DBMS Project

Work in Progress Document

Group No. 11

Team Name: Lazarus

MUSIFY

Our project involves developing a **Music Management Application** named '**Musify**' that provides its users with a variety of songs to listen to based on their choices of **mood**, **genre**, **language, artist and film**.

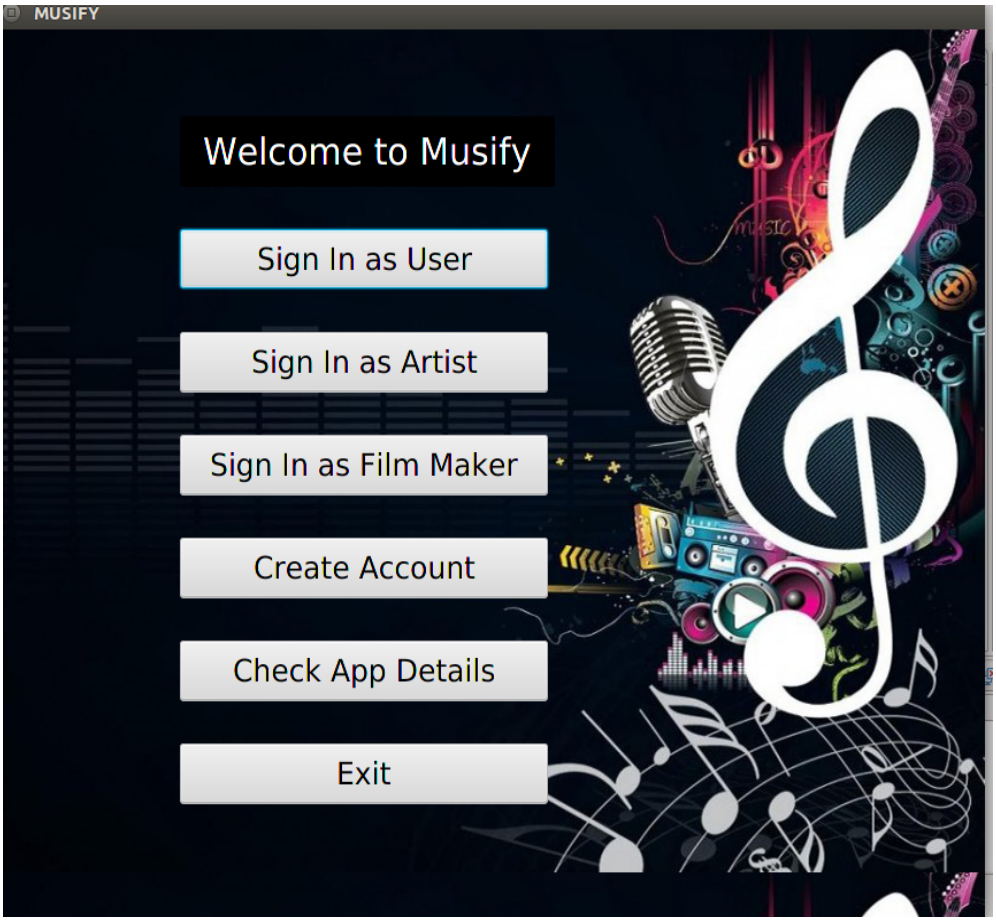
This Application also provides support to a variety of **artists** to showcase their art to the audience by **adding their newly sung/composed songs** for our users and earn.

The filmmakers also have the option to upload all the songs of their films.

The songs can be rated by the users and also added to the list of favorite songs for the users.

A user logs into the Application with LoginID and Password. He/she is also given the option of availing for a subscription and upgrade to a premium user, if a normal user.

The Application also supports additional benefits like display of lyrics and image along with the song being played.



WEEK WISE PROGRESS

Week1

Stakeholders

1. Administrator
2. User (Premium/ Normal)
3. Artists
4. Filmmakers

Week2

Different users will be using the system for different purposes. These are listed below:

Administrator

1. The administrator can find which song is mostly searched by the user.
2. The administrator can check all premium users.
3. The administrator can check the artist followed by most users.
4. The administrator can check the total number of songs in the app.
5. The administrator can access the payment details of an artist.

Listener (User)

1. The listener can search the song by the song's name.
2. The listener can search the song he/she wishes to listen to on the basis of the language.
3. The listener can search a song by its genre or mood.
4. The user can search all songs of an artist or a film.
5. The listener can check his/her favorites list.
6. The listener can check his/her recently played music tracks.
7. The listener can update their account in the premium account.

Artist

1. An artist can add their newly composed song to their playlist.
2. Artists can check their payment details.
3. Artists can check their composed songs added in the Application database.
4. Artists can also search over all the songs currently present in the database and listen to any based on language, genre, mood, artist or film.
5. Artists can write a bio, save it to their profile, and update it at any time.

Filmmaker

1. Filmmakers can add their film to the Application.
2. Filmmakers can update their added films (Description).
3. Filmmakers can see an artist who has worked in their movie.
4. Filmmakers can see all the added films.
5. Film makers can also search over all the songs currently present in the database and listen to any based on language, genre, mood, artist or film.

Week3

Database Schema

The following relations will be used in the backend of our application.

1) **User Table**: To store the details of the various users of our application, we define the following table:

   User (UserID **Integer,** UserName **varchar(100),** LoginID **varchar(100),** Password **varchar(20),** Subscription **boolean,** Followers **Integer,** FavouriteArtist **varchar(100), primary key**(UserID), **unique**(LoginID, Password))

2) **Artist Table**: To store the details of the Artists, we define the following Table:

   Artist (ArtistID **Integer**, ArtistName **varchar(100)**,LoginID **varchar(100),** Password **varchar(20),** TotalSongs **Integer**, UsersFollowing **Integer**, ArtistBio **varchar(1000),** Payment **Integer, primary key**(ArtistID), **unique**(LoginID, Password))

3) **Film Makers Table**: To store the details of all the FilmMakers, we define the following Table:

   FilmMakers (FilmMakerID **Integer,** FilmMakerName **varchar(100)**, LoginID **varchar(100),** Password **varchar(20),** FilmName **varchar(100)**, FilmID **Integer**, FilmDescription **varchar(1000), primary key**(FilmMakerID, FilmID), **unique**(LoginID, Password))

   We store the songs of different languages in separate tables. This is to ensure faster retrieval of information upon a user query.

