Project Musica

Song management system

Members

1. Ashita Boyina - 2019028
2. Rishi Raj - 2019090
3. Akshat Wadhwa - 2019231
4. Mudit Balooja - 2019258
5. Navam Chaurasia - 2019314

Link to the repository (private) where the CSV and sql files are stored:
https://github.com/Vikinghacker258/Music-Management-System

Week 1

We came up with ideas for our project. After discussing various ideas, we selected song management as its part of our daily life and quite relatable.

Abstract

In today's world, music enthusiasts are facing issues on a daily basis. Due to the amount of applications designed for the purpose of spreading music across the globe, there has been a lot of conflict in people's minds. The answer to the question often changes with time, and will be different for every individual. As a result, if people want to share their likes or playlists, it often causes problems, as different people might be using different applications, and hence, it becomes cumbersome to actually listen to songs.

The problems do not end here. In our country, many music enthusiasts are not able to attend concerts near them, due to the lack of awareness, and lesser reach of currently existing event organizers. Many budding artists also do not get a chance to reach the big stage, and a huge amount of talent goes wasted (optional).

Objective

The objective of this project is to create a system for people to easily search songs. It also enables the artists to publish their songs and albums for the world to see. It consists of many unique features for example, users can like their favorite songs and follow their favourite artists or create a playlist to save and listen to a set of songs on repeat. Artists can also keep track of the statistics associated with their songs and albums. Musica also gives budding businesses a chance to advertise their business. For unlocking the full potential of Musica, users can opt for a premium membership, which will guarantee them benefits

over regular listeners like special discounts on booking tickets for concerts and events and they can also enjoy their favourite music without the interruption of ads.

Main features

People/users can listen to songs or like them if they like the song or unlike if they used to like it but not anymore. Artists can add their songs to the app and see how many people have listened to it. If a live event/concert is organised where the artists perform, people can buy tickets to the said event and go listen to songs sung by the artist along with other people in the same place at the same time.

Week 2

Deciding the main stakeholders and their attributes and information required.

Various Stakeholders

1. Users/Listeners
2. Artists
3. Administrator
4. Concert organisers
5. Advertisers
6. Manager

Entity Information

I. Users

1. UserID
2. Name
3. Number of songs played
4. Status - Premium/Free user
5. Location
6. Balance
7. Events booked
8. Features for premium
    a. Create own playlists
    b. No ads
    c. Price for concerts is 25% reduced

II. Song

1. SongID
2. ArtistID
3. AlbumID
4. SongName
5. Year of release
6. Genre
7. Ratings

8. Length

III. Artist
1. ArtistName
2. ArtistID
3. Monthly listeners
4. Total Albums
5. Total Songs
6. Location-country and city
7. Balance

IV. Album
1. AlbumName
2. ArtistID
3. AlbumID
4. Release year

V. Admin
1. AdminID
2. AdminName
3. Password
4. Joining Date
5. City
6. Country

VI. Event organiser
1. Company name
2. OrganiserID
3. Balance
4. Event schedule(past and upcoming)

VII. Event
1. EventName
2. EventID
3. Location
4. Date/Time
5. Number of tickets sold
6. Total users allowed
7. Event organiserID
8. Artists performing

VIII. Manager
1. ManagerID
2. ManagerName
3. ManagerCity
4. ManagerCountry

5. Password
6. Balance
7. Amount pending
8. Amount collected

IX. Ticket(Weak)
1. UserID
2. EventID
3. Price

X. Advertisers
1. Balance
2. Name,
3. AdvertiserID
4. Total number of clicks/views on all their ads

XI. Ads(Weak)
1. Name
2. AdvertiserID
3. Number of clicks/views
4. Price

XII. Playlists
1. PlaylistId
2. Name
3. UserID
4. Number of songs
5. ManagerID

List of various Queries .

Admin
1. Enables advertisements-feature
2. Analysis of the concerts and views
3. Manage advertisements

Users
1. Search by name, album, artist, year, genre
2. Upgrade to premium
3. Create a playlist
4. Edit(Add/ remove a song) playlist
5. Can like/unlike a song
6. Follow/unfollow an artist
7. Pay for concert tickets
8. List all upcoming and current event(concerts)

9. List all booked events

Artist

1. *Create,view* (likes, searches,streams,statistics(monthly,yearly or weekly)), a song
2. Edit(delete/update) details of their songs/albums
3. Bookings done for concerts
4. View all their previous, upcoming concerts
5. Check amount in their balance
6. List all scheduled events where he/she needs to perform

Event/concert organisers

1. Create,view a new event
2. Edit an event(update trailers,songs etc/remove it)
3. Manage booking (open / close the registrations)
4. Select artists for concerts
5. Check amount in their balance
6. List all upcoming and current event(concerts)
7. View tickets sold for events

