DBMS PROJECT REPORT

Team members : Arka Sarkar, Dhruv Yadav, Nikhil Dahiya, Amit Maurya, Harman

Team 57: **Quinennials**

presents:

BAJAO!

Our logo

App Icon

Application

The aim of this project is the development of **mobile music streaming and event ticket booking application**. The application has an **online music store** where our users can choose what songs to listen to and also create **online** playlists. Our audience can buy a **premium subscription** for an **ad-free experience**. The application also enables the user to **search and book concert tickets**.
Musical professionals would upload their songs to the database for our audience to listen to.
Advertisers can post Ads for their products. **Event Organisers** can post **events and live concerts**, and **tickets for live concerts** would be sold by **ticketing services** on our application.
All the transactions would be handled by the **banking services** affiliated with our company.

Stakeholders

- **Musicians** - Willing to upload their songs to a wider audience
  - Singers
  - Bands
  - Indie artists
  - DJs
  - Instrumentalists
- **Audience/ General Public** - Willing to listen to songs in their leisure time
- **Music Event Handlers/ Organizers** - Willing to publicize their event and sell tickets for the same
- **Event Ticket Buyers** - Willing to buy tickets for the said music events
- **Bankers** - Willing to gain profit by the utilization of their services in our app for selling tickets
- **Advertisers** - Willing to advertise their product on our app

Database Schema

TABLES	FIELDS	DATA TYPES	KEYS
USER	USER_ID NAME GENDER ICON COUNTRY EMAIL DOB	int varchar(50) varchar(50) mediumblob varchar(20) varchar(20) date	PRIMARY KEY - USER_ID
USER_AUTH	USER_ID PASSWORD	int varchar(20)	FOREIGN KEY - USER_ID (references USER)
USER_PREF_GENRE	USER_ID GENRE	int varchar(30)	FOREIGN KEY - USER_ID (references USER)
SUBSCRIPTION	USER_ID SUB_TYPE SUB_START SUB_END TRANSACTION_ID	int varchar(20) datetime datetime int	FOREIGN KEY - USER_ID (references USER), TRANSACTION_ID (references TRANSACTIONS),
ALL_SONGS	SONG_ID TITLE LANGUAGE ARTIST_ID ALBUM_ID LIKES COVER RELEASE_DATE	int varchar(50) varchar(30) int int int mediumblob date	PRIMARY KEY - SONG_ID FOREIGN KEY - ARTIST_ID (references ALL_ARTISTS), ALBUM_ID (references ALBUMS)
SONG_GENRE	SONG_ID GENRE	int varchar(30)	FOREIGN KEY - SONG_ID (references ALL_SONGS)
SONG_DATA	SONG_ID SONG_BITDATA	int largeblob	FOREIGN KEY - SONG_ID (references ALL_SONGS)
ALL_ARTISTS	ARTIST_ID NAME SONG_NUM POPULARITY	int varchar(50) int int	PRIMARY KEY - ARTIST_ID
ARTIST_GENRE	ARTIST_ID GENRE	int varchar(30)	FOREIGN KEY - ARTIST_ID (references ALL_ARTISTS)
ALBUMS	ALBUM_ID NAME ARTIST_ID LABEL RELEASE_DATE	int varchar(50) int varchar(50) date	PRIMARY KEY - ALBUM_ID FOREIGN KEY - ARTIST_ID (references ALL_ARTISTS)

ALBUM_GENRE	ALBUM_ID GENRE	int varchar(30)	FOREIGN KEY - ALBUM_ID (references ALBUMS)
USER_PLAYLISTS	PLAYLIST_ID USER_ID NAME SONG_NUM	int int varchar(30) int	PRIMARY KEY - PLAYLIST_ID FOREIGN KEY - USER_ID (references USER)
PLAYLIST_SONGS	PLAYLIST_ID SONG_ID	int int	FOREIGN KEY - PLAYLIST_ID (references USER_PLAYLISTS), SONG_ID (references ALL_SONGS)
ADVERTISERS	ADVERTISER_ID NAME EMAIL ADDRESS CONTACT_NUM	int varchar(50) varchar(30) varchar(100) varchar(12)	PRIMARY KEY - ADVERTISER_ID
ADVERTISEMENTS	AD_ID ADVERTISER_ID	int int	PRIMARY KEY - AD_ID FOREIGN KEY - ADVERTISER_ID (references ADVERTISERS)
LIVE_PERFOMANCES	LIVE_PERF_ID ARTIST_ID SPONSORS ADVERTISER_ID LIVE_RATINGS TICKETS_NUM DETAILS	int int varchar(50) int float(10,2) int mediumtext	PRIMARY KEY - LIVE_PERF_ID FOREIGN KEY - ARTIST_ID (references ALL_ARTISTS), ADVERTISER_ID (references ADVERTISER)
SCHEDULE	SCHEDULE_ID LIVE_PERF_ID VENUE DATE_TIME	int int varchar(50) datetime	PRIMARY KEY - SCHEDULE_ID FOREIGN KEY - LIVE_PERF_ID (references LIVE_PERFORMANCES)
BOOKINGS	BOOKING_ID TICKET_ID USER_ID TRANSACTION_ID STATUS	int int int int varchar(50)	PRIMARY KEY - BOOKING_ID FOREIGN KEY - USER_ID (references USER), TRANSACTION_ID (references TRANSACTIONS), TICKET_ID (references TICKETS)
TRANSACTIONS	TRANSACTION_ID PAYMENT_TYPE BANK_ID PAYMENT_AMOUNT DATE_TIME CURRENCY	int varchar(50) float(10,2) int datetime varchar(50)	PRIMARY KEY - TRANSACTION_ID FOREIGN KEY - BANK_ID (references BANKS)
TICKETS	TICKET_ID LIVE_PERF_ID TIC_SERV_ID DATE_TIME AMOUNT	int int int datetime float(10,2)	PRIMARY KEY - TICKET_ID FOREIGN KEY - LIVE_PERF_ID (references LIVE_PERFORMANCE), TIC_SERV_ID (references TICKETING_SERVICES)

