B.tech. ICT(semester-4)

Course - DBMS (DataBase Management System)

Project - Music Management system

1741054 - Ajay Bechara

1741063 - Akshay Gopani

1741077 - Jishant Patel

1. Description of Project

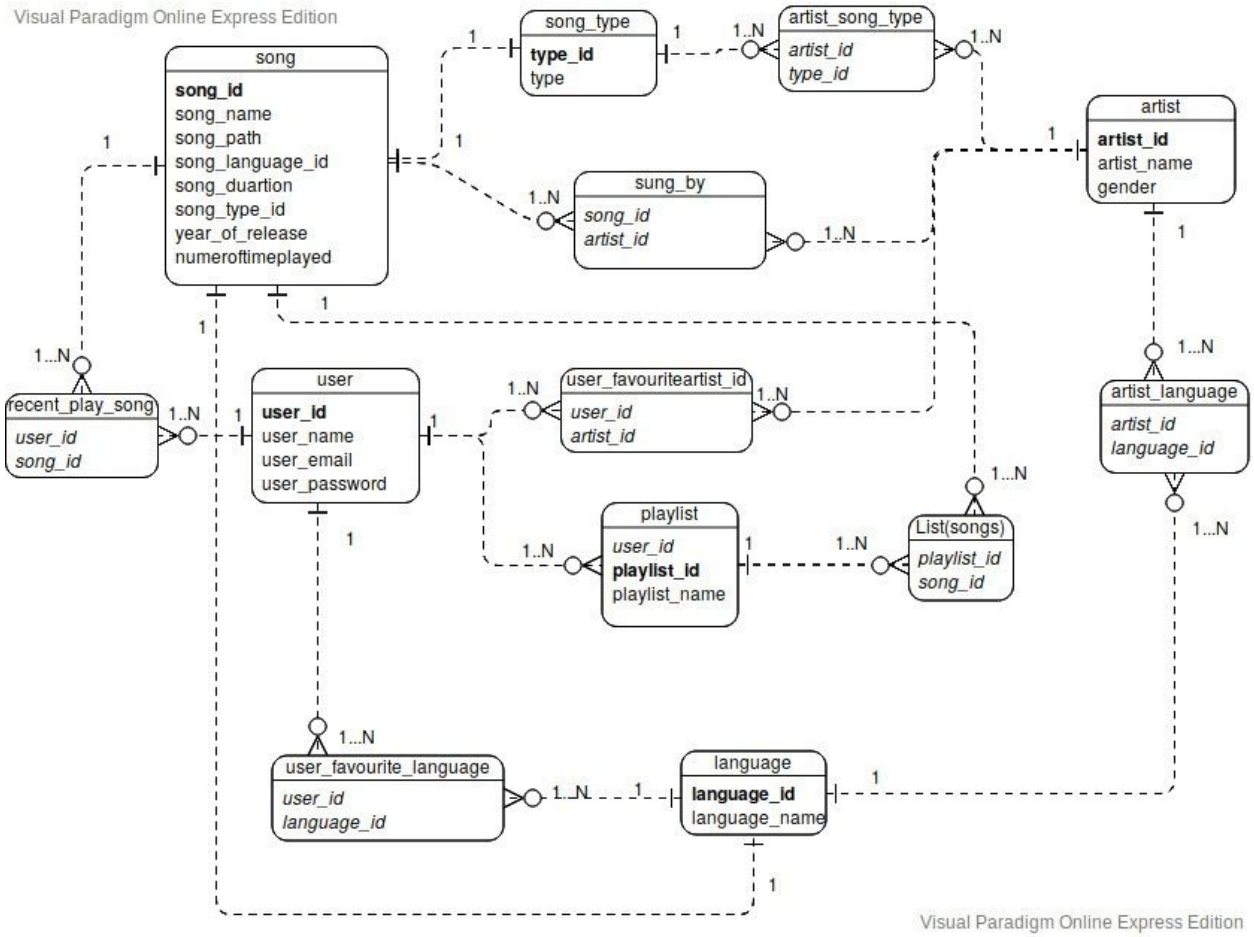
What is it about?

