Creating an Efficient Database Structure for Spotify-like System

Author: Mobin Kheibary [994421017]

Supervisor: Dr. Ehsan Shoja

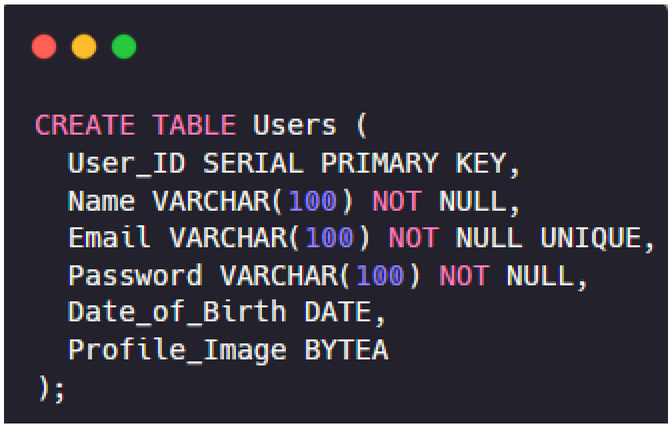
Introduction

The purpose of this report is to present the design of a highly efficient and scalable database for a Spotify-like music streaming system using PostgreSQL. The database design encompasses various tables to manage users, artists, albums, tracks, playlists, followers, likes, and incorporates additional features such as premium user functionality, payment system, recommendation engine, and notification system. The report covers the logical design, table structures, relationships, data normalization, indexing strategy, and provides sample SQL codes for table creation, data insertion, and retrieval queries.

Table Design and Structure

Users Table

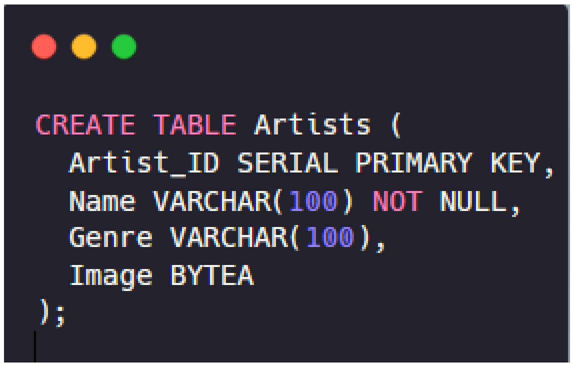
The Users table stores user information such as name, email, password, date of birth, and profile image. The table structure in SQL is as follows:



```sql
CREATE TABLE Users (
  User_ID SERIAL PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  Email VARCHAR(100) NOT NULL UNIQUE,
  Password VARCHAR(100) NOT NULL,
  Date_of_Birth DATE,
  Profile_Image BYTEA
);
```

Artists Table

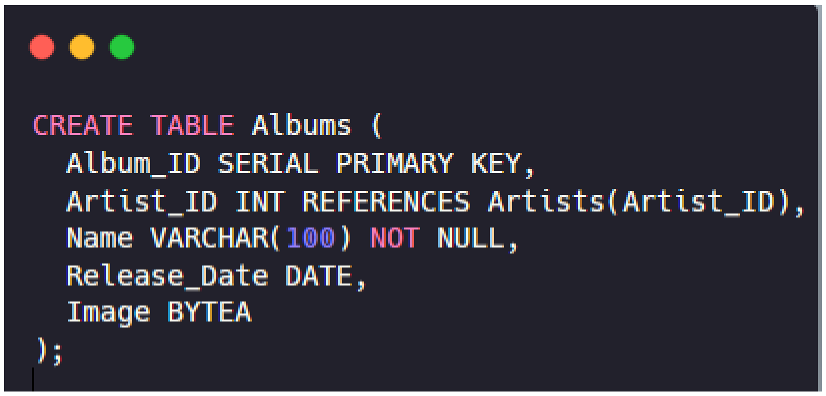
The Artists table holds artist details including name, genre, and image. The table structure in SQL is as follows:



```sql
CREATE TABLE Artists (
  Artist_ID SERIAL PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  Genre VARCHAR(100),
  Image BYTEA
);
```

