Gold       |
| Thriller                    | Disco Fever        |
| Stayin' Alive               | Disco Fever        |
| Billie Jean                 | Disco Fever        |
| Shape of You                | EDM Hits           |
| Wake Me Up                  | EDM Hits           |
| Lose Yourself               | Hip-Hop Essentials |
| What a Wonderful World      | Jazz Vibes         |
| Despacito                   | Latin Fiesta       |
| Imagine                     | Rock Legends       |
| Highway to Hell             | Rock Legends       |
| Hotel California            | Rock Legends       |
| Bohemian Rhapsody           | Rock Legends       |
| Rolling in the Deep         | Soulful Sounds     |
| Sufi Qawwali                | Sufi Soul          |
+----------------------------+--------------------+
```

21. Stored Procedure: Add New Playlist

- Question : Write a stored procedure to add a new playlist for a user.
- Query :

```
sql
DELIMITER //
CREATE PROCEDURE AddNewPlaylist(
  IN userId INT,
  IN playlistName VARCHAR(100)
)
BEGIN
```

```
    INSERT INTO Playlist (UserID, PlaylistName, CreatedDate)
    VALUES (userId, playlistName, CURDATE());
  END //
  DELIMITER ;
```

8.sample code of project with input/output screenshot

Welcome to MELODIFY !!!!!!!!

1: Create an Account

2: Login

Choose an option: 1

Enter username: Prerana

Enter email: prerana@gmail.com

Set password: pd123

Account created successfully!

Enter email: prerana@gmail.com

Enter password: pd123

You have logged in successfully!

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 1

Enter name of playlist: My Songs

Playlist created successfully!

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 2

Enter playlist name: My Songs

Enter title of the song you want to add: Tum Hi Ho

Song added to the playlist successfully!

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 3

Enter title of the song you want to like: Tum Hi Ho

Song liked!

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 4

My Songs

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 5

Enter playlist name: My Songs

Tum Hi Ho

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 6

Enter title of song: Wake Me Up

Song found!

Song title: Wake Me Up

Song duration: 00:04:07

Release date: 2013-06-17

Menu:

1: Create Playlist

2: Add songs to playlist

3: Like a song

4: Show playlists

5: Show songs from a playlist

6: Search a song

7: Exit

Choose an option: 7