4) **Hindi Songs Table**: To store all Bollywood (Hindi) songs, we define the following table:

   Hindi_Songs (SongID **Integer,** SongName **varchar(100)**, ArtistID **Integer not null**, ArtistName **varchar(100),** Lyrics **varchar(100000)**, ReleaseDate **Date**, Rating **Integer**, TotalReviews **Integer,** GenreID **varchar(100)**, MoodID **varchar(100) not null,** FilmID **Integer unique**, SongFilePath **varchar(200)**, ImagePath **varchar(200)**, **primary key**(SongID), **foreign key**(GenreID) references **Genre**, **foreign key**(ArtistID) references **Artist**, **foreign key**(MoodID) references **Mood, foreign key**(FilmID) references **FilmMakers**)

5) **English Songs Table**: To store all English songs, we define the following table:

English_Songs (SongID **Integer**, SongName **varchar(100)**, ArtistID **Integer not null**, ArtistName **varchar(100),** Lyrics **varchar(100)**, ReleaseDate **Date**, Rating **Integer**, TotalReviews **Integer**, GenreID **varchar(100) not null**, MoodID **varchar(100) not null**, FilmID **Integer not null**, SongFilePath **varchar(200)**, ImagePath **varchar(200)**, SongFilePath **varchar(200)**, Image **Blob(16)**, **primary key**(SongID), **foreign key**(GenreID) references **Genre**, **foreign key**(ArtistID) references **Artist**, **foreign key**(MoodID) references **Mood, foreign key**(FilmID) references **FilmMakers**)

6) **Punjabi Songs Table**: To store all Punjabi songs' details, we define the following table:

Punjabi_Songs (SongID **Integer**, SongName **varchar(100)**, ArtistID **Integer not null**, ArtistName **varchar(100),** Lyrics **varchar(100)**, ReleaseDate **Date**, Rating **Integer**, TotalReviews **Integer**, GenreID **varchar(100) not null**, MoodID **varchar(100) not null**, FilmID **Integer not null**, SongFilePath **varchar(200)**, ImagePath **Varchar(200)**, **primary key**(SongID), **foreign key**(GenreID) references **Genre**, **foreign key**(ArtistID) references **Artist**, **foreign key**(MoodID) references **Mood, foreign key**(FilmID) references **FilmMakers**)

7) **Gujrati Songs Table**: To store the Gujrati songs, we define the following table:

Gujrati_Songs (SongID **Integer**, SongName **varchar(100)**, ArtistID **Integer not null**, ArtistName **varchar(100),** Lyrics **varchar(100)**, ReleaseDate **Date**, Rating **Integer**, TotalReviews **Integer**, GenreID **varchar(100) not null**, MoodID **varchar(100) not null**, FilmID **Integer not null**, SongFilePath **varchar(200)**, ImagePath **varchar(200)**, **primary key**(SongID) **foreign key**(GenreID) references **Genre**, **foreign key**(ArtistID) references **Artist**, **foreign key**(MoodID) references **Mood, foreign key**(FilmID) references **FilmMakers**)

8) **Korean Songs Table**: To store the korean song details, we define the following table:

Korean_Songs (SongID **Integer,** SongName **varchar(100),** ArtistID **Integer not null,** ArtistName **varchar(100),** Lyrics **varchar(100),** ReleaseDate **Date,** Rating **Integer,** TotalReviews **Integer,** GenreID **varchar(100) not null,** MoodID **varchar(100) not null,** FilmID **Integer not null,** SongFilePath **varchar(200),** ImagePath **varchar(200),** **primary key**(SongID, SongName)**, foreign key**(GenreID) references **Genre, foreign**

key(ArtistID) references **Artist, foreign key**(MoodID) references **Mood, foreign key**(FilmID) references **FilmMakers)**

9) **Genre Table**: To store all the available music genres, we define the following table:

Genre (GenreID **Integer not null,** GenreName **varchar(100), primary key**(GenreID))

10) **Mood Table**: To store the various moods in songs, we define the following table:

Mood (MoodID **Integer not null,** MoodName **varchar(100), primary key**(MoodID))

11) **Favorite songs Table**: To store the favorite songs of users so as to prepare the playlist of their favorite songs, we define the following table:

Favorite_Songs (SongID **Integer not null,** UserID **Integer not null, foreign key**(UserID) references **User)**

12) **SongsListened Table**: To store the songs listened by the users, we define the following table:

Songs_Listened (SongID **Integer not null,** UserID **Integer not null, foreign key**(UserID) references **User)**

BONUS MARKS

As Bonus, We have implemented the idea of **recommending songs** to a user upon his login:

For all the favourite songs of the user, we retrieve the artists of those songs and the users who also have that song in their favorites list and recommend the songs of that artist and matching users' favourites to that particular user.

SQL Queries for BONUS task of Recommending Songs to the User:

We retrieve the user favorite songs as follows:

String **userFavSongs**= "(select SongID from Favorite_Songs where userID = " + userID+")";

We then check for the users having the songs of the particular user also as their favourite songs:

String **matchingUser** = "(select UserID from Favorite_Songs where songID in " + userFavSongs+")";

We find the artist of the songs that are the user's favorite.

String **matchingArtist** = "select ArtistID from English_Songs where SongID in ("+userFavSongs+") union " +"select ArtistID from Hindi_Songs where SongID in ("+userFavSongs+") union "+"select ArtistID from Punjabi_Songs where SongID in ("+userFavSongs+") union "+"select ArtistID from Korean_Songs where SongID in ("+userFavSongs+") union "+"select ArtistID from Gujrati_Songs where SongID in ("+userFavSongs+")";

We recommend the songs of the user with matching interests (stored in matchingUser) and the artist's all songs whose songs the user likes.

String songQuery1 = "select SongID from Favorite_Songs where userID in " + matchingUser;

String songQuery2 = "select SongID from English_Songs where ArtistID in ("+matchingArtist+") union " +"select SongID from Hindi_Songs where ArtistID in ("+matchingArtist+") union "+"select SongID from Punjabi_Songs where ArtistID in ("+matchingArtist+") union "+"select SongID from Korean_Songs where ArtistID in ("+matchingArtist+") union "+"select SongID from Gujrati_Songs where ArtistID in ("+matchingArtist+")";

We combine the results which are the songs to be recommended

String songQuery = songQuery1 +" union "+songQuery2;

From the song ID's retrieved above, we retrieve the song details from the various language tables using union operator

String query ="select SongName from English_Songs where SongID in ("+songQuery+") union " +"select SongName from Hindi_Songs where SongID in ("+songQuery+") union "+"select SongName from Punjabi_Songs where SongID in ("+songQuery+") union "+"select SongName from Korean_Songs where SongID in ("+songQuery+") union "+"select SongName from Gujrati_Songs where SongID in ("+songQuery+");"

Week4

Populated the data in the above 12 tables, with the following no. of entries per tables:

Table	No. of Entries
User Table	200
Artist Table	100
Film Makers Table	45
Hindi Songs Table	27
English Songs Table	100
Punjabi Songs Table	20
Gujrati Songs Table	30
Korean Songs Table	25
Genre Table	9
Mood Table	6
Favorite songs Table	200

Songs Listened Table	200

Week6

Attributes as Indexes:

For making the indices we have made several searching keys on which the user will make search for the desired songs like:-

1. To search on the basis of the artist's name.
2. To search on the basis of film name.
3. To search on the basis of mood type.
4. To search on the basis of genre type.
5. Look for the user's favourites.
6. Look for the user’s recently played songs.

Therefore,

The indexes were made in **each language song table** on the following attributes:-

1. Artist ID
2. File ID
3. Mood ID
4. Genre ID
5. SongName

To incorporate searching based on names, We also made the following indices:

1. The indexes were made on the **Artist** table on the ArtistName attribute.

2. The indexes were made on the **FilmMakers** table on the FilmName attribute.
3. The indexes were made on the **Genre** table on the GenreName attribute.
4. The indexes were made on the **Mood** table on the MoodName attribute.

