Ece Esmer, Ilnaza Saifutdinova, Neilya Burkitbayeva
Software Engineering, Group B

Spotify Database Design Project

Abstract

This project document describes a relational database modeled after Spotify, a music streaming service. The main goal of the database is to organize and manage data about users, artists, albums, songs, and playlists. The document starts by identifying the main parts of the database: Users, Artists, Albums, Songs, and Playlists, along with a Playlist_Song table that connects songs and playlists in many-to-many relationships.

The schema section describes the SQL configuration for various components, including tables with keys, constraints, and data types to ensure data consistency and quick searches.
An Entity Relationship Diagram (ERD) is given to visually explain how these components are connected, with a focus on the one-to-many and many-to-many relationships required for the Spotify-like service's operation.

According to the publication, the database is structured using all three levels of normalization, up to the Third Normal Form (3NF), to reduce unnecessary data duplication and ensure correctness and efficiency.

Overview

Database Model Design

Description.

1. What is Data?

In simple words data can be facts related to any object inconsideration. For example, a picture, image, file, or PDF, etc. can also be considered data.

2. What is a Database?

Any Data we have can be random. So, a Database is a systematic collection of certain Data. Since the Data in a Database is organized it makes Data Management easy.

3. What is a Database Management System (DBMS)?

Database Management System (DBMS) is a collection of programs which enables its users to access Database, manipulate Data, and help on representation of Data. It also helps control access to the Database by various users.

For this project we chose Spotify. The Spotify database is a dynamic, relational database system that underpins the entire Spotify platform. Spotify Database needs to store, manipulate, and present Data related to user information, music data, playlist and user activities, relationships and integrations.

Here's a descriptive overview of its components:

User Information

- Users: The database holds data on every user, such as user IDs, usernames, email addresses, passwords, subscription plans and additional personal information. This enables Spotify to handle user accounts, preferences and subscriptions.

Music Data

- Artist:: It contains information about artists, such as their artist IDs, names and genres. This feature helps users discover and follow their artists effortlessly.
- Albums: Each album is assigned an ID, title and release date. Is connected to an artist. This system allows Spotify to neatly arrange and showcase albums in a way.
- Songs: Songs are associated with both albums and artists providing details like song IDs, titles, durations, genres and release dates. This data plays a role in playing songs and generating music recommendations.

Playlists and User Activities

- Playlists: Users have the option to make playlists each, with its identification code, title, date of creation, brief description and privacy preferences. These playlists can consist of songs and users also have the ability to subscribe to playlists crafted by users.

Relationships and Integrations

- Foreign Keys and Relationships: The database features relationships through foreign keys to uphold the integrity of data. For instance songs are associated with albums and artists playlists connect to users and user history connects to songs.
- Composite Keys: In tables such as Playlist_Song, composite keys are utilized to handle the too many connections between playlists and songs.

