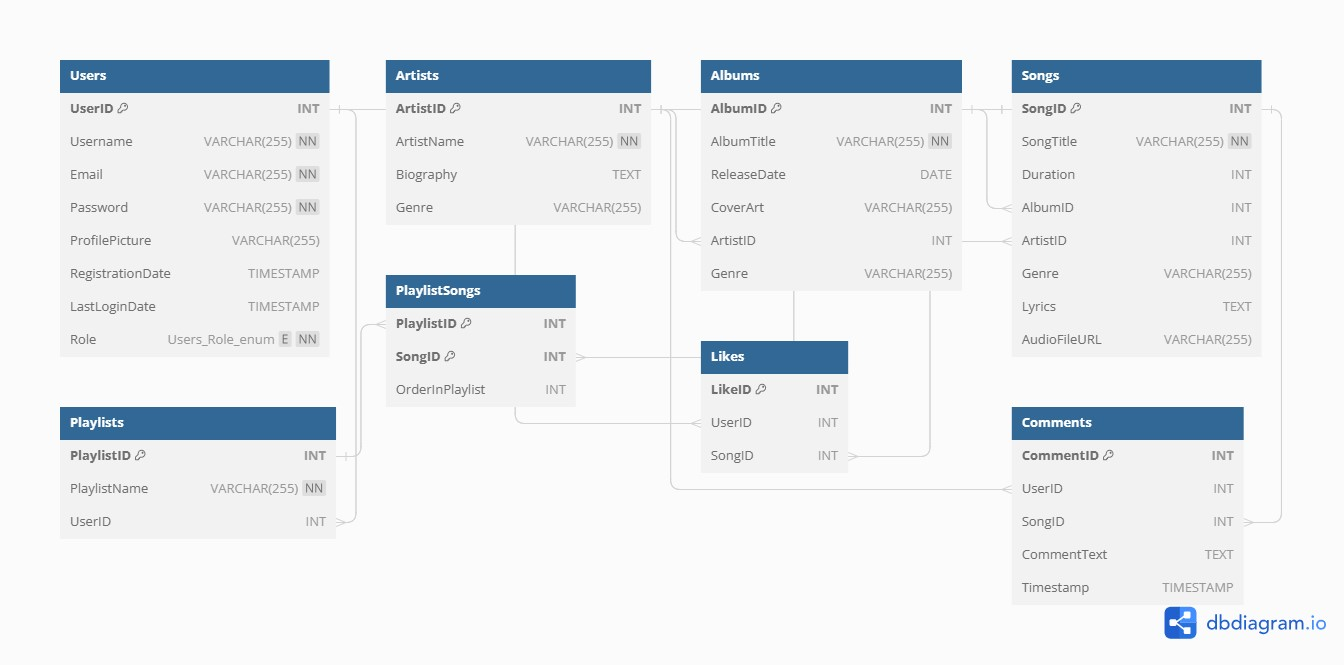
**Appdev1**



Suggestion from chatgpt

**Entities and Their Attributes:**

* **Users**
* UserID (Primary Key)
* Username
* Email
* Password (hashed)
* Profile Picture
* Registration Date
* Last Login Date
* Role (Admin, Creator, User)
* **Artists**
* ArtistID (Primary Key)
* Artist Name
* Biography
* Genre
* **Albums**
* AlbumID (Primary Key)
* Album Title
* Release Date
* Cover Art
* ArtistID (Foreign Key)
* Genre
* **Songs**
* SongID (Primary Key)
* Song Title
* Duration
* AlbumID (Foreign Key)
* ArtistID (Foreign Key)
* Genre
* Lyrics
* Audio File URL
* **Playlists**
* PlaylistID (Primary Key)
* Playlist Name
* UserID (Foreign Key)
* **PlaylistSongs (Many-to-Many Relationship Table)**
* PlaylistID (Foreign Key)
* SongID (Foreign Key)
* Order (to specify the order of songs in a playlist)
* **Likes (Tracks which songs a user has liked)**
* LikeID (Primary Key)
* UserID (Foreign Key)
* SongID (Foreign Key)
* **Comments (User comments on songs)**
* CommentID (Primary Key)
* UserID (Foreign Key)
* SongID (Foreign Key)
* Comment Text
* Timestamp

**Relationships:**

* Each user can have multiple playlists, so there's a one-to-many relationship between Users and Playlists (UserID in Playlists table).
* Each playlist can contain multiple songs, and each song can belong to multiple playlists, which is a many-to-many relationship managed through the PlaylistSongs table.
* Users can like multiple songs, creating a many-to-many relationship between Users and Songs through the Likes table.
* Users can comment on multiple songs, creating a one-to-many relationship between Users and Comments. Comments are related to songs through the SongID.

**Indexes:**

You should also consider creating indexes on commonly queried columns, such as UserID, ArtistID, AlbumID, and SongID, to optimize database performance.

This is a simplified database schema for a music app, and depending on the specific features and complexity of your application, you may need to expand upon or modify this schema. Additionally, you should implement proper data validation and security measures to protect user data and ensure the integrity of your database.