Also, for fetching user's favourite and recently played songs using UserID,

The indexes were made on the **Favourite_Songs** table on the UserID attribute.

The indexes were made on the **Songs_Listened** table on the UserID attribute.

Through the making of indices it was made clear that the user search gets optimized in order to fetch details accurately and in less backend search comparisons leading to concized block memory

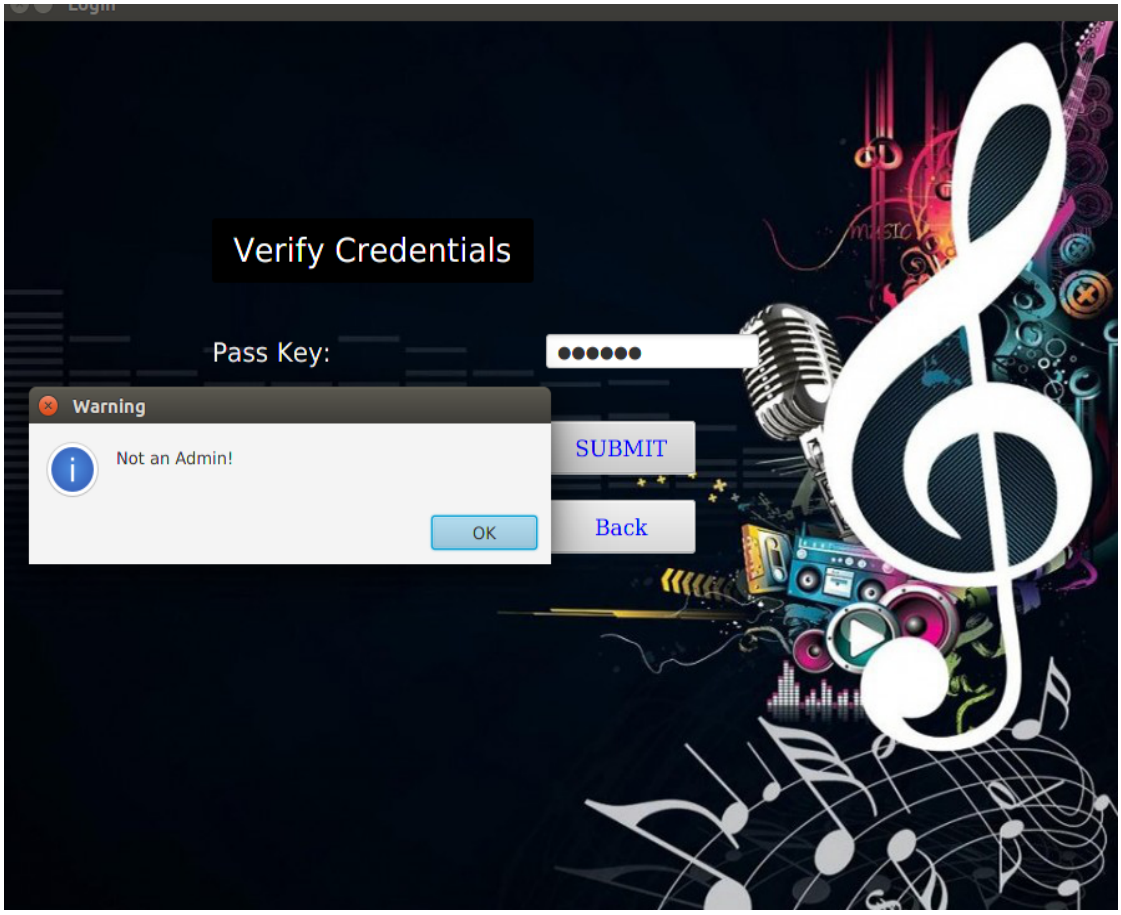
Week7

SQL Queries (Advanced Aggregation functions)

Queries supporting **Admin's** Functionality:

We first check if the Admin credentials are valid i.e the user accessing our system as an Admin is valid Administrator or not.

We do this by validating the pass key entered by the user.



- **Listing all premium users**:

  select * from User where Subscription = 1

- **See the payment details of an Artist (eg: ID 98)**:

  select Payment from Artist where ArtistID = 98

- **Mostly searched song**:

  From the Songs_Listened table, we retrieve the song which has been played by the maximum number of users. **(Aggregation function used - Count and Max)**

  Select SongID, count(UserID) as Count from Songs_Listened group by SongID having Count = (select max(a.count) from (Select SongID, count(UserID) as Count from Songs_Listened group by SongID)a);

- **Artist followed by most of the users**:

From the Artist table, we retrieve the artist who has been followed by the maximum number of users. **(Aggregation function used - Max)**

select ArtistID from Artist where UsersFollowing = (select max(UsersFollowing) from Artist);

- **Total songs in the Application**:

  Union is used to combine the results (songs fetched) of all the language tables.

  select count(*) from (select * from Hindi_Songs union select * from English_Songs union select * from Punjabi_Songs union select * from Korean_Songs union select * from Gujrati_Songs)a;

Queries supporting **Sign-Up** Functionality:

- **Validating that loginID and Password is unique**:

  We first check if the user with the given login ID and password exists. If the below query returns null, then we create the account, else we generate an alert of duplicate login credentials and ask the user to enter a different login ID and password.

  Select * from User where LoginID = \"" + id.getText() + "\"" + "and Password = \"" + pass.getText() + "\";"

  Note : the variables id and pass are the values retrieved from the text boxes for login IDs and password entered by the user on the GUI.

- **To give a unique ID to the user signing up (generated by the system)**

  Each time a new user signs-up we assign him a unique ID = max of existing ID + 1

  To fetch the last registered userID (max user ID) we write the following query: **(Aggregation function used - Max)**

  select UserID from User where UserID=(select max(UserID) from User);

- **Inserting values for the newly signed up user in the User table:**

Insert into User values(" + uid + "," + "\"" + name.getText()+ "\"" + "," + "\""+ id.getText()+ "\"" + "," + "\""+pass.getText()+ "\"" + "," + subs + "," +'0' + ',' + "\""+ favAr.getText()+ "\"" + ");"

Note : the variables name, id, pass and subscription, favArtist are the values retrieved from the text boxes entered by the user on the GUI.

uid - unique userID generated by the system.

id - login ID entered by the user

name - name entered by the user

pass - password entered by the user

favAr - User's favourite artist

Sub - Subscription opted for by the user
(subs = 0 if normal user and subs = 1 for premium user)

If the user opts for premium subscription we make him enter the necessary bank transaction details.

Initially we assign the number of followers of the user = 0

If the user signs up as an artist or film maker, apart from adding him in the user table, we also add his credentials in the artist or film maker table respectively.