The Music Database project is to categorize and catalog every single piece of music. This system provide music playing facility.user can create playlist and add song into playlist.we store 5 recent play song of every user and display it whenever user signup into our system.we display most favourite artist.when user click on artist name we display the list of all song sung by that artist.we display new release song albums based on song type.so our system provide complete music player type of system to user.

Why we chose it?

Everyone loves Music.The idea of music database arose out of common interest of project members in Music. The concept seemed different and very interesting right in the first go.

2. Entity-Relationship Diagram



3. Table Design

Table name : **Song**
Table description : This table will store the details of Songs.
Table Fields :

1. Song_name: varchar (50) ,**Primary Key**
2. Song_id : int 11
3. Song_type_id : int 11
4. Song_path : varchar(150)
5. Song_language_id : int (11)
6. Song_duration : time
7. Year_of_release : year(4)
8. Numberoftimeplayed : int (11)

Table name :**Artist**
Table description : This table will store the details of Artist.
Table Fields :

1. Artist_id : int (11) ,**Primary Key**
2. Artist_name : varchar(30)
3. gender : varchar(6)

Table name : **User**
Table description : This table will store the details of Users.
Table Fields :

1. User_name: varchar(30)
2. User_id : varchar(20) , **Primary Key**
3. User_email : varchar(40)
4. user_password : varchar(25)

Table name : **language**
Table description : This table will store the of language name and id.
Table Fields :

1. Language_name : varchar(15)
2. language_id : int (11) ,**Primary Key**

Table name : **Playlist**

Table description : This table will store the playlist and the the user related to it.
Table Fields :

1. User_id : varchar(20) ,**Foreign Key from user table**
2. Playlist_id : int (11) ,**Primary Key**
3. playlist_name : varchar(20)

Table name :**Song type**
Table description : This table will store the types of music eg. classical, rock ,vocals.
Table Fields :

1. Type_id : int (11) ,**Primary Key**
2. type : varchar(20)

Table name : **Sung_by**
Table description : This table will show the song and its artist.
Table Fields :

1. Song_id : int(11) **Foreign Key from song table**
2. artist_id : int(11) **Foreign Key from artist table**

Table name : **artist_song_type**
Table description : This table shwos what kind of songs artist play.
Table Fields :

1. Artist_id :int (11) **Foreign Key from artist table**
2. Type_id : int (11) **Foreign Key from song_type table**

Table name : **Artist_language**
Table description : This table stores artist and song he/she sung in that language.
Table Fields :

1. Artist_id : int (11) **Foreign Key from artist table**
2. Language_id : int (11) **Foreign Key from language table**

Table name : **User_favioraiteartist_id**
Table description :This table will store the favorite artist of users.
Table Fields :

1. User_id : varchar(20) **Foreign Key from user table**
2. Artist_id : int (11) **Foreign Key from artist table**

Table name : **List(songs)**
Table description : This table will store the playlist and song it contains.
Table Fields :

1. Playlist_id : int (11) ,**Primary Key**
2. song_id : int (11) **Foreign Key from song table**

Table name : **recent_play_song**
Table description : This table will store song users have recently played.
Table Fields :

1. User_id : varchar(20) **Foreign Key from user table**
2. Song_id : int(11) **Foreign Key from song table**

Table name : **user_favoritelanguage**
Table description : This table will store user and their favorite languages.
Table Fields :

1. User_id : varchar(20) **Foreign Key from user table**
2. Language_id : int(11) **Foreign Key from language table**

4.Stored Procedure, Functions and Triggers

```
1.Procedure to take new user input data


DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `newuserinput`(IN
`userid` VARCHAR(20), IN `username` VARCHAR(30), IN `useremail`
VARCHAR(40), IN `userpass` VARCHAR(25), IN
`userfavouritelanguage` VARCHAR(15))

    NO SQL

BEGIN

INSERT into USER (user_id,user_name,user_email,user_password)
VALUES (userid,username,useremail,userpass);
```

```
CALL userfavouritelanguage(userid,userfavouritelanguage);

end$$

DELIMITER ;
```