SQL

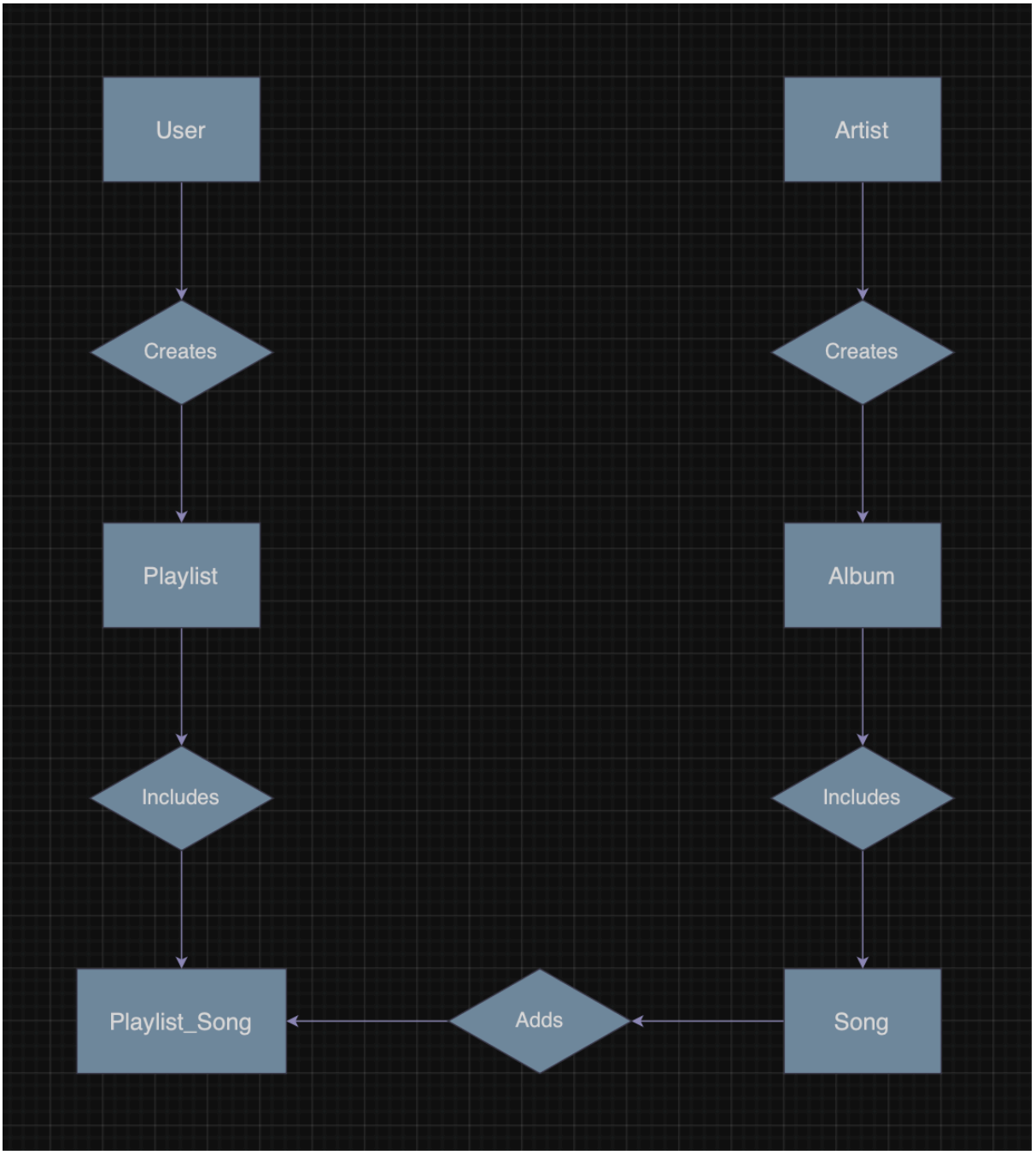
```sql
CREATE TABLE User (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL
);

CREATE TABLE Artist (
    artist_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(255) NOT NULL,
    genre VARCHAR(50)
);
```

```sql
CREATE TABLE Album (
    album_id INTEGER PRIMARY KEY AUTOINCREMENT,
    title VARCHAR(255) NOT NULL,
    release_date DATE NOT NULL,
    artist_id INTEGER,
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id)
);

CREATE TABLE Song (
    song_id INTEGER PRIMARY KEY AUTOINCREMENT,
    title VARCHAR(255) NOT NULL,
    duration INTEGER NOT NULL,
    artist_id INTEGER,
    album_id INTEGER,
    FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);
```

```sql
CREATE TABLE Playlist (
    playlist_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(255) NOT NULL,
    user_id INTEGER,
    FOREIGN KEY (user_id) REFERENCES User(user_id)
);

CREATE TABLE Playlist_Song (
    playlist_id INTEGER,
    song_id INTEGER,
    PRIMARY KEY (playlist_id, song_id),
    FOREIGN KEY (playlist_id) REFERENCES Playlist(playlist_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id)
);
```

Schema Creation & Entity Relationship Diagram

Database Schema



Entities:

- User: Represents a registered user of the service.
- Artist: Represents a musical artist or group.
- Song: Represents an individual song.
- Album: Represents a collection of songs by an artist or group of artists.
- Playlist: Represents a list of songs selected and organized by the user.
- Playlist_Song: Represents a list of songs selected individually by each user.