- **To give a unique ID to the Artist signing up**

  Each time a new artist signs-up in we assign him a unique ID = max of existing ID + 1

  To fetch the last registered artistID (max artist ID) we write the following query:
  (Aggregation function used - Max)

  select ArtistID from Artist where ArtistID=(select max(ArtistID) from Artist);

- **Inserting into Artist table:**

  Insert into Artist values(" + uid + "," + "\""+ name.getText()+ "\"" + "," + id.getText() + "," +pass.getText() + "0,0," + "\"" + "\","+'0' + ");

Note : the variables name, id, pass and subscription, favArtist are the values retrieved from the text boxes entered by the user on the GUI.

uid - unique artistID generated by the system.

id - login ID entered by the signing artist

name - name entered by the artist

pass - password entered by the artist

Initially we assign the total songs, users following and payment = 0 for the newly signing-up artist. And the Artist Bio is set empty, this can be changed by the artist upon login.

- **To give a unique ID to the FilmMaker signing up**

  Each time a new filmmaker signs-up in we assign him a unique ID = max of existing ID + 1

  To fetch the last registered FilmMakerID (max film maker ID) we write the following query: **(Aggregation function used - Max)**

  select FilmMakerID from FilmMakers where FilmMakerID=(select max(FilmMakerID) from FilmMakers);

  To assign a unique film ID, we make the ID of the newly added film = maximum of existing ID + 1

  To fetch the last registered FilmID (max film ID) we write the following query: **(Aggregation function used - Max)**

  select FilmID from FilmMakers where FilmID=(select max(FilmID) from FilmMakers);

- **Inserting into FilmMakers table:**

  Insert into FilmMakers values(" + uid + "," + "\""+ name.getText() + "\""+ "," + "\""+ id.getText() + "," +pass.getText() +fn.getText() + "\""+ "," + filmID+"," + "\""+ desc.getText() + "\"" + ");"

  fn - film Name entered by the user on GUI

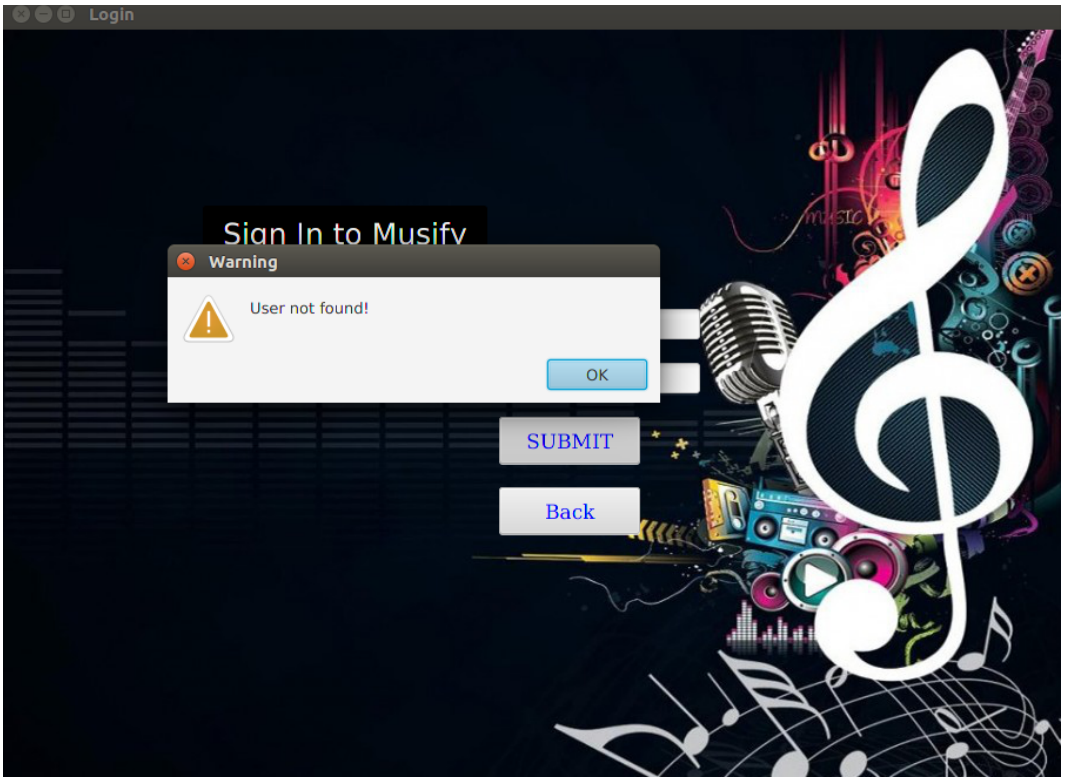
Queries supporting **Login** Functionality:

- **Validating Login ID and Password:**

To check if the login ID and password entered by the user are correct/not, we retrieve all the users in a ResultSet (using query - **Select * from User**) and iterate over it to check if a user with the given login ID exits:

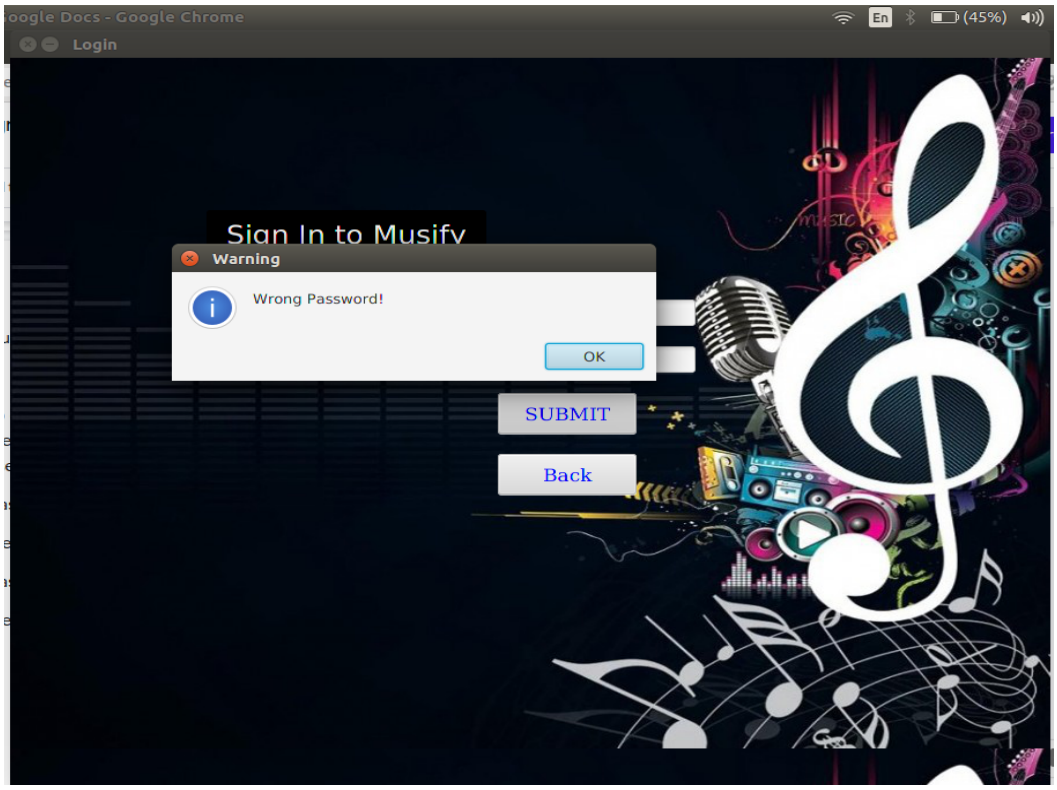
Case 1: LoginID not found:

We generate an alert of no such user found.