Albums Table

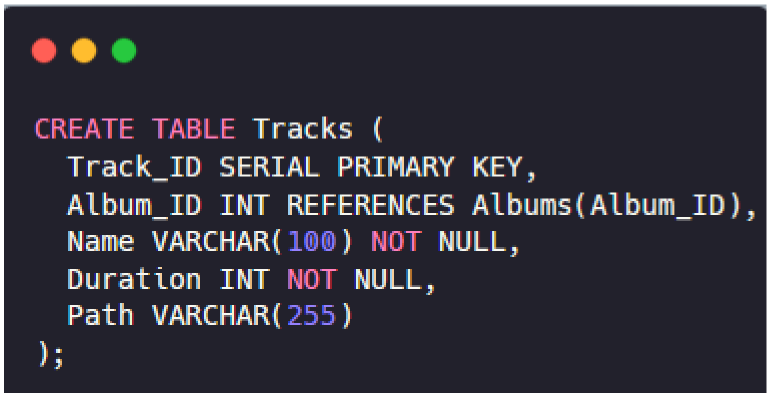
The Albums table stores album information such as name, release date, image, and establishes a foreign key relationship with the Artists table. The table structure in SQL is as follows:



```sql
CREATE TABLE Albums (
  Album_ID SERIAL PRIMARY KEY,
  Artist_ID INT REFERENCES Artists(Artist_ID),
  Name VARCHAR(100) NOT NULL,
  Release_Date DATE,
  Image BYTEA
);
```

Tracks Table

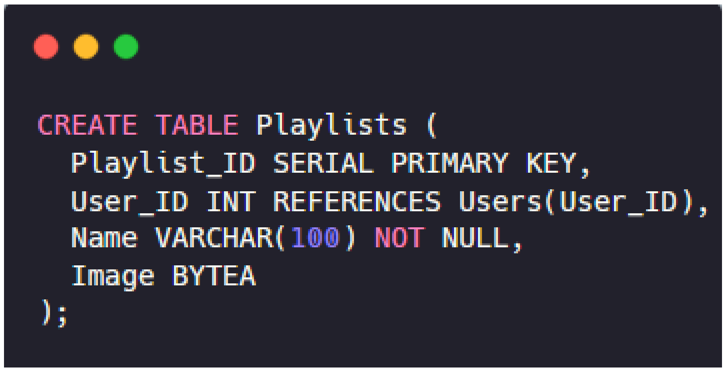
The Tracks table stores track details like name, duration, file path, and establishes a foreign key relationship with the Albums table. The table structure in SQL is as follows:



```sql
CREATE TABLE Tracks (
  Track_ID SERIAL PRIMARY KEY,
  Album_ID INT REFERENCES Albums(Album_ID),
  Name VARCHAR(100) NOT NULL,
  Duration INT NOT NULL,
  Path VARCHAR(255)
);
```

Playlists Table

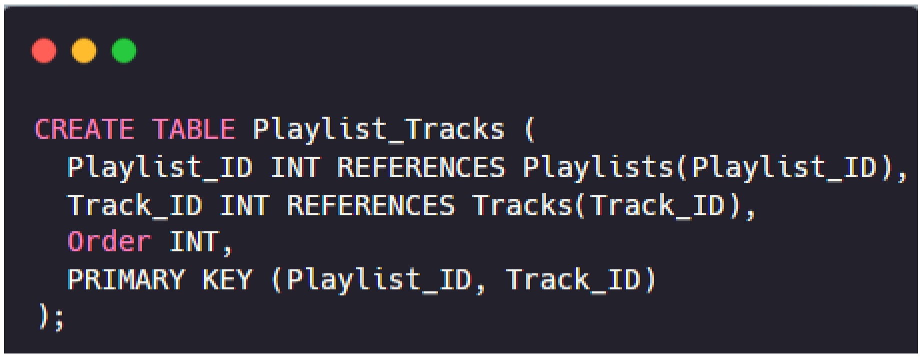
The Playlists table contains information about user-created playlists, including name and image, and establishes a foreign key relationship with the Users table. The table structure in SQL is as follows:



```
CREATE TABLE Playlists (
  Playlist_ID SERIAL PRIMARY KEY,
  User_ID INT REFERENCES Users(User_ID),
  Name VARCHAR(100) NOT NULL,
  Image BYTEA
);
```