Attributes:

User	user_id (INT, Primary Key)
	username (VARCHAR(20), NOT NULL, Unique Key)
	email (VARCHAR(50), NOT NULL, Unique Key)
	password (VARCHAR(100), NOT NULL)
	subscription_type (ENUM('Free', 'Premium'))

The entity “User” includes five attributes. There are “user_id”, “username”, “email”, “password”, and “subscription_type”.

“User” attributes are simple attributes which means all of them are single, indivisible attributes.

Artist	artist_id (INT, Primary Key)
	name (VARCHAR(255), NOT NULL)
	genre (VARCHAR(50), NOT NULL)

The entity “Artist” includes three attributes. There are “artist_id”, “name”, and “genre”.

“Artist” attributes are simple attributes which means all of them are single, indivisible attributes.

Album	album_id (INT, Primary Key)
	title (VARCHAR(255), NOT NULL)
	release_date (DATE)
	artist_id (INT, Foreign Key references Artist.artist_id)

The entity “Album” includes four attributes. There are album_id, title, release_date, artist_id.

Simple attributes: “album_id”, “title”, “release_date”.

Foreign key: “artist_id” is a foreign key referencing the “artist_id” in the “Artist” table.

Song	song_id (INT, Primary Key)
	title (VARCHAR(255), NOT NULL)
	duration (INT)
	artist_id (INT, Foreign Key references Artist.artist_id)
	album_id (INT, Foreign Key references Album.album_id)

The entity “Song” includes four attributes. There are “song_id”, “title”, “duration” - in seconds, “album_id”.

Simple attributes: "song_id", "title", and "duration" are simple attributes.

Foreign key: “artist_id” is a foreign key referencing the “artist_id” in the “Artist” table, “album_id” is a foreign key referencing the “album_id” in the “Album” table.

Playlist	playlist_id (INT, Primary Key)
	name (VARCHAR(255), NOT NULL)

	user_id (INT, Foreign Key references User.user_id)

The entity “Playlist” includes three attributes. There are “playlist_id”, “name”, “user_id”.

Simple attributes: "playlist_id", and "name".

Foreign key: “user_id” is a foreign key referencing the “user_id” in the “User” table.

Playlist_Song:	playlist_id (INT, Foreign Key references Playlist.playlist_id)
	song_id (INT, Foreign Key references Song.song_id)
	Primary Key (playlist_id, song_id)

The entity “Playlist_Song” includes three attributes. There are “playlist_id”, “song_id”.

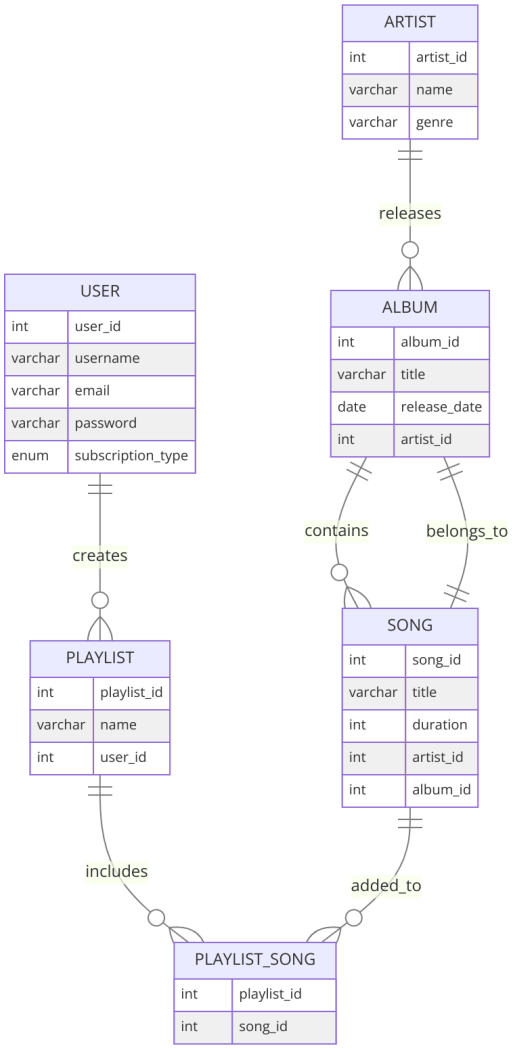
Composite Attribute: The combination of “playlist_id” and “song_id” forms a composite Primary Key for this table.