Case2 : Login ID matches, but the password does not,

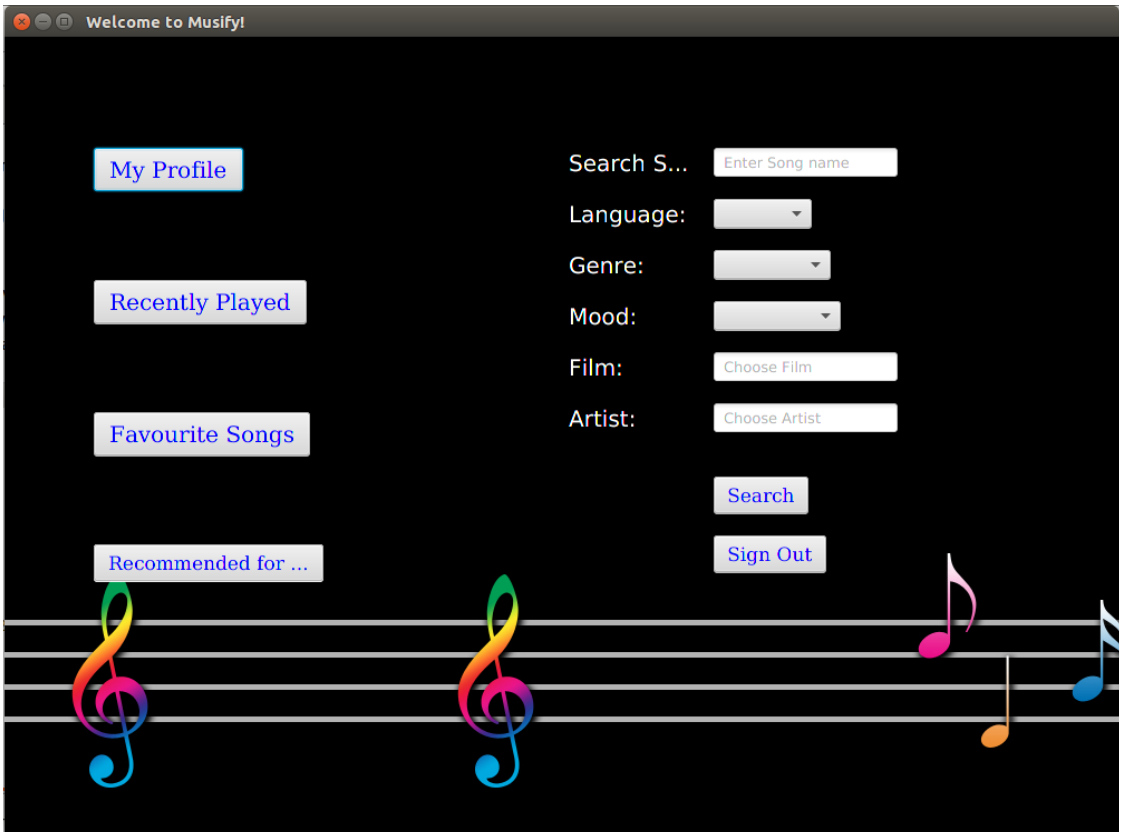
We generate an alert that the password is incorrect.



Else we enter the user page.

The user is given the option of searching songs by the Song Name, Artist Name, Film Name, Genre and mood.

User Page



- **Viewing Profile:**

  "select * from User where UserId = " + userID + ";"

- **Viewing Recently played songs:**

  We retrieve the ID of the songs recently played by the user from the Songs_Listened table using the query

  innerQuery = "(select SongID from Songs_Listened where UserID = " + userID +");"

  and then show the details of the songs by retrieving them from all 5 language Songs tables by performing a union operation as follows

String [] languages = {"Hindi_Songs", "English_Songs", "Korean_Songs", "Punjabi_Songs", "Gujrati_Songs"};

for(int i=0;i<5;i++)

{ qe[i] = "select * from " + languages[i] + " where SongID in " + innerQuery; }

String query = qe[0] + " union " + qe[1] + " union " +qe[2]+" union "+qe[3]+" union "+qe[4]+";";

- **Similarly for Favourite songs:**

  We retrieve the ID of the songs recently played by the user from the Songs_Listened table using the query

  innerQuery = "(select SongID from Favorite_Songs where UserID = " + userID +");"

  and then show the details of the songs by retrieving them from all 5 language Songs tables by performing a union operation as follows:

  String [] languages = {"Hindi_Songs", "English_Songs", "Korean_Songs", "Punjabi_Songs", "Gujrati_Songs"};

  for(int i=0;i<5;i++)

  { qe[i] = "select * from " + languages[i] + " where SongID in " + innerQuery; }

  String query = qe[0] + " union " + qe[1] + " union " +qe[2]+" union "+qe[3]+" union "+qe[4]+";";

- **Updating Subscription:**

  Update User set Subscription = 1 where UserID = 6;

- **Searching over Songs**:

  For searching over songs, we give the option of applying filters to the user:

  - If the user specifies a **genre**:

  We retrieve the genreID of the song as:

(select distinct GenreID from Genre where GenreName = " + "\"" + genre + "\")";

- Else if the user **doesn't specify any genre:**

We search over all possible genres as follows:

gid = any(select distinct GenreID from Genre)

- If the user specifies a **mood**:

We retrieve the moodID of the song as:

(select distinct MoodID from Mood where MoodName = " + "\"" + mood + "\")";

- Else if the user **doesn't specify any mood:**

We search over all possible moods as follows:

mid = any(select distinct MoodID from Mood)

- If the user specifies a **film:**

We retrieve the filmID of the song as:

(select distinct FilmID from FilmMakers where FilmName = " + "\"" + film + "\")";

- Else if the user **doesn't specify any film:**

We search over all possible films as follows:

fid = any(select distinct FilmID from Film)

If the user specifies an **Artist:**

We retrieve the ArtistID of the song as:

(select distinct ArtistID from FilmMakers where ArtistName = " + "\"" + artist + "\")";

- Else if the user **doesn't specify any Artist:**

We search over all possible artists as follows:

aid = any(select distinct ArtistID from Artist)

- If the user specifies a language, we search for the songs only in that language table.