Playlist_Tracks Table

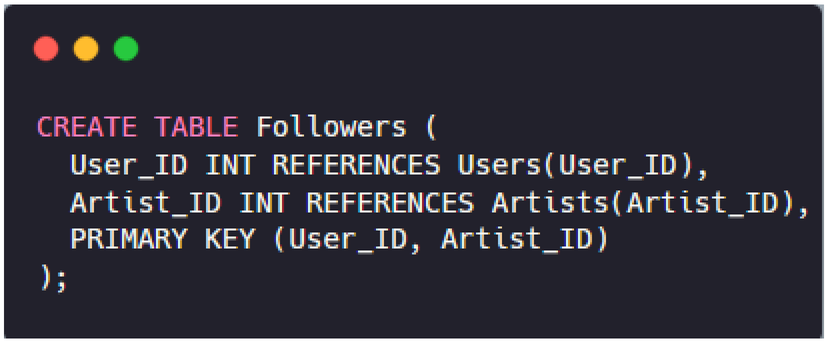
The Playlist_Tracks table serves as an association between playlists and tracks. It includes foreign keys linking to the Playlists and Tracks tables, as well as an Order column to maintain track order within a playlist. The table structure in SQL is as follows:



```
CREATE TABLE Playlist_Tracks (
  Playlist_ID INT REFERENCES Playlists(Playlist_ID),
  Track_ID INT REFERENCES Tracks(Track_ID),
  Order INT,
  PRIMARY KEY (Playlist_ID, Track_ID)
);
```

Followers Table

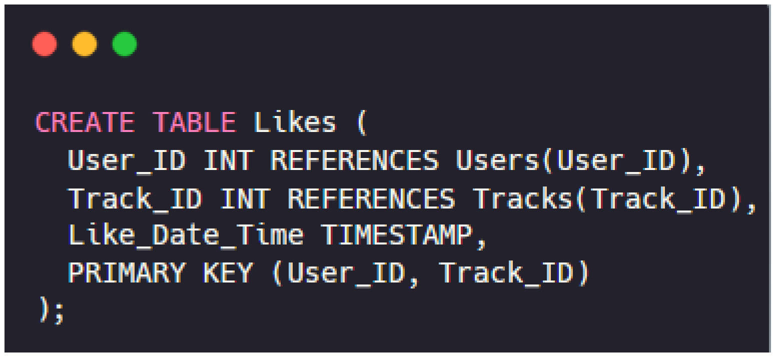
The Followers table establishes the relationship between users and artists, allowing users to follow multiple artists. It includes foreign keys linking to the Users and Artists tables. The table structure in SQL is as follows:



```sql
CREATE TABLE Followers (
  User_ID INT REFERENCES Users(User_ID),
  Artist_ID INT REFERENCES Artists(Artist_ID),
  PRIMARY KEY (User_ID, Artist_ID)
);
```

Likes Table

The Likes table stores information about tracks liked by users, including the date and time of the like. It includes foreign keys linking to the Users and Tracks tables. The table structure in SQL is as follows:

