Designing an Optimized Database for Spotify using PostgreSQL

Author: Mobin Kheibary [994421017]

Supervisor: Dr. Ehsan Shoja

Introduction:

The goal of this project is to design a highly efficient and scalable database for Spotify, leveraging the capabilities of PostgreSQL. This database will serve as the foundation for managing users, artists, albums, tracks, playlists, followers, likes, and more. By following best practices in database design and utilizing PostgreSQL's advanced features, we aim to create a robust and performant system to enhance the Spotify music streaming platform.

1. Database Design:

To begin the database design process, we will identify and define the essential tables required for the Spotify-like system. These tables will include:

- **Users table:** This table will store user information such as name, email, password, date of birth, and profile image.

- **Artists table:** This table will hold artist details including their name, genre, and image.

- **Albums table:** This table will store album information such as name, release date, and image. It will also establish a foreign key relationship with the Artists table.

- **Tracks table:** This table will store track details like name, duration, and file path. It will establish a foreign key relationship with the Albums table.

- **Playlists table:** This table will contain information about user-created playlists, including name and image. It will establish a foreign key relationship with the Users table.

- **Playlist_Tracks table:** This table will serve as an association between playlists and tracks. It will include foreign keys linking to the Playlists and Tracks tables.

- **Followers table:** This table will establish the relationship between users and artists, allowing users to follow multiple artists. It will include foreign keys linking to the Users and Artists tables.

- **Likes table:** This table will store information about tracks liked by users, including the date and time of the like. It will include foreign keys linking to the Users and Tracks tables.

By organizing the data into these tables, we can effectively manage user interactions, artist information, music albums, playlists, and more.

2. Entity-Relationship Diagram (ERD):

To provide a visual representation of the database structure, an Entity-Relationship Diagram (ERD) will be created. The ERD will illustrate the relationships between tables, primary and foreign keys, and cardinality of these relationships. This diagram will aid in understanding the database structure at a glance and facilitate further development and optimization.

3. Data Types and Constraints:

Each column within the tables will be assigned appropriate data types and constraints to ensure data integrity and optimize storage. For example:

- **User_ID** in the Users table will be an integer primary key with an auto-increment feature.

- **Name** in the Users and Artists tables will be of type VARCHAR, restricting the length to a suitable value.

- **Foreign key** constraints will be defined to enforce referential integrity between related tables.

4. Data Normalization:

The database design will follow the principles of data normalization to eliminate redundancy and improve data integrity. By decomposing data into separate tables, we will achieve higher efficiency in data storage, updates, and retrieval.

5. Indexing Strategy:

To optimize query performance, appropriate indexing will be employed. Columns frequently used in search conditions or joins, such as User_ID, Artist_ID, and Track_ID, will be indexed to accelerate data retrieval. Careful consideration will be given to the type of indexes (e.g., B-tree, Hash, or GiST) based on specific use cases and requirements.

6. Additional Features:

In addition to the core functionalities, we will incorporate the following features into our Spotify database design:

- **Premium User Feature:** A "premium user" functionality will be implemented to differentiate between regular and premium users. This feature will involve adding a user type column to the Users table, as well as creating tables to store premium features and their associations with users.

- **Payment System:** To handle the payment system, a Payment table will be created to store payment information. A Subscription_Plan table will also be implemented to manage different subscription options. Users will be associated with their respective subscription plans through a junction table.

- **Recommendation Feature:** A recommendation system will be incorporated, leveraging pre-computed similarity metrics between songs and users. The Similarity table will store these metrics, and recommendations can be generated based on a user's listening history. SQL queries will be designed to retrieve personalized recommendations.

- **Notification Feature:** To handle notifications, a Notification table will be created to store relevant information such as user ID, title, content, and creation date. Users can retrieve their notifications by querying this table.

7. Implementation using PostgreSQL:

The entire database design will be implemented using PostgreSQL, an open-source and highly extensible relational database management system. PostgreSQL provides comprehensive features for data modeling, query optimization, and scalability, making it an ideal choice for building a robust and efficient database for Spotify.

Conclusion:

By designing and implementing a well-structured database for Spotify using PostgreSQL, we aim to provide an optimized platform for managing user interactions, artist information, music albums, playlists, recommendations, and notifications. Through careful consideration of database design principles, normalization, indexing strategies, and the integration of additional features, we will create a powerful foundation for a high-performance Spotify-like system.

The End.

Special Thanks to Dr. Shoja for all his efforts.