  "select * from " +language + "_Songs"+" where SongName = " + songName + " and ArtistID = " + aID + " and MoodID = " + mID + " and GenreID = "+ gID + " and FilmID = " + fID +";" ;

  Else we do a **union** over all language tables

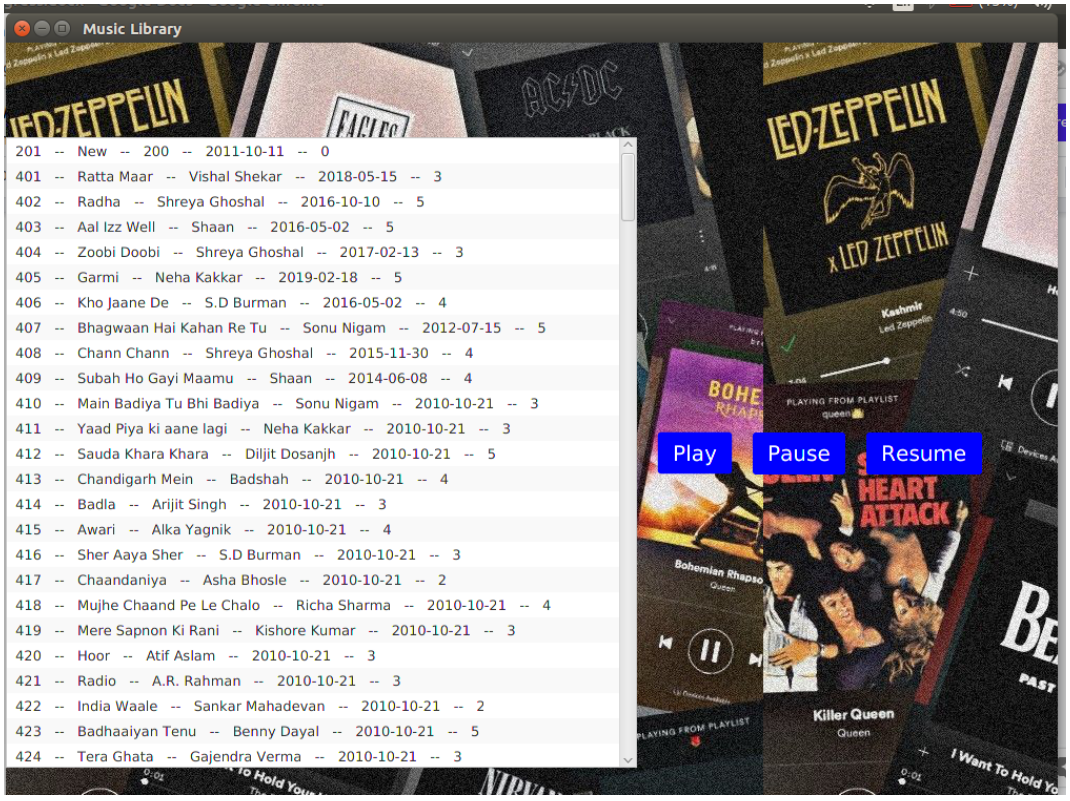
for(int i=0;i<5;i++)

{

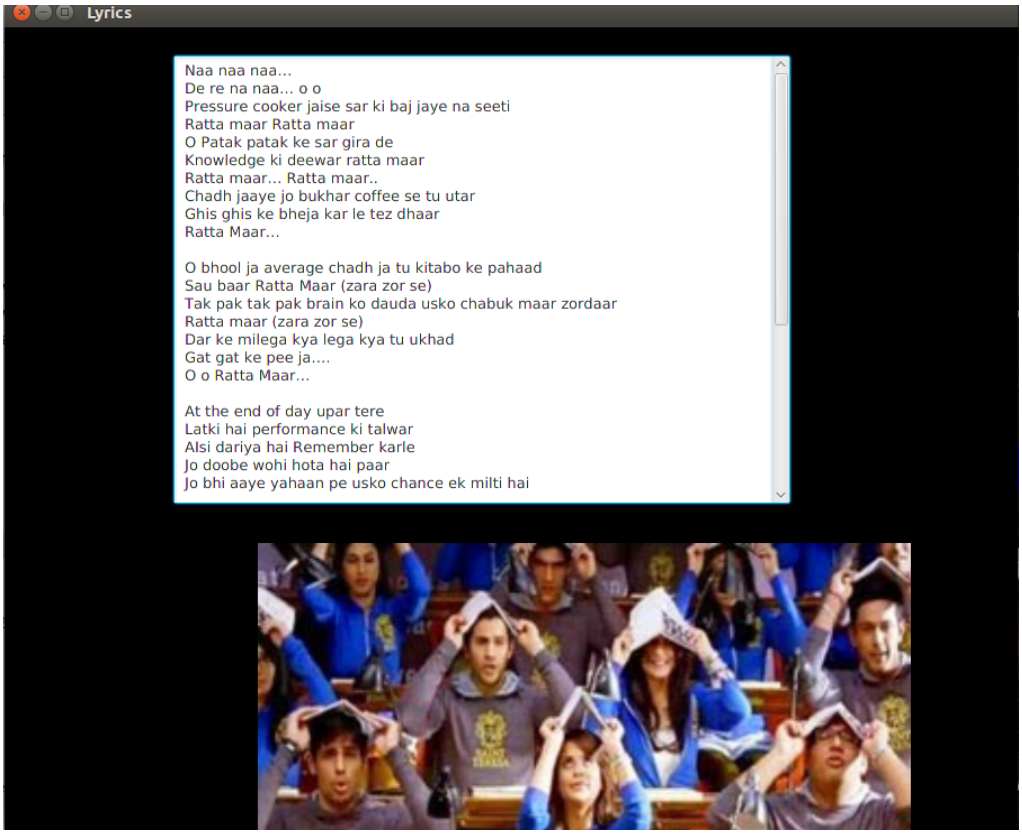
q[i] = "select * from " +languages[i]+" where SongName = " + songName + " and ArtistID = " + aID + " and MoodID = " + mID + " and GenreID = "+ gID + " and FilmID = " + fID;

}

query = q[0] + " union " + q[1] + " union " +q[2]+" union "+q[3]+" union "+q[4]+";";



Display of Lyrics along with song



Queries supporting **Artist's** Functionality:

- **Displaying the details of the Artist:**

  "select * from Artist where ArtistID = " + userID+ ";";

- **Updating the Bio of the Artist:**

  "Update Artist set ArtistBio = \"" +bio.getText()+"\" where ArtistID = " + artistID+";";

- **Displaying the payment of the Artist:**

  We are updating the payment of the artist as follows:

  We first calculate the no. of users who have the songs of that artist as their favourite songs and recently played songs

  Query 1: "select SongID, count(distinct(Favorite_Songs.UserID)) as FavUser, count(distinct(Songs_Listened.UserID)) as RecUser from Favorite_Songs inner join Songs_Listened using(SongID) group by SongID";

Query 2: "select * from English_Songs where ArtistID = " + artistID+ " union " +"select * from Hindi_Songs where ArtistID = " + artistID+ " union " +"select * from Punjabi_Songs where ArtistID = " + artistID+ " union " +"select * from Korean_Songs where ArtistID = " + artistID+ " union " +"select * from Gujrati_Songs where ArtistID = " + artistID+ " ;";

We update the payment as follows:

New payment = original payment + 10*(no of users with his song as favourite_song + those with recently_played_songs)

Query 3: String updatePaymentQuery = "Update Artist set Payment = Payment+" + incrPay+" where ArtistID = " + artistID +";";