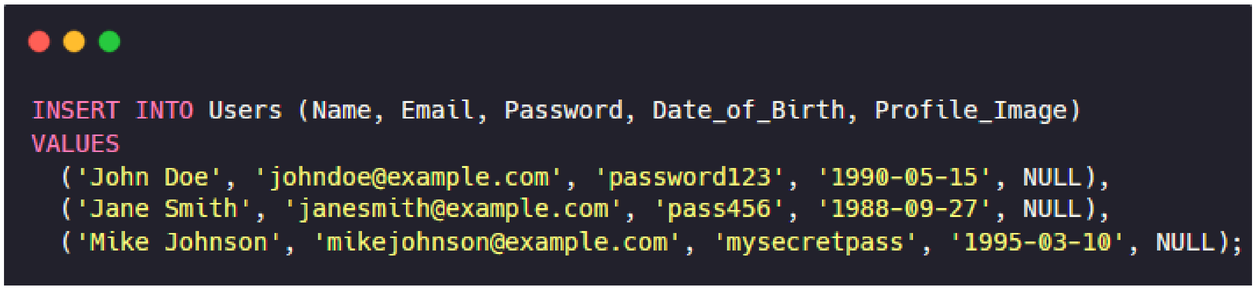


```sql
CREATE TABLE Likes (
  User_ID INT REFERENCES Users(User_ID),
  Track_ID INT REFERENCES Tracks(Track_ID),
  Like_Date_Time TIMESTAMP,
  PRIMARY KEY (User_ID, Track_ID)
);
```

Data Insertion

Inserting Sample Data into Users Table

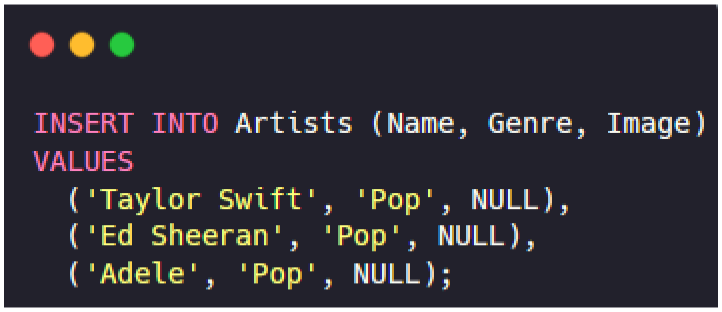
To insert sample data into the Users table, the following SQL query can be used:



```sql
INSERT INTO Users (Name, Email, Password, Date_of_Birth, Profile_Image)
VALUES
  ('John Doe', 'johndoe@example.com', 'password123', '1990-05-15', NULL),
  ('Jane Smith', 'janesmith@example.com', 'pass456', '1988-09-27', NULL),
  ('Mike Johnson', 'mikejohnson@example.com', 'mysecretpass', '1995-03-10', NULL);
```

Inserting Sample Data into Artists Table

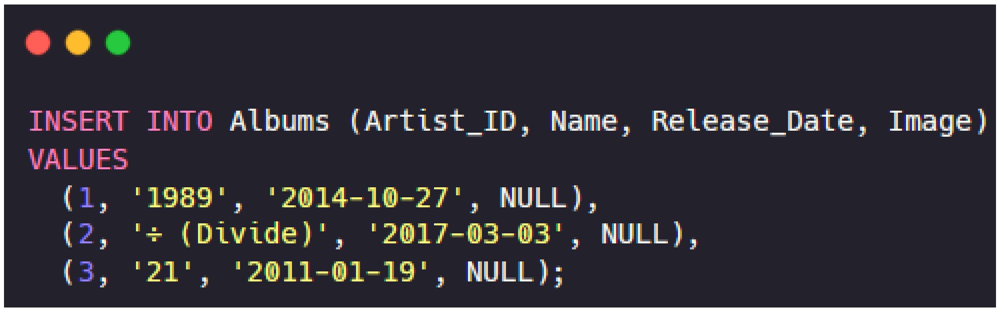
To insert sample data into the Artists table, the following SQL query can be used:



```sql
INSERT INTO Artists (Name, Genre, Image)
VALUES
  ('Taylor Swift', 'Pop', NULL),
  ('Ed Sheeran', 'Pop', NULL),
  ('Adele', 'Pop', NULL);
```