Advertisers

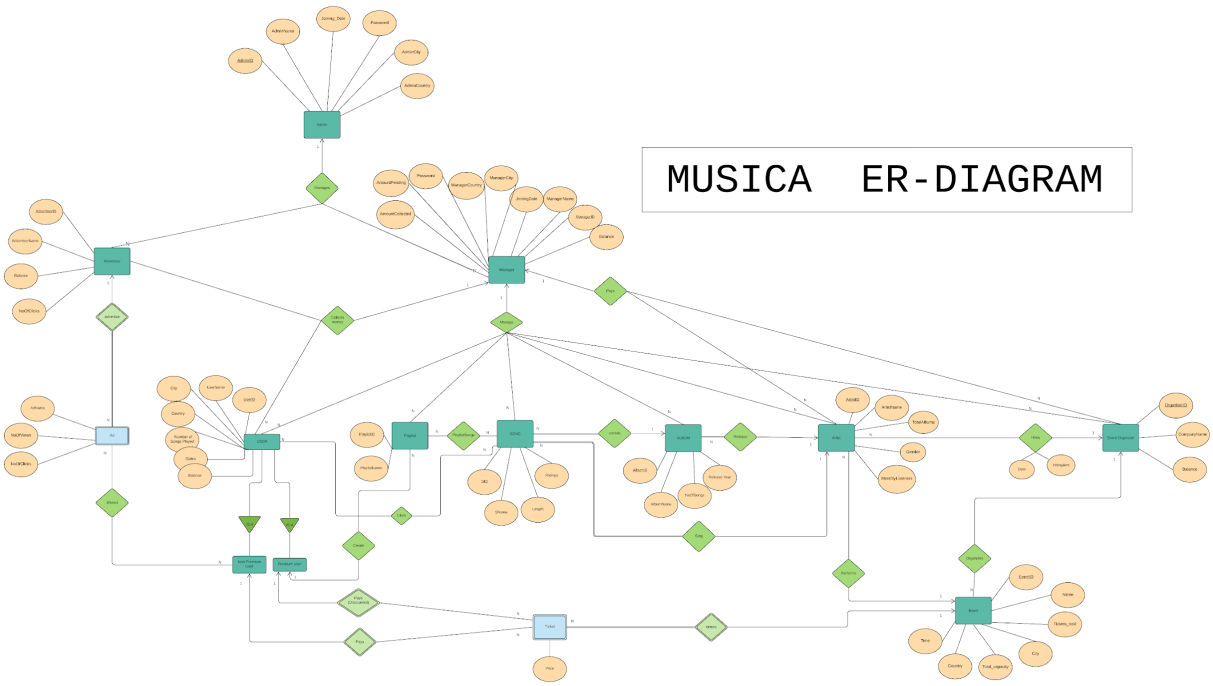
1. Create/view/edit their ads
2. View number of times seen or clicks on ad
3. Pay money to manager according to viewership

Managers

1. Create and suggest playlists to a set of users based on their likes/view history
2. Premium members -features
3. Collect money from advertisers
4. Collect money from users for concerts
5. Pay money to event organizers and artist
6. List all premium members till a particular date

Week 3

Entities and Attributes



[Link to board]

Week 4

WEAK entities.

- Tickets
- Ads

Admin manages advertisers and managers.
Manager manages playlists,users,event organizers, songs, albums and artists.
Manager collects money from users and advertisers
Manager pays to artists and event organisers

Event Organizers organises events.

Artists perform in events.
Events have tickets
Artists sing songs

Users(premium) create/play playlists
Users like/unlike songs
Users buy event tickets
Users buy premium

Songs appear in Albums
Songs included in playlists

Ads shown to users
Advertisers create/edit ads(add its table)

Admin with manager-one to one
Manger to any another-one to many
Admin to any other -one to many
Event organizer to artist is one to many
Event to artist -one to one
Any other entity with any other entity(if exists) - many to many

Week 5.

- ER diagram was updated.
- Some more relationships were defined and redundant ones were removed.
- Entity attributes were optimised and updates
- Primary keys were defined for entities
- Tables were created and data added.

Schemas (Primary key, *Foreign key*)

1. Admin(AdminID int, AdminName varchar(45), Joining_Date varchar(45), Password varchar(45), AdminCity varchar(45), AdminCountry varchar(45))

2. Advertiser(AdvertiserID int,Name varchar(45),Balance float,NoOfClicks int)

3. Ads(Name varchar(45), *AdvertiserID int*, NoOfViews int, NoOfClicks int)

4. Albums(AlbumID int, Name varchar(45), Number Of Songs int, Release Year int, *ArtistID int)*

5. Artists(ArtistID int, Name varchar(45), Gender varchar(8), MonthlyListeners int, TotalAlbums)

6. EventOrganiser(OrganiserID int, CompanyName varchar(45), Balance int)

7. Event(EventID int, Name varchar(45), City varchar(45), Country varchar(4), Time varchar(45), Tickets_sold int, Total_capacity int, *EventOrganiserID int*, *ArtistID int*)

8. Manager(ManagerID int, ManagerName varchar(45), JoiningDate varchar(45), ManagerCity varchar(45), ManagerCountry varchar(45), Password varchar(45), Balance int, AmountPending int, AmountCollected int)

9. Playlist(PlaylistID int, PlaylistName varchar(45), *UserID int*)

10. PlaylistSongs(*PlaylistID int*, *SongID int*)

11. Songs(SID int, Sname varchar(45), AlbumName varchar(45), Length int, Ratings int, Release Year int, *AlbumID int*, *ArtistID int*)

12. Tickets(*UserID int*, *EventID int*, Price int)

13. Users(UserID int, Name varchar(45), Status varchar(45), City varchar(45), Country varchar(45), Balance int, Number of Songs Played int)

Alter Table commands:

1. Alter command for ADMIN

```
ALTER TABLE `musica`.`admin`
CHANGE COLUMN `AdminID` `AdminID` INT NOT NULL ,
CHANGE COLUMN `AdminName` `AdminName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Joining_Date` `Joining_Date` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Password` `Password` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `AdminCity` `AdminCity` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `AdminCountry` `AdminCountry` VARCHAR(100) NOT NULL ,
ADD PRIMARY KEY (`AdminID`);
;
```

2. Alter command for MANAGER

```
ALTER TABLE `musica`.`manager`
CHANGE COLUMN `ManagerID` `ManagerID` INT NOT NULL ,
CHANGE COLUMN `ManagerName` `ManagerName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `JoiningDate` `JoiningDate` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `ManagerCity` `ManagerCity` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `ManagerCountry` `ManagerCountry` VARCHAR(100) NOT NULL
,
CHANGE COLUMN `Password` `Password` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Balance` `Balance` FLOAT NOT NULL ,
CHANGE COLUMN `AmountPending` `AmountPending` FLOAT NOT NULL ,
CHANGE COLUMN `AmountCollected` `AmountCollected` FLOAT NOT NULL ,
ADD PRIMARY KEY (`ManagerID`);
;
```