2.Procedure for Insert Data into recentplaysong table

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`addintorecentplaysong`(IN `userid` VARCHAR(20), IN `songname`
VARCHAR(50))
    NO SQL
BEGIN
DECLARE sng int;
DECLARE cnt int;
SELECT COUNT(recentplay_song.user_id) into cnt from
recentplay_song;
SELECT song.song_id into sng FROM song WHERE
song.song_name=songname;
if cnt >=5 then
DELETE from recentplay_song WHERE recentplay_song.user_id=userid
LIMIT 1;
INSERT INTO recentplay_song VALUES (userid,sng);
ELSE
INSERT INTO recentplay_song VALUES (userid,sng);
end IF;
END$$
DELIMITER ;
```

3.Procedure for display artist wise songlist

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`diplayartistsonglist`(IN `artistname` VARCHAR(30))
    NO SQL
BEGIN
declare artistid int;
SELECT artist.artist_id into artistid from artist where
artist.artist_name=artistname;
select artistid;
select song.song_name
from song
```

```
inner JOIN
artist_song  ON song.song_id=artist_song.song_id
where artist_song.artist_id=artistid;
end$$
DELIMITER ;
```

4.Procedure for add song into playlist

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`addsongintoplaylist`(IN `userid` VARCHAR(20), IN `playlistname`
VARCHAR(20), IN `songname` VARCHAR(50))

    NO SQL

BEGIN

DECLARE pid int;

DECLARE sng int;

SELECT playlist.playlist_id INTO pid FROM playlist WHERE
playlist.playlist_name=playlistname and playlist.user_id=userid;

SELECT song.song_id into sng FROM song WHERE
song.song_name=songname;

INSERT INTO list VALUES (pid,sng);

END$$

DELIMITER ;
```

5.Procedure for display top 3 artist

```
DELIMITER $$
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`mostfavouriteartist`()

    NO SQL

BEGIN

SELECT artist.artist_name from user_favourite_artist,artist
where artist.artist_id=user_favourite_artist.artist_id  GROUP by
user_favourite_artist.artist_id ORDER  by
COUNT(user_favourite_artist.artist_id) desc LIMIT 3;

END$$

DELIMITER ;
```

6. Procedure for Display Playlist

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`playlistnamelist`(IN `userid` VARCHAR(20))

    NO SQL

BEGIN

SELECT playlist_name from playlist where
playlist.user_id=userid;

END$$

DELIMITER ;
```

7.Procedure to display random song type

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`displayrandomsongtype`()

    NO SQL

BEGIN

SELECT song_type.type from song_type GROUP by rand() desc LIMIT
5;

END$$

DELIMITER ;
```

8.Procedure to diplay type wise newrelease song;

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`displaytypewisenewrealessong`(IN `type` VARCHAR(20))

    NO SQL

BEGIN

SELECT song.song_name

from song

WHERE song.song_type_id=(SELECT song_type.type_id from

song_type WHERE song_type.type=type)

and song.year_of_release=YEAR(CURRENT_DATE) ORDER by rand() desc
LIMIT 5;

END$$
```

```
DELIMITER ;
```

Triggers :

1.Artistlanguageinputvalidity

```
CREATE TRIGGER `artistlanguageinputvalidity` BEFORE INSERT ON
`artist_language`

 FOR EACH ROW BEGIN

DECLARE msg varchar(60);

if EXISTS(SELECT
artist_language.artist_id,artist_language.language_id from
artist_language where artist_language.artist_id=new.artist_id
and artist_language.language_id=new.language_id) THEN

 set msg = 'Error: Entry already Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

   if NOT EXISTS (SELECT artist.artist_id from artist WHERE
artist.artist_id=new.artist_id) THEN

     set msg = 'Error: Artist not Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

  if NOT EXISTS (SELECT language.language_id from language WHERE
language.language_id=new.language_id) then

      set msg = 'Error: Language not Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

END
```