Inserting Sample Data into Albums Table

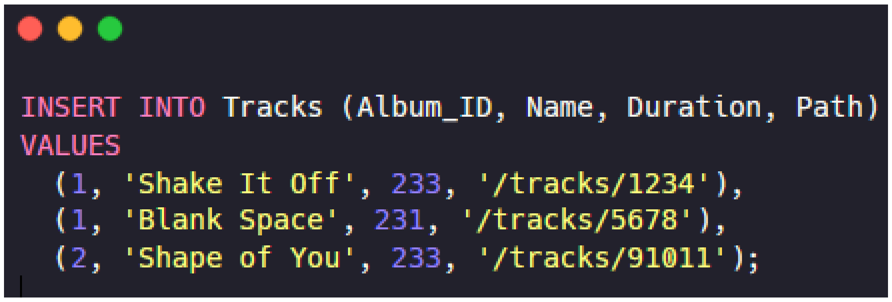
To insert sample data into the Albums table, the following SQL query can be used:



```sql
INSERT INTO Albums (Artist_ID, Name, Release_Date, Image)
VALUES
  (1, '1989', '2014-10-27', NULL),
  (2, '÷ (Divide)', '2017-03-03', NULL),
  (3, '21', '2011-01-19', NULL);
```

Inserting Sample Data into Tracks Table

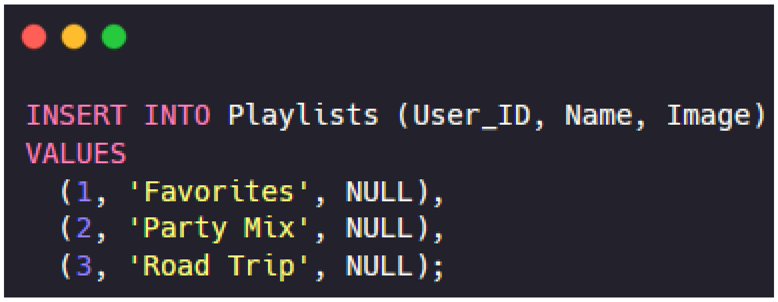
To insert sample data into the Tracks table, the following SQL query can be used:



```sql
INSERT INTO Tracks (Album_ID, Name, Duration, Path)
VALUES
  (1, 'Shake It Off', 233, '/tracks/1234'),
  (1, 'Blank Space', 231, '/tracks/5678'),
  (2, 'Shape of You', 233, '/tracks/91011');
```

Inserting Sample Data into Playlists Table

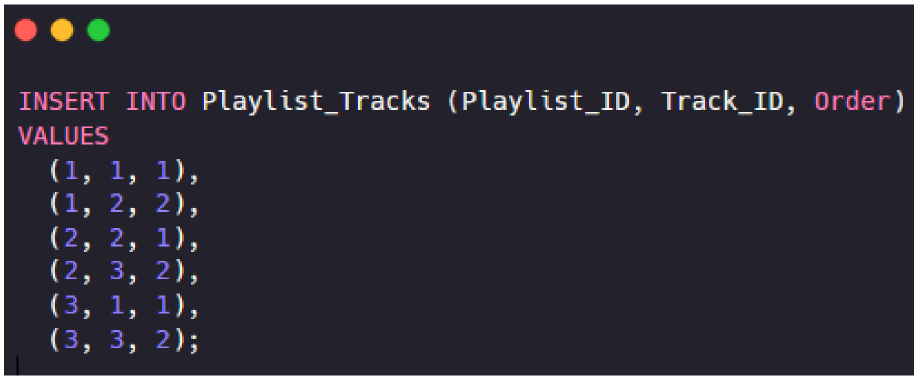
To insert sample data into the Playlists table, the following SQL query can be used:



```sql
INSERT INTO Playlists (User_ID, Name, Image)
VALUES
  (1, 'Favorites', NULL),
  (2, 'Party Mix', NULL),
  (3, 'Road Trip', NULL);
```