3. Alter command for ADVERTISER

```
ALTER TABLE `musica`.`advertisers`
CHANGE COLUMN `AdvertiserID` `AdvertiserID` INT NOT NULL ,
CHANGE COLUMN `Name` `Name` TEXT NOT NULL ,
CHANGE COLUMN `Balance` `Balance` INT NOT NULL DEFAULT 0 ,
CHANGE COLUMN `Total Number Of clicks` `Total Number Of clicks` INT NOT
NULL DEFAULT 0 ,
ADD PRIMARY KEY (`AdvertiserID`);
;
```

4. Alter command for ADVERTISEMENT

```
ALTER TABLE `musica`.`advertisers`
CHANGE COLUMN `Name` `AdvertiserName` TEXT NOT NULL ;   // DOUBT
advertiser

ALTER TABLE `musica`.`advertisements`
CHANGE COLUMN `AdvertiserID` `AdvertiserID` INT NOT NULL ,
CHANGE COLUMN `Name` `AdsName` VARCHAR(45) NOT NULL ,
CHANGE COLUMN `Number of clicks/views` `Number of clicks/views` INT NOT
NULL DEFAULT 0 ,
ADD PRIMARY KEY (`AdsName`);
;
ALTER TABLE `musica`.`advertisements`
ADD CONSTRAINT `AdvertiserID`
  FOREIGN KEY (`AdvertiserID`)
  REFERENCES `musica`.`advertisers` (`AdvertiserID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```

5. Alter command for ARTIST

```
ALTER TABLE `musica`.`artists`
CHANGE COLUMN `ID` `ArtistID` INT NOT NULL ,
CHANGE COLUMN `Name` `ArtistName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Gender` `Gender` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `MonthlyListeners` `MonthlyListeners` INT NOT NULL DEFAULT
0 ,
CHANGE COLUMN `TotalAlbums` `TotalAlbums` INT NOT NULL ,
CHANGE COLUMN `TotalSongs` `TotalSongs` INT NOT NULL DEFAULT 0 ,
CHANGE COLUMN `Location` `City` VARCHAR(50) NULL DEFAULT NULL ,
CHANGE COLUMN `Balance` `Country` VARCHAR(50) NOT NULL ,
CHANGE COLUMN `MyUnknownColumn` `Balance` INT NOT NULL ,
ADD PRIMARY KEY (`ArtistID`);
;
```

6. Alter command for ALBUM

```
ALTER TABLE `musica`.`albums`
CHANGE COLUMN `AlbumID` `AlbumID` INT NOT NULL ,
CHANGE COLUMN `Name` `AlbumName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Number Of Songs` `NumberOfSongs` INT NOT NULL ,
CHANGE COLUMN `Release Year` `ReleaseYear` INT NOT NULL ,
CHANGE COLUMN `ArtistID` `ArtistID` INT NOT NULL ,
ADD PRIMARY KEY (`AlbumID`),
ADD INDEX `ArtistID_idx` (`ArtistID` ASC) VISIBLE;
;
ALTER TABLE `musica`.`albums`
ADD CONSTRAINT `ArtistID`
```

```
  FOREIGN KEY (`ArtistID`)
  REFERENCES `musica`.`artists` (`ArtistID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```

7. Alter command for EVENT-ORGANISER

```
ALTER TABLE `musica`.`eventorganisers`
CHANGE COLUMN `ID` `ID` INT NOT NULL ,
CHANGE COLUMN `CompanyName` `CompanyName` TEXT NOT NULL ,
CHANGE COLUMN `Balance` `Balance` INT NOT NULL ,
ADD PRIMARY KEY (`ID`);
;


ALTER TABLE `musica`.`eventorganisers`
CHANGE COLUMN `ID` `EventOrganiserID` INT NOT NULL ;
```

8. Alter command for EVENT

```
ALTER TABLE `musica`.`event`
CHANGE COLUMN `ID` `EventID` INT NOT NULL ,
CHANGE COLUMN `Name` `EventName` TEXT NOT NULL ,
CHANGE COLUMN `City` `City` TEXT NOT NULL ,
CHANGE COLUMN `Country` `Country` TEXT NOT NULL ,
CHANGE COLUMN `Time` `Date` TEXT NOT NULL ,
CHANGE COLUMN `Tickets_sold` `TicketsSold` INT NOT NULL DEFAULT 0 ,
CHANGE COLUMN `Total_capacity` `TotalCapacity` INT NOT NULL ,
CHANGE COLUMN `EventOrganiserID` `EventOrganiserID` INT NOT NULL ,
CHANGE COLUMN `ArtistID` `ArtistID` INT NOT NULL ,
ADD PRIMARY KEY (`EventID`),
ADD INDEX `ArtistID_idx` (`ArtistID` ASC) VISIBLE,
ADD INDEX `EventOrganiserID_idx` (`EventOrganiserID` ASC) VISIBLE;
;
ALTER TABLE `musica`.`event`
ADD CONSTRAINT `ArtistID`
  FOREIGN KEY (`ArtistID`)
  REFERENCES `musica`.`artists` (`ArtistID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
ADD CONSTRAINT `EventOrganiserID`
  FOREIGN KEY (`EventOrganiserID`)
  REFERENCES `musica`.`eventorganisers` (`EventOrganiserID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```

```
ALTER TABLE `musica`.`event`
ADD INDEX `EventOrganiserID_idx` (`EventOrganiserID` ASC) VISIBLE,
DROP INDEX `ArtistID_UNIQUE` ;
;
ALTER TABLE `musica`.`event`
ADD CONSTRAINT `EventOrganiserID`
  FOREIGN KEY (`EventOrganiserID`)
  REFERENCES `musica`.`eventorganisers` (`EventOrganiserID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `musica`.`event`
ADD CONSTRAINT `ArtistID`
  FOREIGN KEY (`ArtistID`)
  REFERENCES `musica`.`artists` (`ArtistID`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```