TICKETING_SERVICE	TIC_SERV_ID NAME EMAIL CONTACT_NUM	int varchar(50) varchar(50) varchar(12)	PRIMARY KEY - TIC_SERV_ID
BANKS	BANK_ID NAME EMAIL ADDRESS CONTACT_NUM BANK_AMOUNT	int varchar(50) varchar(50) varchar(50) varchar(12) float	PRIMARY KEY - BANK_ID
USER_WALLET	USER_ID AMOUNT	int float	FOREIGN KEY - USER_ID (references USER)

Key for Schema

- BLUE - Tables
- GREEN - NOT_NULL
- ORANGE - PRIMARY KEYS
- BROWN - FOREIGN KEYS

Database Creation Queries

```
CREATE TABLE `ADVERTISEMENTS` (
  `ad_id` int(11) NOT NULL,
  `advertiser_id` int(11) NOT NULL,
  PRIMARY KEY (`ad_id`),
  KEY `advertiser-ad_idx` (`advertiser_id`),
  CONSTRAINT `advertiser-ad` FOREIGN KEY
(`advertiser_id`) REFERENCES `ADVERTISERS`
(`advertiser_id`)
)
```

```
CREATE TABLE `ADVERTISERS` (
  `advertiser_id` int(11) NOT NULL,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `address` varchar(45),
  `contact` varchar(45),
  PRIMARY KEY (`advertiser_id`)
)
```

```
CREATE TABLE `ALBUMS` (
  `album_id` int(11) NOT NULL,
  `name` varchar(45) NOT NULL,
  `artist_id` int(11) NOT NULL,
  `label` varchar(45) NOT NULL,
  `release_date` date NOT NULL,
  PRIMARY KEY (`album_id`),
  KEY `album-artist_idx` (`artist_id`),
  CONSTRAINT `album-artist` FOREIGN KEY (`artist_id`)
REFERENCES `ALL_ARTISTS` (`artist_id`)
)
```

```
CREATE TABLE `ALBUM_GENRE` (
  `album_id` int(11) NOT NULL,
  `genre` varchar(45) NOT NULL,
  KEY `album-genre_idx` (`album_id`),
  CONSTRAINT `album-genre` FOREIGN KEY (`album_id`)
REFERENCES `ALBUMS` (`album_id`)
)
```

```
CREATE TABLE `ALL_ARTISTS` (
  `artist_id` int(11) NOT NULL,
  `name` varchar(45) NOT NULL,
  `song_num` int(11) NOT NULL,
  `popularity` int(11) DEFAULT NULL,
  PRIMARY KEY (`artist_id`)
)
```

```
CREATE TABLE `ALL_SONGS` (
  `song_id` int(11) NOT NULL,
  `title` varchar(45) NOT NULL,
  `language` varchar(45) NOT NULL,
  `artist_id` int(11) NOT NULL,
  `album_id` int(11) DEFAULT NULL,
  `likes` int(11) DEFAULT NULL,
  `duration` time NOT NULL,
  `release_date` date NOT NULL,
  PRIMARY KEY (`song_id`),
  KEY `song-artist_idx` (`artist_id`),
  KEY `song-album_idx` (`album_id`),
  CONSTRAINT `song-album` FOREIGN KEY (`album_id`)
REFERENCES `ALBUMS` (`album_id`),
  CONSTRAINT `song-artist` FOREIGN KEY (`artist_id`)
REFERENCES `ALL_ARTISTS` (`artist_id`)
)
```

```
CREATE TABLE `ARTIST_GENRE` (
  `artist_id` int(11) NOT NULL,
  `genre` varchar(45) NOT NULL,
  KEY `artist_genre_idx` (`artist_id`),
  CONSTRAINT `artist_genre` FOREIGN KEY (`artist_id`)
REFERENCES `ALL_ARTISTS` (`artist_id`)
)
```

```
CREATE TABLE `BANKS` (
  `bank_id` int(11) NOT NULL,
  `name` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `address` varchar(45),
  `contact` varchar(45),
  `bank_amount` float NOT NULL,
  PRIMARY KEY (`bank_id`)
)
```

```
CREATE TABLE `SCHEDULE` (
  `schedule_id` int(11) NOT NULL,
  `live_perf_id` int(11) NOT NULL,
  `venue` varchar(45) NOT NULL,
  `date_time` datetime NOT NULL,
  PRIMARY KEY (`schedule_id`),
  KEY `lp-schedule_idx` (`live_perf_id`),
  CONSTRAINT `lp-schedule` FOREIGN KEY (`live_perf_id`)
REFERENCES `LIVE_PERFORMANCES` (`live_perf_id`)
)
```

```
CREATE TABLE `BOOKINGS` (
  `booking_id` int(11) NOT NULL,
  `ticket_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `transaction_id` int(11) NOT NULL,
  `status` varchar(45) NOT NULL,
  PRIMARY KEY (`booking_id`),
  KEY `booking-user_idx` (`user_id`),
  KEY `book-trans_idx` (`transaction_id`),
  KEY `book-ticket_idx` (`ticket_id`),
  CONSTRAINT `book-ticket` FOREIGN KEY (`ticket_id`)
REFERENCES `TICKETS` (`ticket_id`),
  CONSTRAINT `book-trans` FOREIGN KEY
(`transaction_id`) REFERENCES `TRANSACTIONS`
(`transaction_id`),
  CONSTRAINT `booking-user` FOREIGN KEY (`user_id`)
REFERENCES `USER` (`user_id`)
)
```

```
CREATE TABLE `LIVE_PERFORMANCES` (
  `live_perf_id` int(11) NOT NULL,
  `artist_id` int(11) NOT NULL,
  `sponsors` varchar(45) NOT NULL,
  `advertiser_id` int(11) NOT NULL,
  PRIMARY KEY (`live_perf_id`),
  KEY `lp-artist_idx` (`artist_id`),
  KEY `lp-advertiser_idx` (`advertiser_id`),
  CONSTRAINT `lp-advertiser` FOREIGN KEY
(`advertiser_id`) REFERENCES `ADVERTISERS`
(`advertiser_id`),
  CONSTRAINT `lp-artist` FOREIGN KEY (`artist_id`)
REFERENCES `ALL_ARTISTS` (`artist_id`)
)
```