Inserting Sample Data into Playlist_Tracks Table

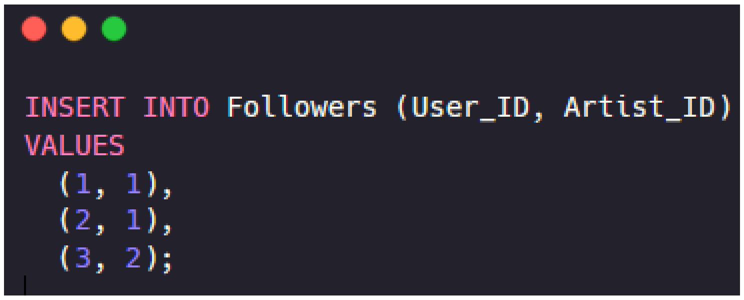
To insert sample data into the Playlist_Tracks table, the following SQL query can be used:



```sql
INSERT INTO Playlist_Tracks (Playlist_ID, Track_ID, Order)
VALUES
  (1, 1, 1),
  (1, 2, 2),
  (2, 2, 1),
  (2, 3, 2),
  (3, 1, 1),
  (3, 3, 2);
```

Inserting Sample Data into Followers Table

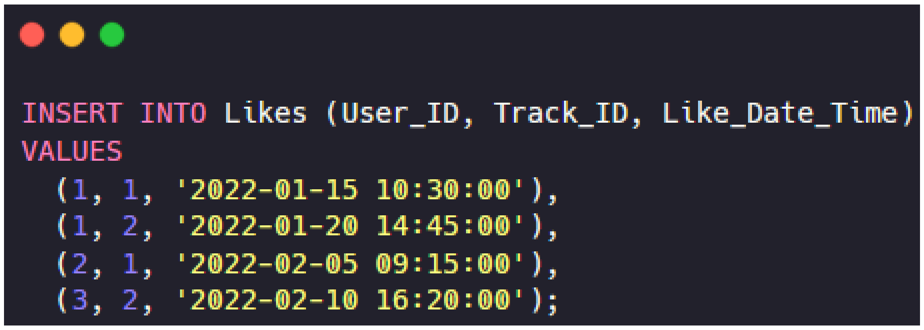
To insert sample data into the Followers table, the following SQL query can be used:



```sql
INSERT INTO Followers (User_ID, Artist_ID)
VALUES
  (1, 1),
  (2, 1),
  (3, 2);
```

Inserting Sample Data into Likes Table

To insert sample data into the Likes table, the following SQL query can be used:

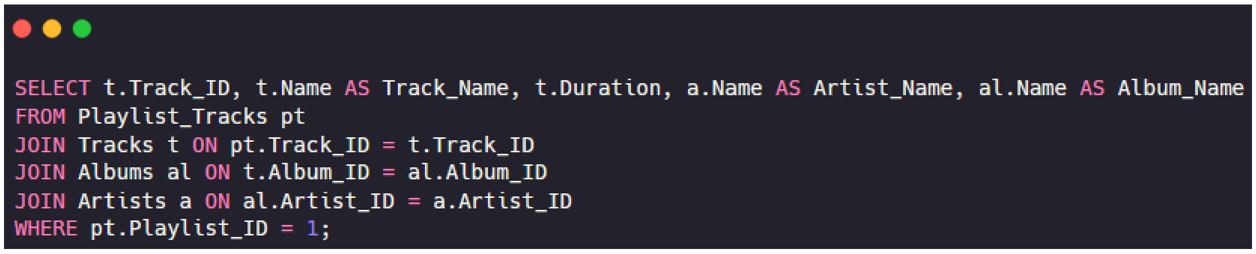


```sql
INSERT INTO Likes (User_ID, Track_ID, Like_Date_Time)
VALUES
  (1, 1, '2022-01-15 10:30:00'),
  (1, 2, '2022-01-20 14:45:00'),
  (2, 1, '2022-02-05 09:15:00'),
  (3, 2, '2022-02-10 16:20:00');
```