9. Alter command for SONGS

```
ALTER TABLE `musica`.`songs`
CHANGE COLUMN `SongName` `SongName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `AlbumName` `AlbumName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Ratings` `Ratings` FLOAT NOT NULL ;
```

10. Alter command for TICKETS

```
ALTER TABLE `musica`.`tickets`
CHANGE COLUMN `UserID` `UserID` INT NOT NULL ,
CHANGE COLUMN `EventID` `EventID` INT NOT NULL ,
CHANGE COLUMN `Price` `Price` FLOAT NOT NULL ;
```

11. Alter command for USERS

```
ALTER TABLE `musica`.`users`
CHANGE COLUMN `User ID` `UserID` INT NOT NULL ,
CHANGE COLUMN `Name` `UserName` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Status` `UserStatus` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `City` `UserCity` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Country` `UserCountry` VARCHAR(100) NOT NULL ,
CHANGE COLUMN `Balance` `Balance` FLOAT NOT NULL DEFAULT 0 ,
CHANGE COLUMN `Number of Songs Played` `NumberOfSongsPlayed` INT NOT
NULL DEFAULT 0 ,
ADD PRIMARY KEY (`UserID`);
;
```

12. Alter command for Playlist

```
ALTER TABLE `musica2`.`playlist`
CHANGE COLUMN `PlaylistID` `PlaylistID` INT NOT NULL ,
CHANGE COLUMN `PlaylistName` `PlaylistName` TEXT NOT NULL ,
CHANGE COLUMN `UserID` `UserID` INT NOT NULL ,
ADD PRIMARY KEY (`PlaylistID`);
;
```

13. Alter command for PlaylistSongs

```
ALTER TABLE `musical 2`.`playlistssongs`
CHANGE COLUMN `PlaylistID` `PlaylistID` INT NOT NULL ,
CHANGE COLUMN `SongID` `SongID` INT NOT NULL ;
```

Create Table commands:

1. CREATE table command for ADVERTISEMENT

```
CREATE TABLE `advertisements` (
  `AdvertiserID` int DEFAULT NULL,
  `Name` text,
  `Number of clicks/views` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

2. CREATE table command for ADVERTISER

```
CREATE TABLE `advertisers` (
  `AdvertiserID` int DEFAULT NULL,
  `Name` text,
  `Balance` int DEFAULT NULL,
  `Total Number Of clicks` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

3. CREATE table command for ALBUM

```
CREATE TABLE `albums` (
  `AlbumID` int DEFAULT NULL,
  `Name` text,
  `Number Of Songs` int DEFAULT NULL,
  `Release Year` int DEFAULT NULL,
  `ArtistID` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

4. CREATE table command for ARTIST

```
CREATE TABLE `artists` (
  `ID` int DEFAULT NULL,
  `Name` text,
  `Gender` text,
  `MonthlyListeners` int DEFAULT NULL,
  `TotalAlbums` int DEFAULT NULL,
  `TotalSongs` int DEFAULT NULL,
  `Location` text,
  `Balance` text,
  `MyUnknownColumn` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

5. CREATE table command for EVENT

```
CREATE TABLE `event` (
  `ID` int DEFAULT NULL,
  `Name` text,
  `City` text,
  `Country` text,
  `Time` text,
  `Tickets_sold` int DEFAULT NULL,
  `Total_capacity` int DEFAULT NULL,
  `EventOrganiserID` int DEFAULT NULL,
  `ArtistID` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

6. CREATE table command for EVENTORGANISER

```
CREATE TABLE `eventorganisers` (
  `ID` int NOT NULL,
  `CompanyName` text NOT NULL,
  `Balance` int NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

7. CREATE table command for SONG

```
CREATE TABLE `songs` (
  `SID` int DEFAULT NULL,
  `Sname` text,
  `AlbumName` text,
  `Length` int DEFAULT NULL,
  `Ratings` int DEFAULT NULL,
  `Release Year` int DEFAULT NULL,
  `AlbumID` int DEFAULT NULL,
  `ArtistID` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

8. CREATE table command for TICKET

```
CREATE TABLE `tickets` (
  `UserID` int DEFAULT NULL,
  `EventID` int DEFAULT NULL,
  `Price` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