Foreign key: “playlist_id” is a foreign key referencing the “playlist_id” in the “Playlist” table, “song_id” is a foreign key referencing the “song_id” in the “Song” table.

Relationships:

- One Artist can have Many Songs (One-to-Many Relationship)
- One Album can have Many Songs (One-to-Many Relationship)
- One User can create Many Playlists (One-to-Many Relationship)
- A Song can belong to One Album (Many-to-One Relationship)
- A Song belongs to One Artist (Many-to-One Relationship)
- A Playlist can contain Many Songs (Zero-to-Many Relationship) - Implemented with a separate table.
- Additional
- Table (Many-to-Many Relationship)

Entity Relationship Diagram



Normalization & Documentation

Levels of Normalization

First Normal Form (1NF)

1. Requirements:
    - Ensuring each table has a primary key
    - Ensuring each column contains atomic values
    - Ensuring each column contains values of a single type
    - Ensuring each record is unique

2. Implementation of these entities:
    - User: Already in 1NF as all attributes are atomic, have single values, and user_id is the primary key
    - Artist: Already in 1NF with atomic attributes and artist_id as the primary key
    - Album: Already in 1NF with atomic attributes and album_id as the primary key
    - Song: Already in 1NF with atomic attributes and song_id as the primary key.
    - Playlist: Already in 1NF with atomic attributes and playlist_id as the primary key
    - Playlist_Song: Already in 1NF with atomic attributes and a composite primary key (playlist_id, song_id)

Second Normal Form (2NF)

1. Requirements:
    - The table must be in 1NF.
    - All non-key attributes must be fully functionally dependent on the entirety of the primary key (no partial dependencies).

2. Implementation for given entities:
    - User: Already in 2NF as all non-key attributes are fully dependent on the primary key user_id.
    - Artist: Already in 2NF as all non-key attributes are fully dependent on the primary key artist_id.
    - Album: Already in 2NF as all non-key attributes are fully dependent on the primary key album_id.
    - Song: Already in 2NF as all non-key attributes are fully dependent on the primary key song_id.
    - Playlist: Already in 2NF as all non-key attributes are fully dependent on the primary key playlist_id.
    - Playlist_Song: Already in 2NF as the non-key attributes are fully dependent on the composite primary key (playlist_id, song_id).

Third Normal Form (3NF)

1. Requirements:
    - The table must be in 2NF.
    - There should be no transitive dependencies (non-key attributes depending on other non-key attributes).

2. Implementation for given entities:
    - User: Already in 3NF as there are no transitive dependencies.
    - Artist: Already in 3NF as there are no transitive dependencies.
    - Album: Already in 3NF as there are no transitive dependencies.
    - Song: Already in 3NF as there are no transitive dependencies.
    - Playlist: Already in 3NF as there are no transitive dependencies.
    - Playlist_Song: Already in 3NF as there are no transitive dependencies.

Summary

The main focus of this project is to create a database model for data gathered by Spotify. Furthermore, the project aims to focus on handling and organizing big datasets for user activity, playlists, music information, and user profiles.

In terms of the concepts of Data and Databases, facts related to any item, including images, files and PDFs, are referred to as data. Moreover, users can access, modify, and manage data in a database with the help of a Database administration System (DBMS).

The Design of the Spotify Database consists of variables including User Data (keeps track of usernames, passwords, email addresses, user IDs, and subscription plans), Music Data (contains information on artists (IDs, names, genres), albums (titles, release dates, and album IDs), and songs (lengths, genres, and song IDs)), Playlists and User Activities (contains information on users preferences), and Relationships and Integrations (contains information on types of keys such as primary, foreign, etc.)

Putting everything into nutshell, the architecture of the Spotify database is crafted to manage amounts of information linked to user profiles, music selections and user engagements. Through this framework Spotify can maintain top notch performance, data reliability. Deliver a smooth and tailored music streaming experience, for its users.