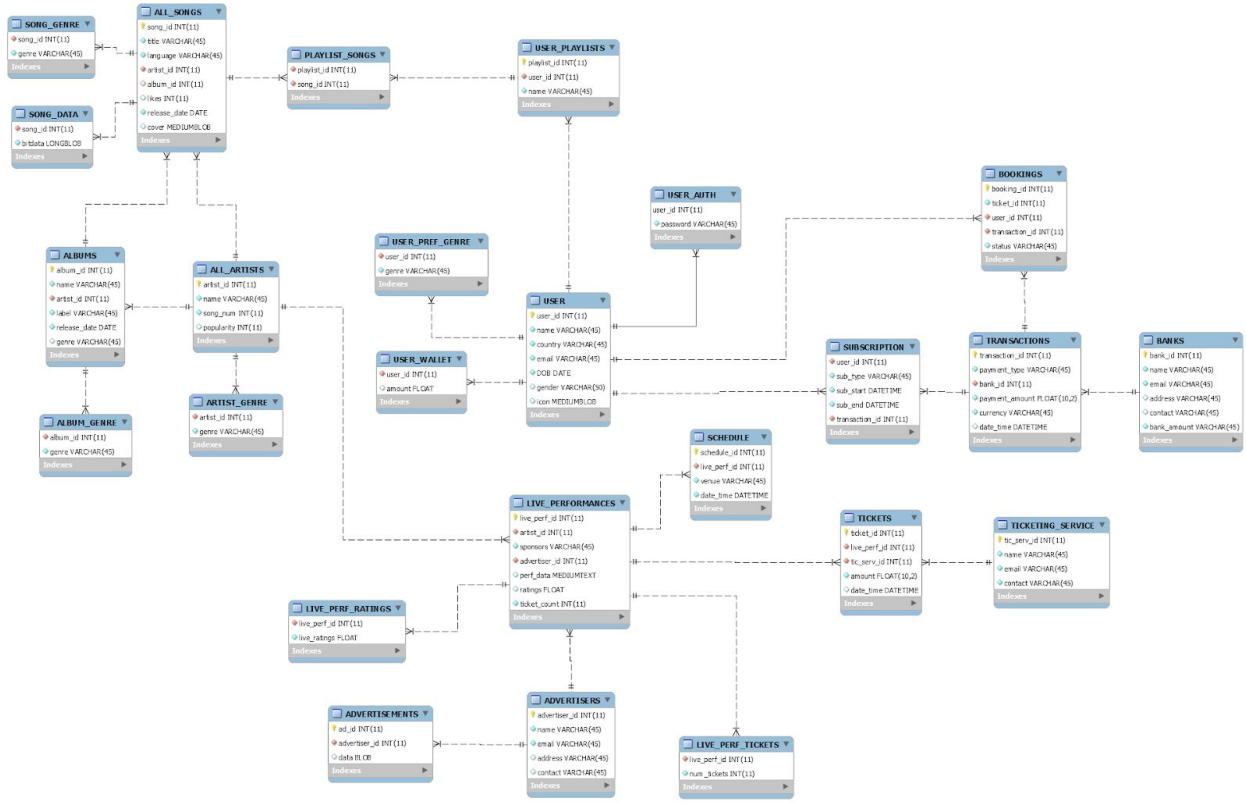
```
CREATE TABLE `LIVE_PERF_RATINGS` (
  `live_perf_id` int(11) NOT NULL,
  `live_ratings` float NOT NULL,
  KEY `lp-rating_idx` (`live_perf_id`),
  CONSTRAINT `lp-rating` FOREIGN KEY (`live_perf_id`)
REFERENCES `LIVE_PERFORMANCES` (`live_perf_id`)
)
```

```
CREATE TABLE `LIVE_PERF_TICKETS` (
  `live_perf_id` int(11) NOT NULL,
  `num_tickets` int(11) NOT NULL,
  KEY `lp-tickets_idx` (`live_perf_id`),
  CONSTRAINT `lp-tickets` FOREIGN KEY (`live_perf_id`)
REFERENCES `LIVE_PERFORMANCES` (`live_perf_id`)
)
```

```
CREATE TABLE `PLAYLIST_SONGS` (
  `plsylist_id` int(11) NOT NULL,
  `song_id` int(11) NOT NULL,
  KEY `playlist-song_idx` (`song_id`),
  KEY `playlist-user_idx` (`plsylist_id`),
  CONSTRAINT `playlist-song` FOREIGN KEY (`song_id`)
REFERENCES `ALL_SONGS` (`song_id`),
  CONSTRAINT `playlist-user` FOREIGN KEY (`plsylist_id`)
REFERENCES `USER_PLAYLISTS` (`playlist_id`)
)
```

```
CREATE TABLE `SONG_DATA` (
  `song_id` int(11) NOT NULL,
  `bitdata` mediumblob NOT NULL,
  KEY `song_id_idx` (`song_id`),
  CONSTRAINT `song_id` FOREIGN KEY (`song_id`)
REFERENCES `ALL_SONGS` (`song_id`)
)


CREATE TABLE `SONG_GENRE` (
  `song_id` int(11) NOT NULL,
  `genre` varchar(45) CHARACTER SET utf8 COLLATE
utf8_unicode_ci NOT NULL,
  KEY `song-genre_idx` (`song_id`),
  KEY `genre` (`genre`),
  CONSTRAINT `song-genre` FOREIGN KEY (`song_id`)
REFERENCES `ALL_SONGS` (`song_id`)
)


CREATE TABLE `SUBSCRIPTION` (
  `user_id` int(11) NOT NULL,
  `sub_type` varchar(45) NOT NULL,
  `sub_start` datetime NOT NULL,
  `sub_end` datetime NOT NULL,
  `transaction_id` int(11) NOT NULL,
  KEY `sub-user_idx` (`user_id`),
  KEY `sub-trans_idx` (`transaction_id`),
  CONSTRAINT `sub-trans` FOREIGN KEY (`transaction_id`)
REFERENCES `TRANSACTIONS` (`transaction_id`),
  CONSTRAINT `sub-user` FOREIGN KEY (`user_id`)
REFERENCES `USER` (`user_id`)
)


CREATE TABLE `TICKETING_SERVICE` (
  `tic_serv_id` int(11) NOT NULL,
  `name` varchar(45) CHARACTER SET utf8 COLLATE
utf8_unicode_ci NOT NULL,
  `email` varchar(45) CHARACTER SET utf8 COLLATE
utf8_unicode_ci NOT NULL,
  `contact` varchar(45) CHARACTER SET utf8 COLLATE
utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`tic_serv_id`)