-- Create the Users table

CREATE TABLE Users (

UserID INT PRIMARY KEY AUTO\_INCREMENT,

Username VARCHAR(255) NOT NULL,

Email VARCHAR(255) NOT NULL,

Password VARCHAR(255) NOT NULL,

ProfilePicture VARCHAR(255),

RegistrationDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

LastLoginDate TIMESTAMP,

Role ENUM('Admin', 'Creator', 'User') NOT NULL

);

-- Create the Artists table

CREATE TABLE Artists (

ArtistID INT PRIMARY KEY AUTO\_INCREMENT,

ArtistName VARCHAR(255) NOT NULL,

Biography TEXT,

Genre VARCHAR(255)

);

-- Create the Albums table

CREATE TABLE Albums (

AlbumID INT PRIMARY KEY AUTO\_INCREMENT,

AlbumTitle VARCHAR(255) NOT NULL,

ReleaseDate DATE,

CoverArt VARCHAR(255),

ArtistID INT,

Genre VARCHAR(255),

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

-- Create the Songs table

CREATE TABLE Songs (

SongID INT PRIMARY KEY AUTO\_INCREMENT,

SongTitle VARCHAR(255) NOT NULL,

Duration INT,

AlbumID INT,

ArtistID INT,

Genre VARCHAR(255),

Lyrics TEXT,

AudioFileURL VARCHAR(255),

FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID),

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

-- Create the Playlists table

CREATE TABLE Playlists (

PlaylistID INT PRIMARY KEY AUTO\_INCREMENT,

PlaylistName VARCHAR(255) NOT NULL,

UserID INT,

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

-- Create the PlaylistSongs table (for the many-to-many relationship)

CREATE TABLE PlaylistSongs (

PlaylistID INT,

SongID INT,

OrderInPlaylist INT,

PRIMARY KEY (PlaylistID, SongID),

FOREIGN KEY (PlaylistID) REFERENCES Playlists(PlaylistID),

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);

-- Create the Likes table

CREATE TABLE Likes (

LikeID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT,

SongID INT,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);

-- Create the Comments table

CREATE TABLE Comments (

CommentID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT,

SongID INT,

CommentText TEXT,

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);

**Storing the Song**

**import json**

**# Define song metadata as a dictionary**

**song\_metadata = {**

**"title": "Song Title",**

**"artist": "Artist Name",**

**"lyrics": "Song Lyrics in UTF-8 Text",**

**}**

**# Encode the metadata as UTF-8 JSON**

**metadata\_json = json.dumps(song\_metadata, ensure\_ascii=False).encode('utf-8')**

**# Store the UTF-8 encoded JSON in a file**

**with open('song\_metadata.json', 'wb') as json\_file:**

**json\_file.write(metadata\_json)**

**# Read the UTF-8 encoded JSON file**

**with open('song\_metadata.json', 'rb') as json\_file:**

**retrieved\_metadata\_json = json\_file.read()**

**# Decode the JSON back to a dictionary**

**retrieved\_metadata = json.loads(retrieved\_metadata\_json.decode('utf-8'))**

**# Access song metadata fields**

**print("Title:", retrieved\_metadata["title"])**

**print("Artist:", retrieved\_metadata["artist"])**

**print("Lyrics:", retrieved\_metadata["lyrics"])**

**SQL Lite 3 db**

-- Create the Users table

CREATE TABLE Users (

UserID INTEGER PRIMARY KEY,

Username TEXT NOT NULL,

Email TEXT NOT NULL,

Password TEXT NOT NULL,

ProfilePicture TEXT,

RegistrationDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

LastLoginDate TIMESTAMP,

Role TEXT NOT NULL

);

-- Create the Artists table

CREATE TABLE Artists (

ArtistID INTEGER PRIMARY KEY,

ArtistName TEXT NOT NULL,

Biography TEXT,

Genre TEXT

);

-- Create the Albums table

CREATE TABLE Albums (

AlbumID INTEGER PRIMARY KEY,

AlbumTitle TEXT NOT NULL,

ReleaseDate DATE,

CoverArt TEXT,

ArtistID INTEGER,

Genre TEXT,

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

-- Create the Songs table

CREATE TABLE Songs (

SongID INTEGER PRIMARY KEY,

SongTitle TEXT NOT NULL,

Duration INTEGER,

AlbumID INTEGER,

ArtistID INTEGER,

Genre TEXT,

Lyrics TEXT,

AudioFileURL TEXT,

FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID),

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

-- Create the Playlists table

CREATE TABLE Playlists (

PlaylistID INTEGER PRIMARY KEY,

PlaylistName TEXT NOT NULL,

UserID INTEGER,

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

-- Create the PlaylistSongs table (for the many-to-many relationship)

CREATE TABLE PlaylistSongs (

PlaylistID INTEGER,

SongID INTEGER,

OrderInPlaylist INTEGER,

PRIMARY KEY (PlaylistID, SongID),

FOREIGN KEY (PlaylistID) REFERENCES Playlists(PlaylistID),

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);

-- Create the Likes table

CREATE TABLE Likes (

LikeID INTEGER PRIMARY KEY,

UserID INTEGER,

SongID INTEGER,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);

-- Create the Comments table

CREATE TABLE Comments (

CommentID INTEGER PRIMARY KEY,

UserID INTEGER,

SongID INTEGER,

CommentText TEXT,

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);