Data Retrieval Queries

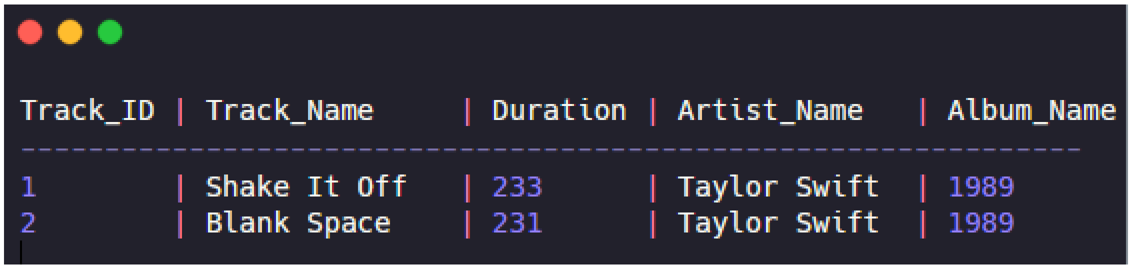
Retrieve Tracks in a Playlist

To retrieve the tracks in a specific playlist along with their details, the following SQL query can be used:



```sql
SELECT t.Track_ID, t.Name AS Track_Name, t.Duration, a.Name AS Artist_Name, al.Name AS Album_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 a ON al.Artist_ID = a.Artist_ID
WHERE pt.Playlist_ID = 1;
```

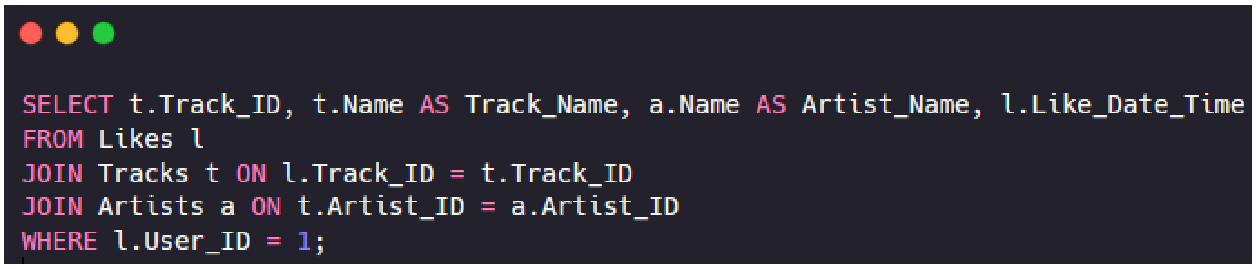
Expected Result:



```
Track_ID | Track_Name      | Duration | Artist_Name    | Album_Name
-----------------------------------------------------------------
1        | Shake It Off    | 233      | Taylor Swift   | 1989
2        | Blank Space     | 231      | Taylor Swift   | 1989
```