)


CREATE TABLE `TICKETS` (
  `ticket_id` int(11) NOT NULL,
  `live_perf_id` int(11) NOT NULL,
  `tic_serv_id` int(11) NOT NULL,
  `amount` float(10,2) NOT NULL,
  `date_time` datetime DEFAULT NULL,
  PRIMARY KEY (`ticket_id`),
  KEY `tickert-lp_idx` (`live_perf_id`),
  KEY `ticket-ticserv_idx` (`tic_serv_id`),
  CONSTRAINT `tickert-lp` FOREIGN KEY (`live_perf_id`)
REFERENCES `LIVE_PERFORMANCES` (`live_perf_id`),
  CONSTRAINT `ticket-ticserv` FOREIGN KEY (`tic_serv_id`)
REFERENCES `TICKETING_SERVICE` (`tic_serv_id`)
)


CREATE TABLE `TRANSACTIONS` (
  `transaction_id` int(11) NOT NULL,
  `payment_type` varchar(45) NOT NULL,
  `bank_id` int(11) NOT NULL,
  `payment_amount` float(10,2) NOT NULL,
  `currency` varchar(45) NOT NULL,
  `date_time` datetime DEFAULT NULL,
  PRIMARY KEY (`transaction_id`),
  KEY `trans-bank_idx` (`bank_id`),
  CONSTRAINT `trans-bank` FOREIGN KEY (`bank_id`)
REFERENCES `BANKS` (`bank_id`)
)


CREATE TABLE `USER` (
  `user_id` int(11) NOT NULL,
  `name` varchar(45) NOT NULL,
  `country` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `DOB` date NOT NULL,
  `gender` varchar(50),
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `email` (`email`)
)


CREATE TABLE `USER_AUTH` (
  `user_id` int(11) NOT NULL,
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`user_id`),
  KEY `user_id_idx` (`user_id`),
  CONSTRAINT `user_id` FOREIGN KEY (`user_id`)
REFERENCES `USER` (`user_id`)
)


CREATE TABLE `USER_PLAYLISTS` (
  `playlist_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `name` varchar(45) NOT NULL,
  `song_num` int(11) NOT NULL,
  PRIMARY KEY (`playlist_id`),
  KEY `user-playlsit_idx` (`user_id`),
  CONSTRAINT `user-playlsit` FOREIGN KEY (`user_id`)
REFERENCES `USER` (`user_id`)
)

CREATE TABLE `USER_PREF_GENRE` (
  `user_id` int(11) NOT NULL,
  `genre` varchar(45) NOT NULL,
  KEY `user_id2_idx` (`user_id`),
  CONSTRAINT `user_id2` FOREIGN KEY (`user_id`)
REFERENCES `USER` (`user_id`)
)


CREATE TABLE `USER_WALLET` (
  `user_id` int(11) NOT NULL,
  `amount` float DEFAULT NULL,
  KEY `user_id_idx` (`user_id`),
  CONSTRAINT `` FOREIGN KEY (`user_id`) REFERENCES
`USER` (`user_id`)
)
```