9. CREATE table command for USER

```
CREATE TABLE `users` (
  `User ID` int DEFAULT NULL,
  `Name` text,
  `Status` text,
  `City` text,
  `Country` text,
  `Balance` int DEFAULT NULL,
  `Number of Songs Played` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

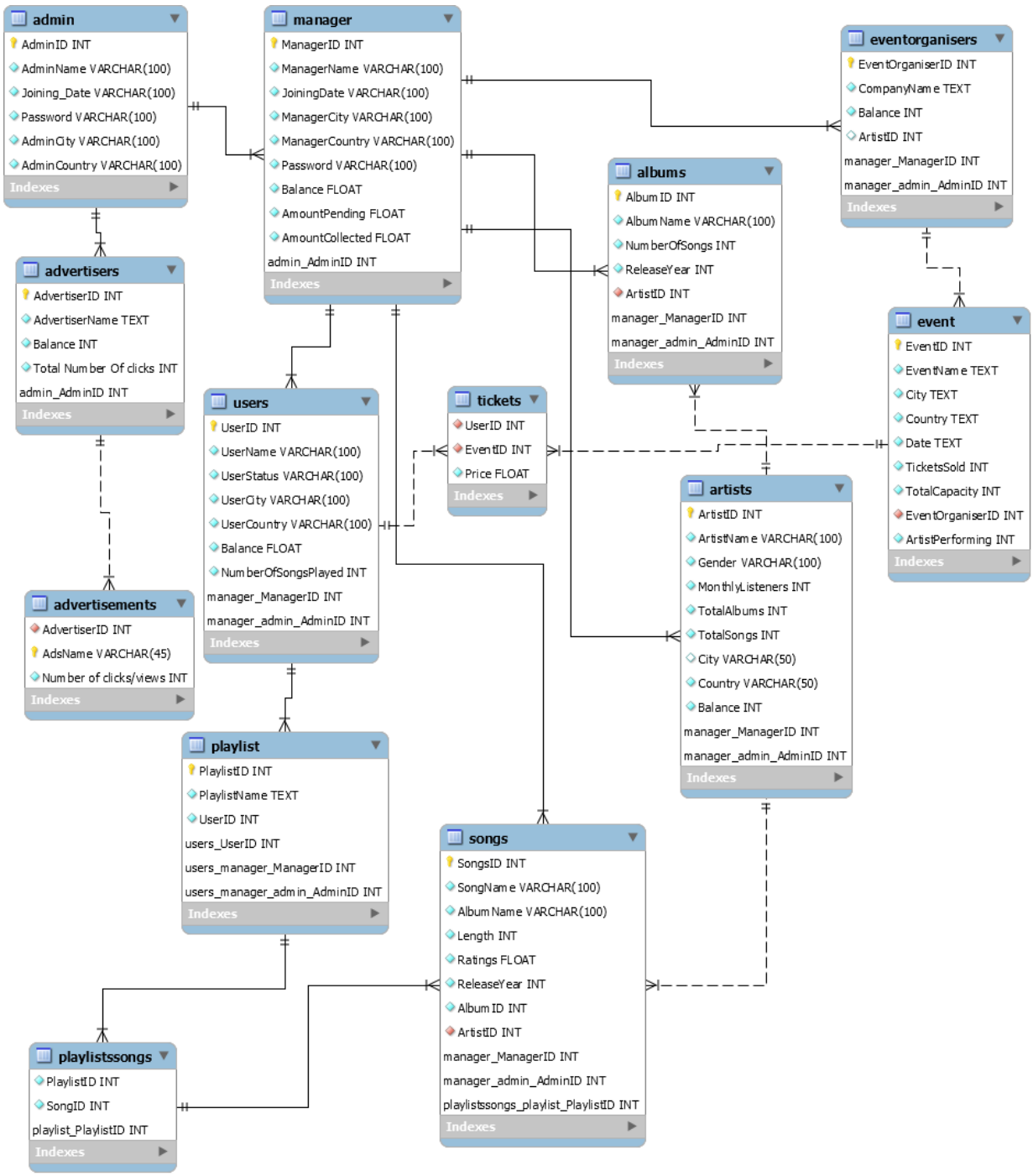
10. CREATE table command for PLAYLISTSSONGS

```
CREATE TABLE `playlistssongs` (
  `PlaylistID` int NOT NULL,
  `SongID` int NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