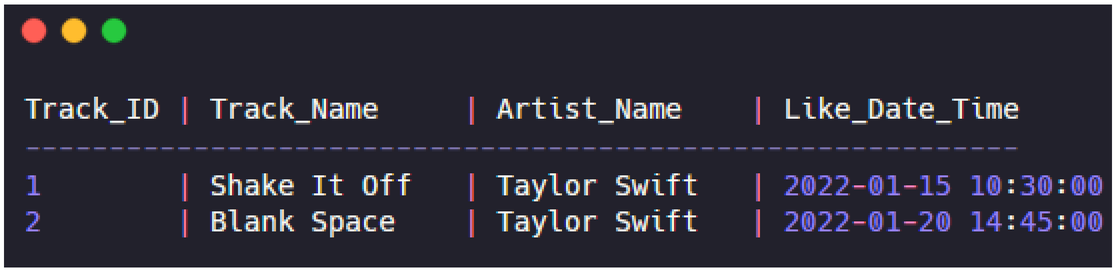
Retrieve Liked Tracks by a User

To retrieve the tracks liked by a specific user, the following SQL query can be used:



```sql
SELECT t.Track_ID, t.Name AS Track_Name, a.Name AS Artist_Name, l.Like_Date_Time
FROM Likes l
JOIN Tracks t ON l.Track_ID = t.Track_ID
JOIN Artists a ON t.Artist_ID = a.Artist_ID
WHERE l.User_ID = 1;
```

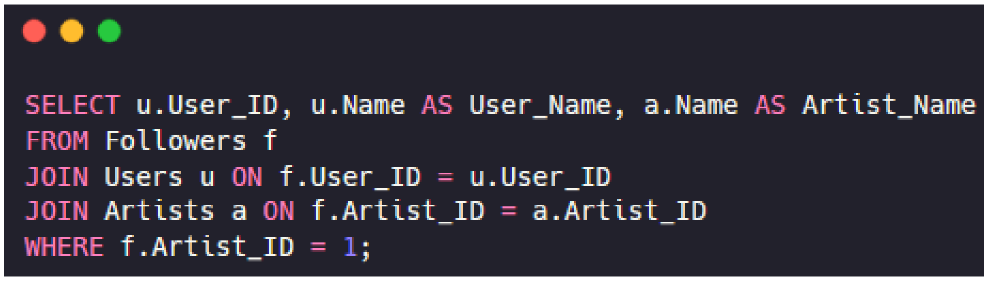
Expected Result:



```
Track_ID | Track_Name     | Artist_Name    | Like_Date_Time
-----------------------------------------------------------------
1        | Shake It Off   | Taylor Swift   | 2022-01-15 10:30:00
2        | Blank Space    | Taylor Swift   | 2022-01-20 14:45:00
```

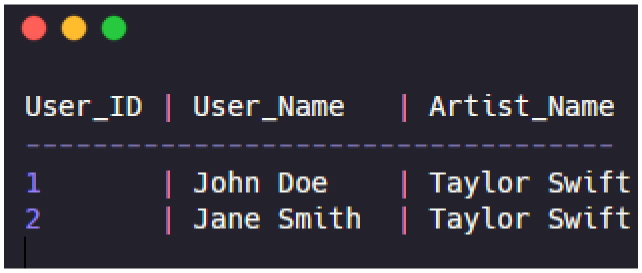
Retrieve Users Following an Artist

To retrieve the users who are following a specific artist, the following SQL query can be used:



```sql
SELECT u.User_ID, u.Name AS User_Name, a.Name AS Artist_Name
FROM Followers f
JOIN Users u ON f.User_ID = u.User_ID
JOIN Artists a ON f.Artist_ID = a.Artist_ID
WHERE f.Artist_ID = 1;
```

Expected Result:



```
User_ID | User_Name   | Artist_Name
--------------------------------------
1       | John Doe    | Taylor Swift
2       | Jane Smith  | Taylor Swift
```

These sample SQL queries demonstrate basic data retrieval operations from the designed database. Further complex queries can be developed based on specific requirements and additional functionalities of the Spotify-like system.

Conclusion

In conclusion, this report has presented the design of a comprehensive database for a Spotify-like music streaming system using PostgreSQL. The logical design includes tables for users, artists, albums, tracks, playlists, followers, and likes. Additional features such as premium user functionality, payment system, recommendation engine, and notification system have been incorporated. Sample SQL codes were provided for table creation, data insertion, and retrieval queries, offering a glimpse into the implementation and utilization of the database. This database design serves as a solid foundation for building a robust and scalable music streaming platform akin to Spotify.

The End.

Special Thanks to Dr. Shoja for all his efforts.