2.ArtistsongInputValidity

```
BEGIN

DECLARE msg varchar(60);

if EXISTS(SELECT artist_song.artist_id,artist_song.song_id from
artist_song where artist_song.artist_id=new.artist_id and
artist_song.song_id=new.song_id) THEN

 set msg = 'Error: Entry already Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

if NOT EXISTS (select artist.artist_id from artist WHERE
artist.artist_id=new.artist_id) then

set msg = 'Error: artist not Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

  if NOT EXISTS (select song.song_id from song WHERE
song.song_id=new.song_id) then set msg = 'Error: song not
Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

END
```

3 Artistsongtypeinputvalidity

```
BEGIN

DECLARE msg varchar(60);
```

```
if EXISTS(SELECT
artist_song_type.artist_id,artist_song_type.type_id from
artist_song_type where artist_song_type.artist_id=new.artist_id
and artist_song_type.type_id=new.type_id) THEN

 set msg = 'Error: Entry already Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

if NOT EXISTS(SELECT artist.artist_id from artist WHERE
artist.artist_id=new.artist_id) THEN

 set msg = 'Error: Artist not Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

  if NOT EXISTS(SELECT song_type.type_id from song_type where
song_type.type_id=new.type_id) THEN

  set msg = 'Error: Song Type not Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

END
```

4. Listinputvalidity

```
BEGIN

DECLARE msg varchar(60);

if EXISTS(SELECT list.playlist_id,list.song_id from list where
list.playlist_id=new.playlist_id and list.song_id=new.song_id)
THEN

 set msg = 'Error: Entry already Exist';
```

```
    SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

  if NOT EXISTS (SELECT song.song_id from song WHERE
song.song_id=new.song_id) THEN

   set msg = 'Error: Song not Exist';

    SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

  if NOT EXISTS (SELECT playlist.playlist_id from playlist WHERE
playlist.playlist_id=new.playlist_id) THEN

   set msg = 'Error: Playlist not Exist';

    SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;



END
```

5. Playlistnamevalidity

```
CREATE TRIGGER `playlistnamevalidity` BEFORE INSERT ON
`playlist`

 FOR EACH ROW BEGIN

DECLARE msg varchar(100);

if EXISTS(SELECT playlist_name from playlist where
playlist.playlist_name=new.playlist_name and
new.user_id=playlist.user_id) THEN

      set msg = 'Error: playlist name already exist';
```

```
      SIGNAL SQLSTATE '45001' set message_text = msg;
      end IF;
END
```

6 UserValiditaion

```
CREATE TRIGGER `uservalidation` BEFORE INSERT ON `user`
 FOR EACH ROW BEGIN
declare msg varchar(100);
     if EXISTS(SELECT user_email FROM User WHERE user_email =
NEW.user_email) or (NEW.user_email NOT LIKE '%_@%_.__%') then
         set msg = 'Error: Enter valid email id';
      SIGNAL SQLSTATE '45001' set message_text = msg;
      ELSEIF char_length(new.user_password)<6 then
       set msg = 'Error: password length must be greater then
6';
      SIGNAL SQLSTATE '45001' set message_text = msg;
      end if;
end
```