11. CREATE table command for PLAYLIST

```
CREATE TABLE `playlist` (
  `PlaylistID` int NOT NULL,
  `PlaylistName` text NOT NULL,
  `UserID` int NOT NULL,
  PRIMARY KEY (`PlaylistID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Relational Schema


admin
AdminID INT
AdminName VARCHAR(100)
Joining_Date VARCHAR(100)
Password VARCHAR(100)
AdminCity VARCHAR(100)
AdminCountry VARCHAR(100)
Indexes
manager
ManagerID INT
ManagerName VARCHAR(100)
JoiningDate VARCHAR(100)
ManagerCity VARCHAR(100)
ManagerCountry VARCHAR(100)
Password VARCHAR(100)
Balance FLOAT
AmountPending FLOAT
AmountCollected FLOAT
admin_AdminID INT
Indexes
eventorganisers
EventOrganiserID INT
CompanyName TEXT
Balance INT
ArtistID INT
manager_ManagerID INT
manager_admin_AdminID INT
Indexes
albums
AlbumID INT
AlbumName VARCHAR(100)
NumberOfSongs INT
ReleaseYear INT
ArtistID INT
manager_ManagerID INT
manager_admin_AdminID INT
Indexes
advertisers
AdvertiserID INT
AdvertiserName TEXT
Balance INT
Total Number Of clicks INT
admin_AdminID INT
Indexes
event
EventID INT
EventName TEXT
City TEXT
Country TEXT
Date TEXT
TicketsSold INT
TotalCapacity INT
EventOrganiserID INT
ArtistPerforming INT
Indexes
users
UserID INT
UserName VARCHAR(100)
UserStatus VARCHAR(100)
UserCity VARCHAR(100)
UserCountry VARCHAR(100)
Balance FLOAT
NumberOfSongsPlayed INT
manager_ManagerID INT
manager_admin_AdminID INT
Indexes
tickets
UserID INT
EventID INT
Price FLOAT
Indexes
artists
ArtistID INT
ArtistName VARCHAR(100)
Gender VARCHAR(100)
MonthlyListeners INT
TotalAlbums INT
TotalSongs INT
City VARCHAR(50)
Country VARCHAR(50)
Balance INT
manager_ManagerID INT
manager_admin_AdminID INT
Indexes
advertisements
AdvertiserID INT
AdsName VARCHAR(45)
Number of clicks/views INT
Indexes
playlist
PlaylistID INT
PlaylistName TEXT
UserID INT
users_UserID INT
users_manager_ManagerID INT
users_manager_admin_AdminID INT
Indexes
songs
SongsID INT
SongName VARCHAR(100)
AlbumName VARCHAR(100)
Length INT
Ratings FLOAT
ReleaseYear INT
AlbumID INT
ArtistID INT
manager_ManagerID INT
manager_admin_AdminID INT
playlistssongs_playlist_PlaylistID INT
Indexes
playlistssongs
PlaylistID INT
SongID INT
playlist_PlaylistID INT
Indexes

ALTER Table Commands while checking INTEGRITY CONSTRAINTS and relationship schema

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';


ALTER TABLE `musica2`.`playlist`
ADD COLUMN `users_UserID` INT(11) NOT NULL AFTER `UserID`,
ADD COLUMN `users_manager_ManagerID` INT(11) NOT NULL AFTER
`users_UserID`,
ADD COLUMN `users_manager_admin_AdminID` INT(11) NOT NULL AFTER
`users_manager_ManagerID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`PlaylistID`, `users_UserID`, `users_manager_ManagerID`,
`users_manager_admin_AdminID`),
ADD INDEX `fk_playlist_users1_idx` (`users_UserID` ASC,
`users_manager_ManagerID` ASC, `users_manager_admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`playlistssongs`
ADD COLUMN `playlist_PlaylistID` INT(11) NOT NULL AFTER `SongID`,
ADD PRIMARY KEY (`playlist_PlaylistID`);
;

ALTER TABLE `musica2`.`songs`
ADD COLUMN `playlistssongs_playlist_PlaylistID` INT(11) NOT NULL AFTER
`manager_admin_AdminID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`SongsID`, `manager_ManagerID`, `manager_admin_AdminID`,
`playlistssongs_playlist_PlaylistID`),
ADD INDEX `fk_songs_playlistssongs1_idx` (`playlistssongs_playlist_PlaylistID` ASC)
VISIBLE;
;

ALTER TABLE `musica2`.`playlist`
ADD CONSTRAINT `fk_playlist_users1`
  FOREIGN KEY (`users_UserID` , `users_manager_ManagerID` ,
`users_manager_admin_AdminID`)
```

```
  REFERENCES `musica2`.`users` (`UserID` , `manager_ManagerID` ,
`manager_admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`playlistssongs`
ADD CONSTRAINT `fk_playlistssongs_playlist1`
  FOREIGN KEY (`playlist_PlaylistID`)
  REFERENCES `musica2`.`playlist` (`PlaylistID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`songs`
ADD CONSTRAINT `fk_songs_playlistssongs1`
  FOREIGN KEY (`playlistssongs_playlist_PlaylistID`)
  REFERENCES `musica2`.`playlistssongs` (`playlist_PlaylistID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;


SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

ALTER TABLE `musica2`.`advertisers`
ADD COLUMN `admin_AdminID` INT(11) NOT NULL AFTER `Total Number Of clicks`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`AdvertiserID`, `admin_AdminID`),
ADD INDEX `fk_advertisers_admin1_idx` (`admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`albums`
ADD COLUMN `manager_ManagerID` INT(11) NOT NULL AFTER `ArtistID`,
ADD COLUMN `manager_admin_AdminID` INT(11) NOT NULL AFTER
`manager_ManagerID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`AlbumID`, `manager_ManagerID`, `manager_admin_AdminID`),
ADD INDEX `fk_albums_manager1_idx` (`manager_ManagerID` ASC,
`manager_admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`artists`
ADD COLUMN `manager_ManagerID` INT(11) NOT NULL AFTER `Balance`,
```

```
ADD COLUMN `manager_admin_AdminID` INT(11) NOT NULL AFTER
`manager_ManagerID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`ArtistID`, `manager_ManagerID`, `manager_admin_AdminID`),
ADD INDEX `fk_artists_manager1_idx` (`manager_ManagerID` ASC,
`manager_admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`eventorganisers`
ADD COLUMN `manager_ManagerID` INT(11) NOT NULL AFTER `ArtistID`,
ADD COLUMN `manager_admin_AdminID` INT(11) NOT NULL AFTER
`manager_ManagerID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`EventOrganiserID`, `manager_ManagerID`,
`manager_admin_AdminID`),
ADD INDEX `fk_eventorganisers_manager1_idx` (`manager_ManagerID` ASC,
`manager_admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`manager`
ADD COLUMN `admin_AdminID` INT(11) NOT NULL AFTER `AmountCollected`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`ManagerID`, `admin_AdminID`),
ADD INDEX `fk_manager_admin1_idx` (`admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`songs`
ADD COLUMN `manager_ManagerID` INT(11) NOT NULL AFTER `ArtistID`,
ADD COLUMN `manager_admin_AdminID` INT(11) NOT NULL AFTER