EER Model



Link to Higher Quality PDF file: https://drive.google.com/open?id=1ppBMau1Sy_jBm2mse0TMgUWtaWgXAQup

Stakeholder Queries

- **Musicians - Bands, DJs, Singers, producers, ...**
  - Check their current popularity.
  - Find the number of views on their latest song.
  - Check out which are their most popular songs.
  - Compare their statistics with artists of a similar genre.
  - Release their new song.
  - Remove their old album from 2003.

- **Users/Audience - Students, Families, Individuals, ...**
  - Find all the concerts happening in New Delhi in the next 3 months.
  - Make a new playlist.
  - Book tickets of a live concert
  - Find the top trending songs
  - Listen to a playlist by their favourite artists.
  - Rate a Live Performance.
  - Search for a specific song.

- **Advertiser - Companies, Sponsors,..**
  - Start a new ad campaign.
  - Calculate the advertising cost for one day.
  - Get the total money spent on ads this month.
  - Delete an advertisement.
  - Updating the current ad Display.

- **Event Organisers**
  - Get the price of an artist.
  - Check for the availability of an artist.
  - Find the best English band in the available budget.
  - Fetch the singer with more than 5 hits and price under 5,00,000.
  - Contract with Artist and fee payment.
  - Checking Sponsors for an event.
  - Setting a schedule for different artists.
  - Fetching the payment made to the Artist.

- **Bankers**
  - Get all the transactions made by a particular user in the past year.
  - Refund a particular transaction
  - Find all the users who have made transactions in a different currency.
  - Fetch the payment made by an advertiser.
  - Transferring amount from users to event organizers.

- **Ticketing Services**
  - Sell tickets for a new live concert.
  - Get how many tickets are bought by somebody.
  - Get how many tickets of ₹ 5000/- were sold in the last concert by Linkin Park.
  - Fetch tickets that were added by an event organizer last month.
  - Find out which event sold the least tickets last month.

Mysql queries for various stakeholders (JDBC).

- **User**

1. User Signup

```
String query = "insert into USER values("+user.getUser_id()+",'" + user.getName() + "','" +
    user.getCountry() + "','" + user.getEmail() + "','" + user.getDOB() + "','" + user.getGender() +
    "' ,"  + null + ")";
System.out.println(query);
stmt.executeUpdate(query);
query = "insert into USER_AUTH values(" + userAuth.getUser_id() + ",'" +
    userAuth.getPassword() + "')"
```

2. User Login

```
String query = "Select COUNT(*) from USER where binary email = '" + email +"'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
```

3. User Add New Playlist

```
String query = "insert into USER_PLAYLISTS values(" + userPlaylist.getPlaylist_id() + "," +
    userPlaylist.getUser_id()  + " ,'" + userPlaylist.getName().toString() + "'," +
    userPlaylist.getSong_num() + ")";
System.out.println(query);
stmt.executeUpdate(query);
```

4. User Make a new Transaction → Part of Booking

```
query = "start transaction";
System.out.println(query);
stmt.executeUpdate(query);

stmt = connection.createStatement();
query = "select * from USER_WALLET where user_id = " + user.getUser_id();
System.out.println(query);
```

```
ResultSet rs = stmt.executeQuery(query);
rs.next();
double wallet = rs.getInt("amount");
if(wallet - amount < 0) throw new InsufficientBalanceException("Not Enough Balance in
    your account");
double balance = wallet - amount;

System.out.println("Amount : " + amount);
query = "update  USER_WALLET  set amount =  " + balance + " where user_id = " +
    user.getUser_id();
System.out.println("Balance : " + balance);
System.out.println(query);
stmt.executeUpdate(query);
Random rand = new Random();
r = rand. nextInt(9000000) + 1000000;

query = "insert into TRANSACTIONS values (" + r + ",'Wallet'"  + "," + 1 + "," + amount + "," +
    "'rupees'" + "," + "'2020-04-30 00:00:00'" + ")";
System.out.println(query);
stmt.executeUpdate(query);

query = "commit";
System.out.println(query);
stmt.executeUpdate(query);
```

5. Remove a booking and refund the transaction

```
query = "start transaction";
System.out.println(query);
stmt.executeUpdate(query);

query = "update BOOKINGS set status = " + "'cancelled'" + "where booking_id = "  +
    Integer.parseInt(booking_id.getText());
System.out.println(query);
stmt.executeUpdate(query);

query = "select ticket_id from BOOKINGS where booking_id = " +
    Integer.parseInt(booking_id.getText());
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
rs.next();
int ticket_id = rs.getInt("ticket_id");

query = "select amount from TICKETS where ticket_id = " + ticket_id;;
System.out.println(query);
ResultSet rs2 = stmt.executeQuery(query);
rs2.next();
double amount = rs2.getFloat("amount");

query = "update USER_WALLET set amount = amount + " + amount + "where user_id = "  +
    MainScreenController.getUser().getUser_id();
System.out.println(query);
stmt.executeUpdate(query);

query = "commit";
System.out.println(query);
stmt.executeUpdate(query);
```

6. Sort your songs by Artists

```
query = "select name from ALL_ARTISTS where artist_id in (select artist_id from
ALL_SONGS as S where S.artist_id = artist_id and S.song_id in (select song_id from
ALL_SONGS where song_id in (select song_id from PLAYLIST_SONGS where playlist_id
```

```
in (select playlist_id from USER_PLAYLISTS where user_id = " +
MainScreenController.getUser().getUser_id() + ")) ))";

System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
while(rs.next()){
  list.add(rs.getString("name"));
  System.out.println(rs.getString("name"));
}
```

7. Sort your songs by Artists

```
query = "select name from ALBUMS where artist_id in (select artist_id from ALL_SONGS
as S where S.artist_id = artist_id and S.song_id in (select song_id from ALL_SONGS
where song_id in (select song_id from PLAYLIST_SONGS where playlist_id in (select
playlist_id from USER_PLAYLISTS where user_id = " +
MainScreenController.getUser().getUser_id() + ")) ))";

System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
while(rs.next()){
  list.add(rs.getString("name"));
  System.out.println(rs.getString("name"));
}
```