7 Userfavouroteartist validity

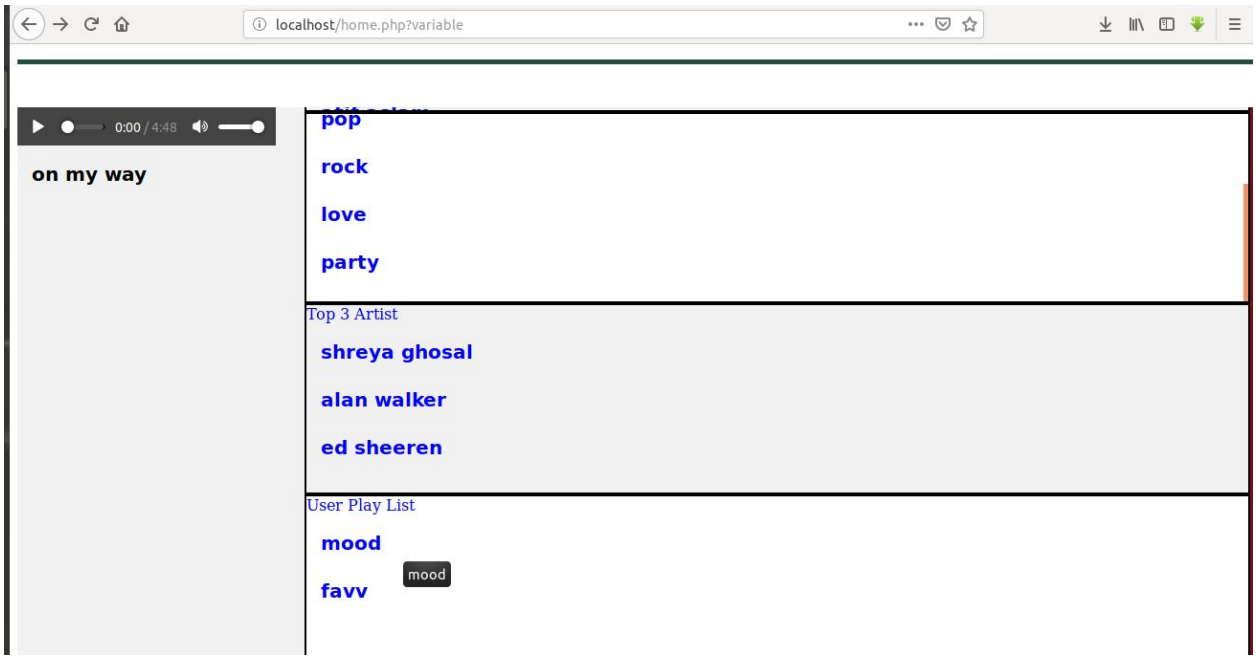
```
CREATE TRIGGER `userfavouriteartistvalidity` BEFORE INSERT ON
`user_favourite_artist`
 FOR EACH ROW BEGIN
DECLARE msg varchar(60);
```

```
if EXISTS(SELECT
user_favourite_artist.user_id,user_favourite_artist.artist_id
from user_favourite_artist where
user_favourite_artist.user_id=new.user_id and
user_favourite_artist.artist_id=new.artist_id) THEN

 set msg = 'Error: Entry already Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

END
```

8 Userfaavouritelanguagevalidity

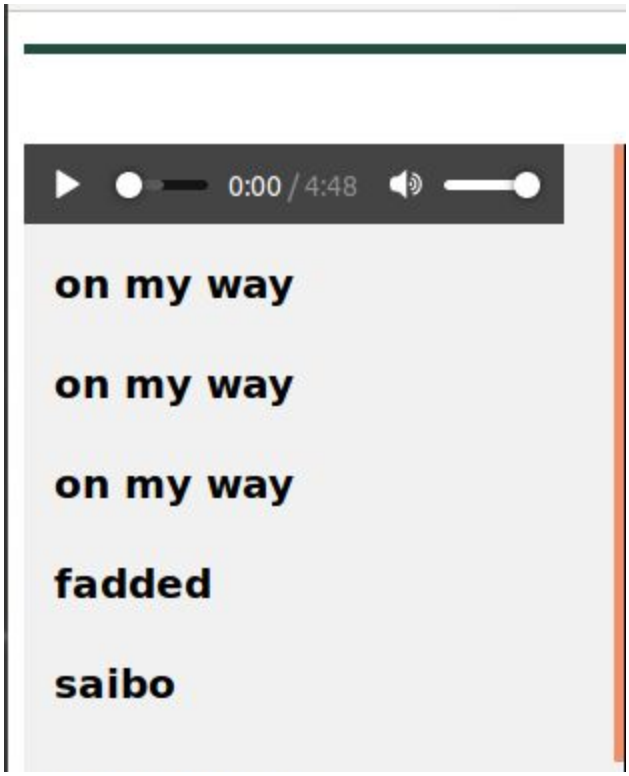
```
CREATE TRIGGER `userfavouritelanguagevalidity` BEFORE INSERT ON
`user_favourite_language`

 FOR EACH ROW BEGIN

DECLARE msg varchar(60);

if EXISTS(SELECT
user_favourite_language.user_id,user_favourite_language.language
_id from user_favourite_language where
user_favourite_language.user_id=new.user_id and
user_favourite_language.language_id=new.language_id) THEN

 set msg = 'Error: Entry already Exist';

     SIGNAL SQLSTATE '45001' set message_text = msg;

  end if;

END
```