`manager_ManagerID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`SongsID`, `manager_ManagerID`, `manager_admin_AdminID`),
ADD INDEX `fk_songs_manager1_idx` (`manager_ManagerID` ASC,
`manager_admin_AdminID` ASC) VISIBLE,
ADD INDEX `fk_songs_artists1_idx` (`ArtistID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`tickets`
ADD INDEX `fk_tickets_users1_idx` (`UserID` ASC) VISIBLE,
ADD INDEX `fk_tickets_event1_idx` (`EventID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`users`
ADD COLUMN `manager_ManagerID` INT(11) NOT NULL AFTER
`NumberOfSongsPlayed`,
```

```
ADD COLUMN `manager_admin_AdminID` INT(11) NOT NULL AFTER
`manager_ManagerID`,
DROP PRIMARY KEY,
ADD PRIMARY KEY (`UserID`, `manager_ManagerID`, `manager_admin_AdminID`),
ADD INDEX `fk_users_manager1_idx` (`manager_ManagerID` ASC,
`manager_admin_AdminID` ASC) VISIBLE;
;

ALTER TABLE `musica2`.`advertisers`
ADD CONSTRAINT `fk_advertisers_admin1`
  FOREIGN KEY (`admin_AdminID`)
  REFERENCES `musica2`.`admin` (`AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`albums`
ADD CONSTRAINT `fk_albums_manager1`
  FOREIGN KEY (`manager_ManagerID` , `manager_admin_AdminID`)
  REFERENCES `musica2`.`manager` (`ManagerID` , `admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`artists`
ADD CONSTRAINT `fk_artists_manager1`
  FOREIGN KEY (`manager_ManagerID` , `manager_admin_AdminID`)
  REFERENCES `musica2`.`manager` (`ManagerID` , `admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`eventorganisers`
ADD CONSTRAINT `fk_eventorganisers_manager1`
  FOREIGN KEY (`manager_ManagerID` , `manager_admin_AdminID`)
  REFERENCES `musica2`.`manager` (`ManagerID` , `admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`manager`
ADD CONSTRAINT `fk_manager_admin1`
  FOREIGN KEY (`admin_AdminID`)
  REFERENCES `musica2`.`admin` (`AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`songs`
ADD CONSTRAINT `fk_songs_manager1`
  FOREIGN KEY (`manager_ManagerID` , `manager_admin_AdminID`)
```

```
  REFERENCES `musica2`.`manager` (`ManagerID` , `admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
ADD CONSTRAINT `fk_songs_artists1`
  FOREIGN KEY (`ArtistID`)
  REFERENCES `musica2`.`artists` (`ArtistID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`tickets`
ADD CONSTRAINT `fk_tickets_users1`
  FOREIGN KEY (`UserID`)
  REFERENCES `musica2`.`users` (`UserID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
ADD CONSTRAINT `fk_tickets_event1`
  FOREIGN KEY (`EventID`)
  REFERENCES `musica2`.`event` (`EventID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`users`
ADD CONSTRAINT `fk_users_manager1`
  FOREIGN KEY (`manager_ManagerID` , `manager_admin_AdminID`)
  REFERENCES `musica2`.`manager` (`ManagerID` , `admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- MySQL Workbench Synchronization
-- Generated: 2021-02-22 14:27
-- Model: New Model
-- Version: 1.0
-- Project: Name of the project
-- Author: ashita

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
```

```
CREATE TABLE IF NOT EXISTS `musica2`.`playlist_has_songs` (
  `playlist_PlaylistID` INT(11) NOT NULL,
  `songs_SongsID` INT(11) NOT NULL,
  `songs_manager_ManagerID` INT(11) NOT NULL,
  `songs_manager_admin_AdminID` INT(11) NOT NULL,
  PRIMARY KEY (`playlist_PlaylistID`, `songs_SongsID`, `songs_manager_ManagerID`,
`songs_manager_admin_AdminID`),
  INDEX `fk_playlist_has_songs_songs1_idx` (`songs_SongsID` ASC,
`songs_manager_ManagerID` ASC, `songs_manager_admin_AdminID` ASC) VISIBLE,
  INDEX `fk_playlist_has_songs_playlist1_idx` (`playlist_PlaylistID` ASC) VISIBLE,
  CONSTRAINT `fk_playlist_has_songs_playlist1`
    FOREIGN KEY (`playlist_PlaylistID`)
    REFERENCES `musica2`.`playlist` (`PlaylistID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_playlist_has_songs_songs1`
    FOREIGN KEY (`songs_SongsID` , `songs_manager_ManagerID` ,
`songs_manager_admin_AdminID`)
    REFERENCES `musica2`.`songs` (`SongsID` , `manager_ManagerID` ,
`manager_admin_AdminID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

DROP TABLE IF EXISTS `musica2`.`playlistssongs` ;

ALTER TABLE `musica2`.`playlist`
ADD CONSTRAINT `fk_playlist_users1`
  FOREIGN KEY (`UserID`)
  REFERENCES `musica2`.`users` (`UserID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- MySQL Workbench Synchronization
-- Generated: 2021-02-22 14:33
-- Model: New Model
-- Version: 1.0
-- Project: Name of the project
```

```
-- Author: ashita

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

ALTER TABLE `musica2`.`playlist`
DROP FOREIGN KEY `fk_playlist_users1`;

ALTER TABLE `musica2`.`playlist_has_songs`
DROP FOREIGN KEY `fk_playlist_has_songs_playlist1`,
DROP FOREIGN KEY `fk_playlist_has_songs_songs1`;

ALTER TABLE `musica2`.`playlist`
ADD CONSTRAINT `fk_playlist_users1`
  FOREIGN KEY (`UserID`)
  REFERENCES `musica2`.`users` (`UserID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;

ALTER TABLE `musica2`.`playlist_has_songs`
ADD CONSTRAINT `fk_playlist_has_songs_playlist1`
  FOREIGN KEY (`playlist_PlaylistID`)
  REFERENCES `musica2`.`playlist` (`PlaylistID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
ADD CONSTRAINT `fk_playlist_has_songs_songs1`
  FOREIGN KEY (`songs_SongsID` , `songs_manager_ManagerID` ,
`songs_manager_admin_AdminID`)
  REFERENCES `musica2`.`songs` (`SongsID` , `manager_ManagerID` ,
`manager_admin_AdminID`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

VIEWS

1. Advertiser
    a. Join on advertiserID on tables advertisers and advertisements
2. Event Organiser
    a. Join on event and eventOrganizer eventOrganizerID common
3. Artist
    a. Join on tables event,artist,albums,songs artistID common
    b. // Cities need to be added in artists table as some values are null and it will affect our queries.
4. User
    a. // Non-premium users had also created playlist and it needs to be corrected as only premium are allowed to make playlist
    b. Status not required in premium_user and non_premium user tables.
    c. Does user know which ad he has seen?
    d. Shown table missing
    e. User likes song How are we going to maintain this?or shall we remove this feature?