- **Displaying all the songs of the Artist**

  select * from English_Songs where ArtistID = " + artistID+ " union " +"select * from Hindi_Songs where ArtistID = " + artistID+ " union " +"select * from Punjabi_Songs where ArtistID = " + artistID+ " union " +"select * from Korean_Songs where ArtistID = " + artistID+ " union " +"select * from Gujrati_Songs where ArtistID = " + artistID+ " ;";

- **Adding a new song for the Artist:**

  Getting unique songID: "SELECT SongID FROM " +cb1.getValue()+"_Songs WHERE SongID=(SELECT max(SongID) FROM "+cb1.getValue()+"_Songs);";

  Getting Genre: "SELECT GenreID from Genre where GenreName = \"" + cb2.getValue() +"\";";

  Getting Mood: "SELECT MoodID from Mood where MoodName = \"" + cb3.getValue() +"\";";

  Getting Film: "SELECT FilmID from FilmMakers where FilmName = \"" + film.getText() +"\";";

  Inserting into the table: "Insert into " + cb1.getValue() + "_Songs values(" + songID + ",\"" + search.getText() +"\","+artistID+",\""+artistName +"\","+ "\"/home/akshyta/Desktop/DBMS_Project/Lyrics/H25.txt\"," + "\"" + rd.getText() + "\"," +

"0,0,"+genreID+","+moodID+","+filmID+",\"/home/akshyta/Desktop/DBMS_Project/Songs/H25.mp3\"," + "\"/home/akshyta/Desktop/DBMS_Project/Images/H1.jpg\";

Queries supporting **Film Maker's** Functionality:

- **Displaying the details of the Film Maker:**

  "select * from filmmaker where FilmMakerID = " + filmMakerID+ ";";

- **Updating the Film Description:**

  "Update FilmMakers set FilmDescription = \"" +ud.getText()+"\" where FilmID = " + filmID+";";

- **Displaying all the movies of the Film Maker:**

  "select * from FilmMakers where FilmMakerID = " + filmMakerID;

- **Displaying the booked Artists:**

  Inner Query:

  for(int i=0;i<5;i++)

  innerQueries[i] = "select ArtistID from "+language[i]+"_Songs natural join FilmMakers where FilmMakerID ="+ filmMakerID;

  innerQuery=innerQueries[0] +" union " +innerQueries[1] +" union"+innerQueries[2] +" union " +innerQueries[3] +" union " +innerQueries[4];

  Outer Query: "select ArtistID, ArtistName from Artist where ArtistID in ";

  Final Query: innerQuery+outerQuery;

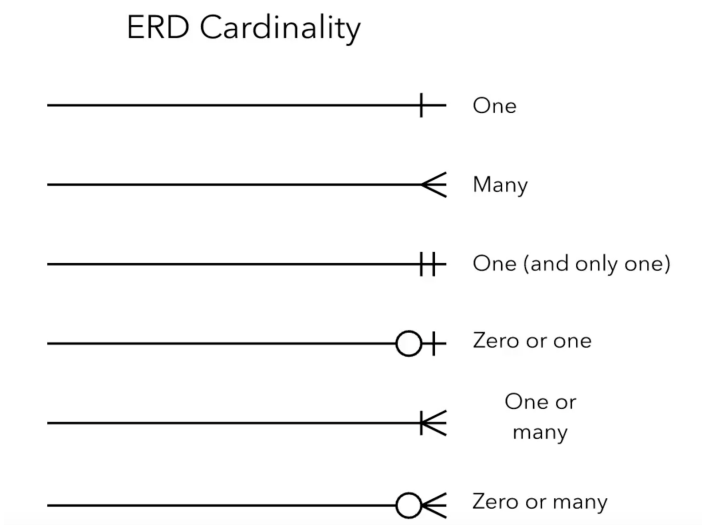
- **Adding a new Film:**

  query = "Insert into FilmMakers values(" + filmMakerID + ",\"" + filmMakerName + "\",\"" + loginID +"\",\""+pass+"\",\""+filmName.getText() +"\","+ filmID+",\"" +fd.getText() + "\");