- **Musical Professionals**

1. Add Song to Database

```
String query = "Select * from ALL_ARTISTS where name = '" + artist.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int artist_id;
int album_id;
rs.next();
artist_id = rs.getInt("artist_id");

if(rs.getRow() == 0){
  throw  new InvalidEntryException("Artist not found");
}

query = "Select * from ALBUMS where name = '" + album.getText()+ "'";
System.out.println(query);
ResultSet rs2 = stmt.executeQuery(query);
rs2.next();
album_id = rs2.getInt("album_id");
if(rs2.getRow() == 0){
  throw  new InvalidEntryException("ALbum not found");
}
Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;
```

```
query = "insert into ALL_SONGS values(" + r +",'" + title.getText() + "','" + lang.getText() + "','"+
    artist_id + ","+ album_id + "," + 0  + ",'" + duration.getText() + "'," + "'2020-04-30'" +",'" +
    genre.getText() + "')";
System.out.println(query);
stmt.executeUpdate(query);
query = "insert into SONG_GENRE values(" + r + ",'" + genre.getText() + "')";
System.out.println(query);
stmt.executeUpdate(query);
```

2. Add album to database

```
String query = "Select * from ALL_ARTISTS where name = '" + artist.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int artist_id;
int album_id;
rs.next();
if(rs.getRow() == 0) throw  new InvalidEntryException("ALbum not found");
artist_id = rs.getInt("artist_id");

Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;

query = "insert into ALBUMS values(" + r +",'" + title.getText() + "','"+ artist_id + ",'" +
    label.getText()+ "'," + "'2020-04-30'" + ",'" + genre.getText() + "')";
System.out.println(query);
stmt.executeUpdate(query);
query = "insert into ALBUM_GENRE values(" + r + ",'" + genre.getText() + "')";
System.out.println(query);
stmt.executeUpdate(query);
respone.setText("Successfully Added Album ");
connection.close()
```

3. Remove Song From Database

```
String query = "Select * from ALL_SONGS where title = '" + title.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int song_id;
rs.next();
if(rs.getRow() == 0) throw  new InvalidEntryException("Song not found");
song_id = rs.getInt("song_id");

Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;
query = "delete from SONG_GENRE where song_id = " + song_id ;
System.out.println(query);
stmt.executeUpdate(query);
query = "delete from ALL_SONGS where song_id = " + song_id ;
System.out.println(query);
stmt.executeUpdate(query);
respone.setText("Successfully Removed Song ");
connection.close();
```

- **Event Organisers**

1. Add a Live Performance

```
String query = "Select * from ALL_ARTISTS where name = '" + artist.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int artist_id;
int advertiser_id;
rs.next();
artist_id = rs.getInt("artist_id");
if(rs.getRow() == 0)
  throw  new InvalidEntryException("Artist not found");
query = "Select * from ADVERTISERS where name = '" + advertiser.getText()+ "'";
System.out.println(query);
ResultSet rs2 = stmt.executeQuery(query);
rs2.next();
advertiser_id = rs2.getInt("advertiser_id");
if(rs2.getRow() == 0)
  throw  new InvalidEntryException("ALbum not found");

Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;
query = "insert into LIVE_PERFORMANCES values(" + r +"," + artist_id+ ",'" +
      sponsors.getText() + "',"+ advertiser_id + ","+ null + "," + 0 + "," +
      Integer.parseInt(ticCount.getText())+ ")";
System.out.println(query);
stmt.executeUpdate(query);
int r2 = rand. nextInt(9000000) + 1000000;
query = "insert into SCHEDULE values(" + r2 +"," + r+ ",'" + venue.getText() + "','" +
      date.getText() + " " + time.getText() +"')";
System.out.println(query);
stmt.executeUpdate(query);

query = "insert into LIVE_PERF_TICKETS values(" + r +","  +
      Integer.parseInt(ticCount.getText())+ ")";
System.out.println(query);
stmt.executeUpdate(query);
```

2. Remove a Live Performance

```
String query ;
query = "delete from SCHEDULE where live_perf_id = " +
      Integer.parseInt(live_perf_id.getText()) ;
System.out.println(query);
stmt.executeUpdate(query);

query = "delete from LIVE_PERF_TICKETS where live_perf_id = " +
      Integer.parseInt(live_perf_id.getText()) ;
System.out.println(query);
stmt.executeUpdate(query);

query = "delete from LIVE_PERFORMANCES where live_perf_id = " +
      Integer.parseInt(live_perf_id.getText()) ;
System.out.println(query);
stmt.executeUpdate(query);
```

3. Check Advertiser is available or not

```
if(!r1) query = "select * from ADVERTISERS where name = '"+ advertiser.getText() + "'";
else{
  query = "select * from ADVERTISERS";
}
if(advertiser.getText().equals("")) query = null;
if(query!=null) {
  System.out.println(query);
  ResultSet rs = stmt.executeQuery(query);
  int k = 0;
  ResultSetMetaData rsmd = rs.getMetaData();
  String s1 = rsmd.getColumnName(1) + " | " + rsmd.getColumnName(2) + " | " +
      rsmd.getColumnName(3) + " | " + rsmd.getColumnName(4) + " | " +
      rsmd.getColumnName(5)  + "\n" ;
  respone.appendText(s1);
  while (rs.next()) {
     ++k;
     String s = rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3) + " " + rs.getString(4) + " " +
      rs.getString(5) +  "\n";
     respone.appendText(s);
  }
  if (k == 0) {
     throw new InvalidEntryException("Entry not found");
  }
  respone.appendText("Done ");
}
else{
  respone.setText("Empty Entry !!!!!!!!");
}
```