    f. Shall artist and event not have n-n relationship or else an artist cannot perform at multiple events.
    g. Join Ticket playlist ads(to ask)
5. Manager
    a. He is the admin and can view all the above views.

ADD A TABLE FOR THE LIKED SONGS, WITH THE CORRESPONDING USERIDs, and SONGID.
ADD ANOTHER TABLE FOR FOLLOWED_ARTISTS WITH CORRESPONDING ARTIST ID AND USER ID.

Queries
Users

1. Search by name, album, artist, year, genre
    a. Name - select * from `dbms-musica`.songs where Sname="song name";
    b. Album - select * from `dbms-musica`.songs where AlbumName="album";
    c. Artist - select S.SID, S.Sname, S.AlbumName, S.Length, S.Ratings, S.Release_Year, A.Name from `dbms-musica`.artists as A inner join `dbms-musica`.songs as S on A.ArtistID=S.ArtistID where A.Name='artist name';
    d. Year(before) - select * from `dbms-musica`.songs where year(Release_Year)<year_inut;

    e. Year(after) - select * from `dbms-musica`.songs where year(Release_Year)>year_input;
    f. Genre - "
2. Upgrade to premium (changing the status of the user)
3. Create a playlist (add an entry to playlist table)
4. Edit(Add/ remove a song) playlist (playlistssongs table to be added/removed corresponding to the songID)
    a. Can like/unlike a song (add an entry to the table, "liked playlist")
5. Follow/unfollow an artist (update the artist_follower table)
6. Pay for concert tickets (add an entry in the table ticket, corresponding to the userID)
7. List all upcoming and current event(concerts)
        'SELECT * FROM `dbms-musica`.event;'
8. List all booked events for a user
        SELECT e.Name, e.City, e.Country, e.Time, ar.Name, t.Price FROM `dbms-musica`.event as e inner join `dbms-musica`.tickets as t on e.EventID=t.EventID inner join `dbms-musica`.artists as ar on e.ArtistID=ar.ArtistID where t.UserID=input_userid;

Artist

1. Create,view (likes, searches,streams,statistics(monthly,yearly or weekly)), a song
    a. Create -
    b. View -
2. Edit(delete/update) details of their songs/albums
3. Bookings done for concerts
        select Name,City,Country,Time,Tickets_sold,Total_capacity,ArtistID from `dbms-musica`.event where ArtistID=artistid;
4. View all their previous, upcoming concerts
        SELECT * FROM `dbms-musica`.event where ArtistID=artistid;
5. Check amount in their balance
        SELECT `Name`,`Balance` FROM `dbms-musica`.artists where ArtistID=artistid;
6. List all scheduled events where he/she needs to perform
        SELECT * FROM `dbms-musica`.event where ArtistID=artistid and Time>curdate();

Admin

1. Enables advertisements-feature
2. Analysis of the concerts and views
3. Manage advertisements

Event/concert organisers

1. Create,view a new event

a. Create - insert into `dbms-musica`.event(`Name`,`City`,`Country`,`Time`,`Total_capacity`,`EventOrganiserID`) values ('new event name','city name','country name','YYYY-MM-DD',capacity,organiserID);
b. View - SELECT * FROM `dbms-musica`.event where EventOrganiserID=organiser id;

2. Edit an event(update trailers,songs etc/remove it)
    a. Name - update `dbms-musica`.event set `Name`='new name' where EventOrganiserID=organiserid and EventID=eventid;
    b. City, Country - update `dbms-musica`.event set `City`='new city', `Country`='new country' where EventOrganiserID=organiserid and EventID=eventid;
    c. Time - update `dbms-musica`.event set `Time`='YYYY-MM-DD' where EventOrganiserID=organiserid and EventID=eventid;
    d. Total_capacity - update `dbms-musica`.event set `Total_capacity`=new_cap where EventOrganiserID=organiserid and EventID=eventid;
3. Manage booking (open / close the registrations)
4. Select artists for concerts
    update `dbms-musica`.event set `ArtistID`=artistid_input where EventOrganiserID = organiser id and EventID=eventid;
5. Check amount in their balance
    select CompanyName,Balance from `dbms-musica`.eventorganizers where EventOrganizerID=organiser id;
6. List all upcoming and current event(concerts)
    SELECT * FROM `dbms-musica`.event where EventOrganiserID=organiser id;
7. View tickets sold for events
    select Name,City,Country,Time,Tickets_sold,Total_capacity,ArtistID from `dbms-musica`.event where EventOrganiserID=organiserid;

Advertisers

1. Create/view/edit their ads
    a. Create - insert into `dbms-musica`.advertisements(`AdvertiserID`,`AdsName`,`Number_of_clicks/views`) values (advertiser id,"new ad name",0);
    b. View - select * from `dbms-musica`.advertisements where AdvertiserID=advertiser id;
    c. Edit - update `dbms-musica`.advertisements set `AdsName` = "update name" where AdvertiserID=advertiser id and AdsName=old ad name';

2. View number of times seen or clicks on ad
    a. All ads - SELECT AdsName,`Number_of_clicks/views` FROM `dbms-musica`.advertisements where AdvertiserID=advertiser id;

    b. One ad - SELECT AdsName,`Number_of_clicks/views` FROM `dbms-musica`.advertisements where AdvertiserID=advertiser id and AdsName='ad name';
3. Pay money to manager according to viewership

Managers

1. Create and suggest playlists to a set of users based on their likes/view history
2. Premium members -features
3. Collect money from advertisers
4. Collect money from users for concerts
5. Pay money to event organizers and artist
6. List all premium members till a particular date

Dear All,
There is an updation for end-sem project document submission:
Demonstrating your system through a UI for data access and manipulation is **optional.** (You may include)

Normalization of the table is **optional.** (You may include)

MUST part:

* Write at least 10 queries in SQL involving various operations supporting the application features involving database access and manipulation.
***** Identify the attribute(s) to create Index tables required for your queries.
***** Write at least 4 embedded SQL queries (PL/SQL), advanced aggregation functions, etc supporting your application features. Here you can use any programming language (like Python, Java, C++, etc.) for connecting the database and perform 4 SQL queries on the database.