5. Screenshots of results generated after procedure and function are called



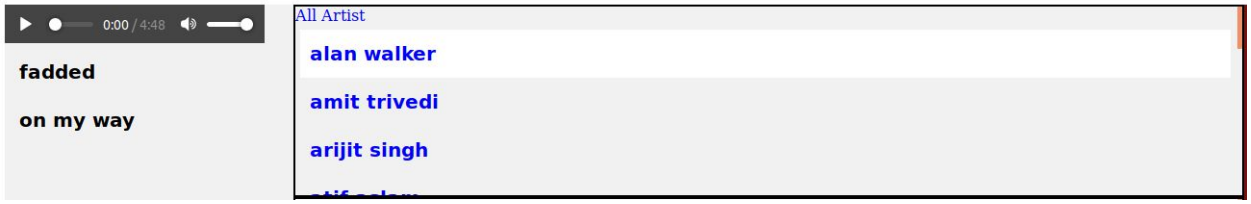
Select playlist (mood)

- **getListByPlayListName**



Recent Play List

- **displayRecentPlaySong()** called

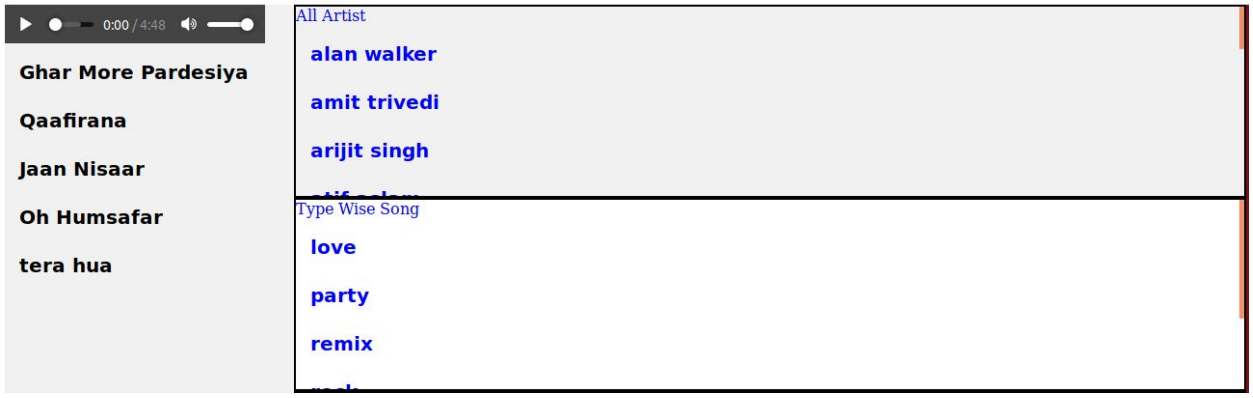


Song of selected Artist

- **diplayArtistSongList**