- **Banking Services**

1. Handle Transactions from users

```
boolean r1 = payment.getText().equals("");
boolean r2 = currency.getText().equals("");
boolean r3 = amount.getText().equals("");
if(r1 && r2 && r3)query = null ;
else if(r1 && r2 && !r3)query = "select * from TRANSACTIONS where payment_amount =" +
      Float.parseFloat(amount.getText()) ;
else if(r1 && !r2 && r3)query = "select * from TRANSACTIONS where  currency = '" +
      currency.getText() + "'";
else if(r1 && !r2 && !r3)query = "select * from TRANSACTIONS where  currency = '" +
      currency.getText() + "' and payment_amount =" + Float.parseFloat(amount.getText());
else if(!r1 && r2 && r3) query = "select * from TRANSACTIONS where payment_type = '" +
      payment.getText() + "'";
else if(!r1 && r2 && !r3) query = "select * from TRANSACTIONS where (payment_type = '" +
      payment.getText() + "' and payment_amount =" + Float.parseFloat(amount.getText())
      +")";
else if(!r1 && !r2 && r3) query = "select * from TRANSACTIONS where (payment_type = '" +
      payment.getText() + "' and  currency = '" + currency.getText() +"')";
else if(!r1 && !r2 && !r3) query = "select * from TRANSACTIONS where (payment_type = '" +
      payment.getText() + "' and  currency = '" + currency.getText() + "' and payment_amount
      =" + Float.parseFloat(amount.getText()) +")";
```

```
else{}
if(query!=null) {
  System.out.println(query);
  ResultSet rs = stmt.executeQuery(query);
  int k = 0;
  while (rs.next()) {
     ++k;
     String s = rs.getInt(1) + " " + rs.getString(2) + " " + rs.getInt(3) + " " + rs.getFloat(4) + " " +
      rs.getString(5) + " " + rs.getString(6) + "\n";
     respone.appendText(s);
  }
  if (k == 0) {
     throw new InvalidEntryException("Entry not found");
  }
```

2. Make a transaction on a special request

```
query = "Select * from BANKS where name = '" + bank.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int bank_id;
rs.next();
bank_id = rs.getInt("bank_id");
if(rs.getRow() == 0){
  throw  new InvalidEntryException("Artist not found");
}
Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;
query = "insert into TRANSACTIONS values(" + r +",'" + payment.getText() + "'," + bank_id + ","+
      Float.parseFloat(amount.getText()) + ",'" + currency.getText() + "'," + null +")";
System.out.println(query);
stmt.executeUpdate(query);
```

3. Check if a Bank is Active on our network or not

```
boolean r1 = bank.getText().equals("All");
if(!r1) query = "select * from BANKS where name = '"+ bank.getText() + "'";
else{
      query = "select * from BANKS";
}
if(bank.getText().equals("")) query = null;
if(query!=null) {
  System.out.println(query);
  ResultSet rs = stmt.executeQuery(query);
  int k = 0;
  ResultSetMetaData rsmd = rs.getMetaData();
  String s1 = rsmd.getColumnName(1) + " | " + rsmd.getColumnName(2) + " | " +
     rsmd.getColumnName(3) + " | " + rsmd.getColumnName(4) + " | " +
     rsmd.getColumnName(5) + " | " + rsmd.getColumnName(6) + "\n" ;
  respone.appendText(s1);
  while (rs.next()) {
     ++k;
     String s = rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3) + " " +
     rs.getString(4) + " " + rs.getString(5) + " " + rs.getString(6) + "\n";
     respone.appendText(s);
```

- **Ticketing Services**

1. Add a new Ticket to an Existing Live Performance

```
String query = "Select * from TICKETING_SERVICE where name = '" + name.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int tic_serv_id;
rs.next();
tic_serv_id = rs.getInt("tic_serv_id");
if(rs.getRow() == 0){
  throw  new InvalidEntryException("Artist not found");
}
Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;
query = "insert into TICKETS values(" + r +"," + Integer.parseInt(lp_id.getText() )+ "," +
      tic_serv_id + ","+ Float.parseFloat(amount.getText()) + ","+null+")";
System.out.println(query);
stmt.executeUpdate(query);
```

2. Remove tickets

```
String query = "delete from TICKETS where ticket_id = " + tic_id.getText() ;
System.out.println(query);
stmt.executeUpdate(query);
```

- **Advertiser**

1. Post an Ad to our network

```
String query = "Select * from ADVERTISERS where name = '" + name.getText()+ "'";
System.out.println(query);
ResultSet rs = stmt.executeQuery(query);
int advertiser_id;
rs.next();
advertiser_id = rs.getInt("advertiser_id");

if(rs.getRow() == 0){
  throw  new InvalidEntryException("Advertiser not found");
}
Random rand = new Random();
int r = rand. nextInt(9000000) + 1000000;
query = "insert into ADVERTISEMENTS values(" + r +"," +advertiser_id + ","+null+")";
System.out.println(query);
stmt.executeUpdate(query);
```

2. Remove an Ad

```
String query = "Select *  from  ADVERTISEMENTS where ad_id = " +
      Integer.parseInt(ad_id.getText());
ResultSet rs = stmt.executeQuery(query);
rs.last();
if(rs.getRow() == 0){
  throw  new InvalidEntryException("Advertiser not found");
}
query = "delete from  ADVERTISEMENTS where ad_id = " + Integer.parseInt(ad_id.getText());
```

```
System.out.println(query);
stmt.executeUpdate(query);
```

Indexing

We have made multiple Indexes to streamline the query search process.