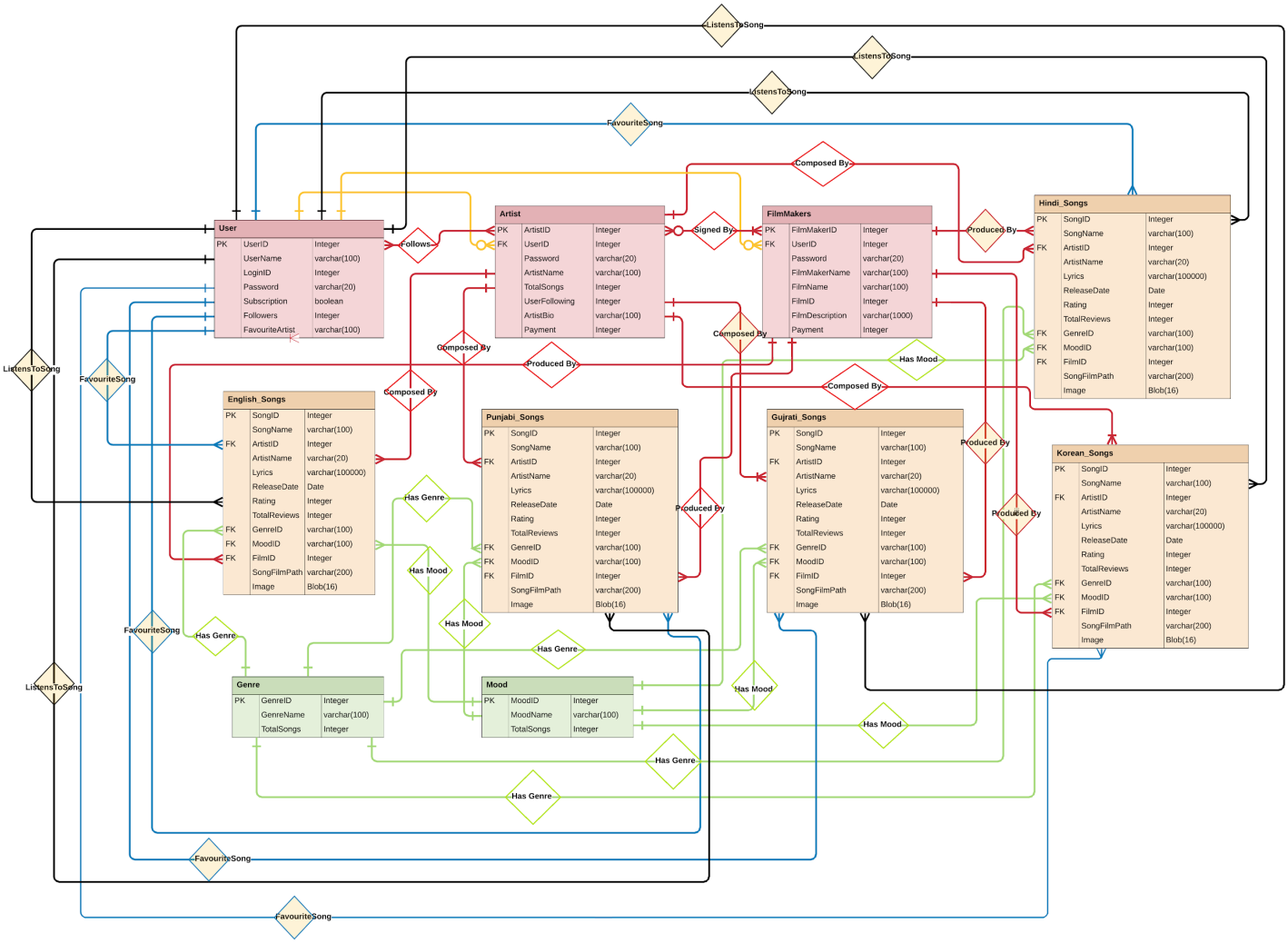
Relational Algebraic Queries

1. $\sigma_{Subscription = 1}$ (User)
2. $_{Payment}$ ($\sigma_{ArtistID = 98}$ (Artist))
3. $_{SongID}$ $_{max(COUNT(SongID))}$(σ_{songID} (Songs_Listened))
4. $_{ArtistID}$ ($\sigma_{max(UsersFollowing)}$(Artist))
5. $_{SongID}$ (Hindi_Songs U English_Songs U Gujrati_Songs U Punjai_Songs U Korean_Songs)
6. $_{UserID + 1}$ ($_{max(UserID)}$(σ_{UserID} (User)))
7. $_{ArtistID + 1}$ ($_{max(ArtistID)}$($\sigma_{ArtistID}$ (Artist)))
8. $_{FilmMakerID + 1}$ ($_{max(FilmMakerID)}$($\sigma_{FilmMakerID}$ (FilmMakers)))
9. $_{All}$ (σ_{UserID} (User))
10. $_{SongName}$ ($\sigma_{MoodID,MoodName}$(Mood) Intersect σ_{MoodID} (Hindi_Songs U English_Songs U Gujrati_Songs U Punjabi_Songs U Korean_Songs))
11. $_{FilmMaker}$($\sigma_{FilmMaker = (}\sigma_{max(FilmMaker))}$(FilmMaker))

Week8



E-R Diagram


ListensToSong
ListensToSong
ListensToSong
FavouriteSong
Composed By
User
PK UserID Integer
UserName varchar(100)
LoginID Integer
Password varchar(20)
Subscription boolean
Followers Integer
FavouriteArtist varchar(100)
Follows
Artist
PK ArtistID Integer
FK UserID Integer
Password varchar(20)
ArtistName varchar(100)
TotalSongs Integer
UserFollowing Integer
ArtistBio varchar(100)
Payment Integer
Signed By
FilmMakers
PK FilmMakerID Integer
FK UserID Integer
Password varchar(20)
FilmMakerName varchar(100)
FilmName varchar(100)
FilmID Integer
FilmDescription varchar(1000)
Payment Integer
Produced By
Hindi_Songs
PK SongID Integer
SongName varchar(100)
FK ArtistID Integer
ArtistName varchar(20)
Lyrics varchar(100000)
ReleaseDate Date
Rating Integer
TotalReviews Integer
FK GenreID varchar(100)
FK MoodID varchar(100)
FK FilmID Integer
SongFilmPath varchar(200)
Image Blob(16)
ListensToSong
FavouriteSong
Composed By
Produced By
Composed By
Composed By
Has Mood
Composed By
English_Songs
PK SongID Integer
SongName varchar(100)
FK ArtistID Integer
ArtistName varchar(20)
Lyrics varchar(100000)
ReleaseDate Date
Rating Integer
TotalReviews Integer
FK GenreID varchar(100)
FK MoodID varchar(100)
FK FilmID Integer
SongFilmPath varchar(200)
Image Blob(16)
Punjabi_Songs
PK SongID Integer
SongName varchar(100)
FK ArtistID Integer
ArtistName varchar(20)
Lyrics varchar(100000)
ReleaseDate Date
Rating Integer
TotalReviews Integer
FK GenreID varchar(100)
FK MoodID varchar(100)
FK FilmID Integer
SongFilmPath varchar(200)
Image Blob(16)
Gujrati_Songs
PK SongID Integer
SongName varchar(100)
FK ArtistID Integer
ArtistName varchar(20)
Lyrics varchar(100000)
ReleaseDate Date
Rating Integer
TotalReviews Integer
FK GenreID varchar(100)
FK MoodID varchar(100)
FK FilmID Integer
SongFilmPath varchar(200)
Image Blob(16)
Produced By
Produced By
Produced By
Korean_Songs
PK SongID Integer
SongName varchar(100)
FK ArtistID Integer
ArtistName varchar(20)
Lyrics varchar(100000)
ReleaseDate Date
Rating Integer
TotalReviews Integer
FK GenreID varchar(100)
FK MoodID varchar(100)
FK FilmID Integer
SongFilmPath varchar(200)
Image Blob(16)
Has Genre
Has Mood
Has Mood
Has Genre
Has Genre
Has Mood
Has Mood
Has Genre
Has Genre
FavouriteSong
Has Genre
FavouriteSong
ListensToSong
Genre
PK GenreID Integer
GenreName varchar(100)
TotalSongs Integer
Mood
PK MoodID Integer
MoodName varchar(100)
TotalSongs Integer
FavouriteSong
FavouriteSong

Individual contributions:

Akshyta: Came up with the database scheme, Primary key-Foreign key relationships and referential integrity constraints for all the tables. Worked upon constructing and populating the User and Hindi_Songs tables. Wrote dynamic SQL queries (backend) for fetching data as per user inputs and worked on creation of GUI (frontend) of the Application in JavaFx along with Harshit and its integration with backend.

Harshit: Based on the different purposes for which the Application will be used by the Artist and the FilmMakers, he came up with the scheme, Primary key-Foreign key relationships, referential integrity constraints for the Artist table, FilmMakers table, English songs table, Korean songs table and the 1st draft of our Entity-Relationship Diagram. Populated the english songs table. Wrote the code for GUI along with Akshyta in JavaFX for the “Musify” application and integrated it with the database.

Mohit: He came up with use cases and the different purposes for which the 4 different types of stakeholders- Listener(user), Artist, Administrator, and Filmmaker, will be using our Application. He will assist Akshyta and Harshit in constructing these tables. Populated the gujrati songs table, lyrics table with a desired poster picture to display. Designed the final E-R diagram with Anil.

Anil: He assisted in writing purposes for each stakeholder that our application will be used for. Populated Korean song table, lyrics table. Designed the final draft of ER- Diagram with Mohit. Assisted in writing relational algebraic queries.

Vipin: Came up with a name for the Application.Also, worked in populating the database especially the song tables namely, Gujrati and Punjabi along with the required mp3 files and lyrical text files for the songs with a desired poster picture to display.Also, made the indexing criteria for the database along with the queries for the same.Also, added some relational algebric queries for the corresponding SQL queries.