Top 3 Artist

- **mostFavouriteArtist**



Selected Type Song

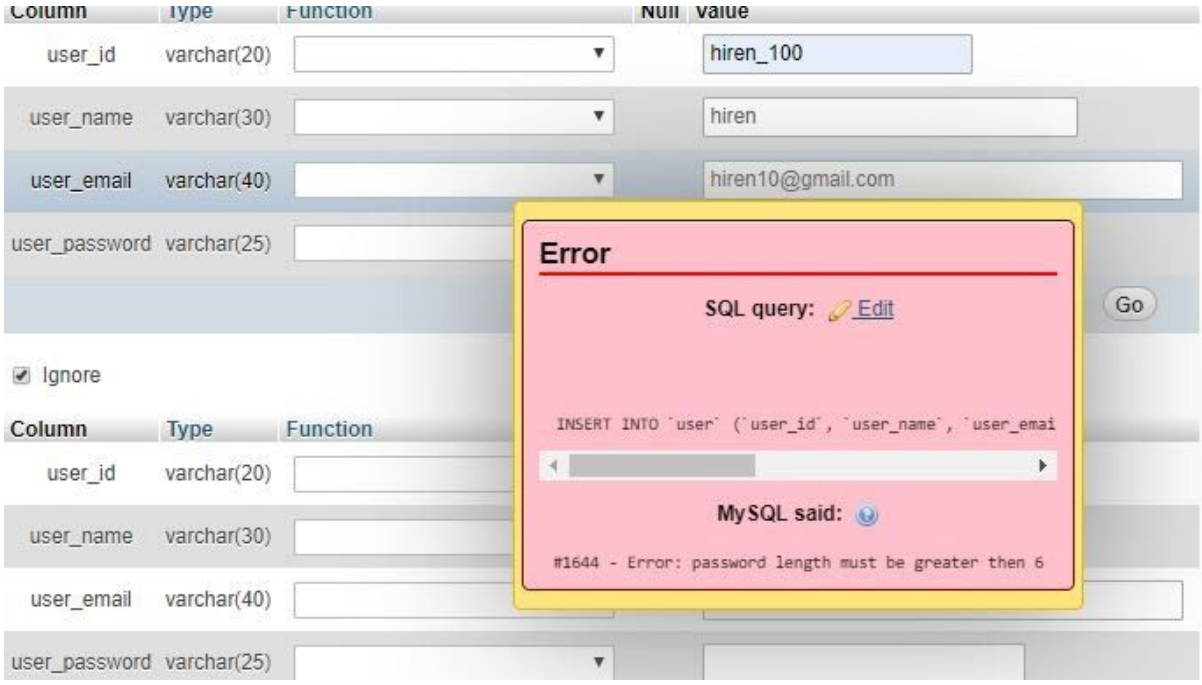
- **displayTypeWiseNewRealesSong**

This procedures are called when page is loded :

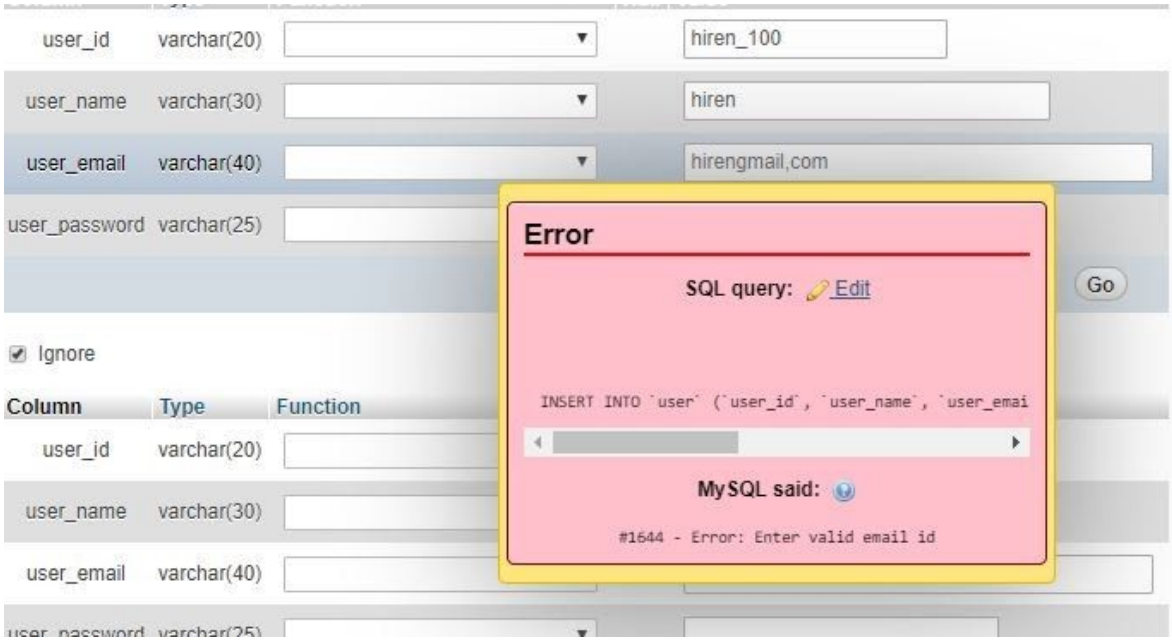
- **getArtistList**
- **displayrandomsongtype**
- **mostfavouriteartist**
- **Playlistnamelist**