- To reduce User search time we have indexes on ALL_SONGS based one
    - Genre
    - Artist
    - Album
- We have created indexes on ALL_ALBUMS based on 'Genre' so as to reduce User and backend queries regarding the same;
- We have created indexes on Booking based in booking status, so as queries for Event Organisers are optimised.
- We have created indexes on Subscription based on subscription type.
- We have created indexes on Transactions based on currency and payment type.

Mysql Commands for creating Indexes.

```
create index genre on SONG_GENRE(genre);
create index artist_genre on ARTIST_GENRE(genre);
create index album_name on ALBUMS(name);
create index album_genre on .ALBUM_GENRE(genre);
create index booking_status on BOOKINGS(status);
create index sub_type on .SUBSCRIPTION(sub_type);
create index currency_type on TRANSACTIONS(currency);
create index payment_type on TRANSACTIONS(payment_type);
```

Domains / Check Constraints

We have made domains for multiple attributes

- Music professionals can only add English, Hindi and Punjabi songs as of our current version.
- Bookings can only be of the type **Confirmed**, **Pending**, **Booked**, **Cancelled** depending upon the status of the user booking.
- User Subscriptions can only be of the type **Free**, **Premium**, **Family** or **Student**.
- Users can make payment through only four methods viz. **NetBanking**, **DebitCard**, **CreditCard**, **UPI** or **Wallet**.
- Transactions can be made only in **INR**, **USD**, **EURO** or **POUNDS.** No other currency will be accepted.

Mysql Commands for creating check constraints.

```
alter table ALL_SONGS add constraint language_check  check( language in ('english' ,'hindi', 'punjabi'));
alter table BOOKINGS add constraint booking_status check(status in ('confirmed', 'pending', 'booked', 'cancelled'));
alter table SUBSCRIPTION add constraint sub_type_check check(sub_type in ('free', 'premium', 'family', 'student'));
alter table TRANSACTIONS add constraint payment_type_check check(payment_type in ('NetBanking', 'DebitCard', 'CreditCard', 'Upi', 'Wallet'));
alter table TRANSACTIONS add constraint currency_type_check check(currency in ('inr', 'usd', 'euro', 'pounds'));
```

Aggregation Functions

- **Ranking**
- We have used ranking to get the **Top 10** songs and artists, based on their likes and popularity.
    - Artists Ranking query

```
select name, (1 + (select count(*) from ALL_ARTISTS A where A.popularity>B.popularity))
as Ranking
from ALL_ARTISTS B
order by Ranking limit 10;
```

    - Songs Ranking query

```
select title, name, (1 + (select count(*) from ALL_SONGS A where A.likes>B.likes)) as
Ranking
from ALL_SONGS B, ALL_ARTISTS
where B.artist_id = ALL_ARTISTS.artist_id
order by Ranking limit 10;
```

Bonus Components

- We have developed a working **PC music streaming software** in JavaFX, in which users can stream multiple songs as per their convenience.
- To reduce errors, we have added **REGEX** so new Users cannot fill wrong details accidentally.
- Users can select from the automatically generated playlists on their Home Screen, or create their own.
- They can also **Search** for their favorite Songs or Artists, using **Search Filters** provided for ease of access.
- The music player is **Dynamic** and uses **Multithreading** to display the current timestamp of the song. The user can like the song, loop it, or pause/play the song anytime.
  The music is stored as a **BLOB** on the database for portability of the songs.
- To reduce search times, we have incorporated multiple **Indexes** in the database.
- We have also added specific **Domains** to tables to prevent Users from entering wrong values.
- We have also made **Transactions** with **Commit** and **Rollback** functions to make them atomic.

Future Scalability

- We can add the option to upload **Podcasts** and **Standups**
- **File sizes** and **Connection Speed** can be improved.
- **UI and UX** can be improved.
- Can be developed for **Android** as well.

Individual Contribution

- **Amit Maurya (2018015)**
  - Created Event Page with Main Event Playlist and Places near you Playlist.
  - Booking Event Page.
  - Search filter in events based on Date, Artist and venue.
  - Filled data into database.
  - BackEnd and FrontEnd for Events at different places .
  - BackEnd and FrontEnd for Booking Events .
- **Arka Sarkar (2018222)**
  - Database and Schema Designing and implementation.
  - Backend users (Adver, Event Org, Music prof, Banks) backend and frontend implementation.
  - User Signup and login backend implementation.
  - Implemented AutoLogin (App Cache) feature for the current user, Error and Exception handling in the project.
  - Add Playlist and Your Library backend and frontend implementation.
  - Event Booking and Transactions backend implemented.
  - Database Management.
- **Dhruv Yadav (2018281)**
  - Document Management
  - Schema Designing
  - Software UI/UX Design.
  - UI Graphics Design
  - Media Player backend and frontend
  - Songs upload and management.
  - Playlists Creation and management
  - Database Management
- **Harman Singh (2018284)**
  - I have designed (Front end) of the following pages - Home Page, Login Page, Sign-Up, Search, Search filters, Profile page, User details edit page.
  - Implemented search algorithm of the project. (Backend + Frontend)
  - Added Search Filters for searching.
  - Play songs and Add to playlist functionality (After Searching)
  - Implemented Profile page of the user.
  - User profile edit details functionality
  - Executed SQL queries for the implemented functionalities.
- **Nikhil Dahiya (2018057)**
  - Filled data into database.
  - Database Management
  - Page Design

Github Link : https://github.com/ArkaSarkar19/BAJAO