6. Screenshots of errors generated on front-end when trigger is violated

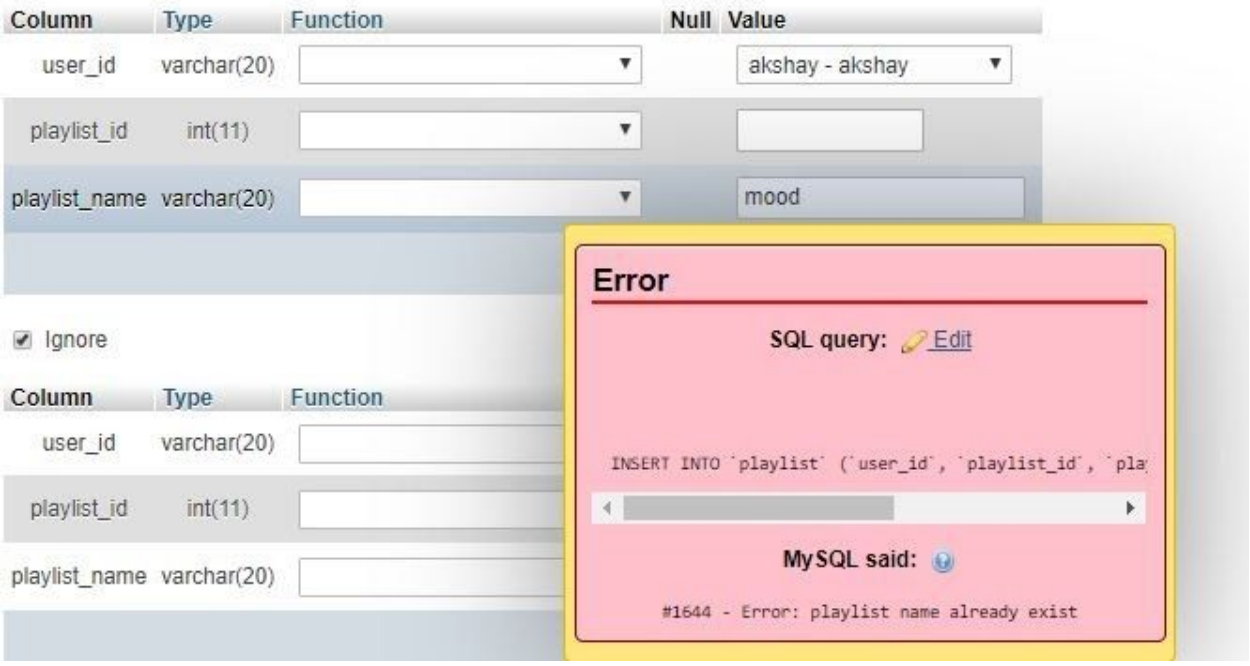
Here New **Uservalidation trigger** violated.here you can see that user enter a invalid email id hence it was violated.



Here New **Uservalidation** trigger violated.here you can see that user enter a invalid password length hence it was violated.



Here **playlistname** trigger : One user can not have playlist with the same name



Leastinputvalid trigger: trigger violats when entry already exist.