Music Share

A project in

Principles of Database Systems

By

Gopikrishna Sathyamurthy
N14794536
gs1922@nyu.edu

Project Overview

The objective of this project is to create a system where a user can create a profile of his self and follow one or more music bands and to be up to date about their concerts. The user can reserve for a concert, attend them and, rate and write a review about it. The user can also post recommendations for other users who are following him. And he can create list of favorite concerts and share with others.

A user can be an artist in a band and he can post the bands concerts on the system for others to follow. A posted concert will be withheld for approval by other eligible users if the posting user is not eligible enough. An artist can also be part of more than one band. A band follows a particular genre; however they can conduct concerts on different genres and subgenres.

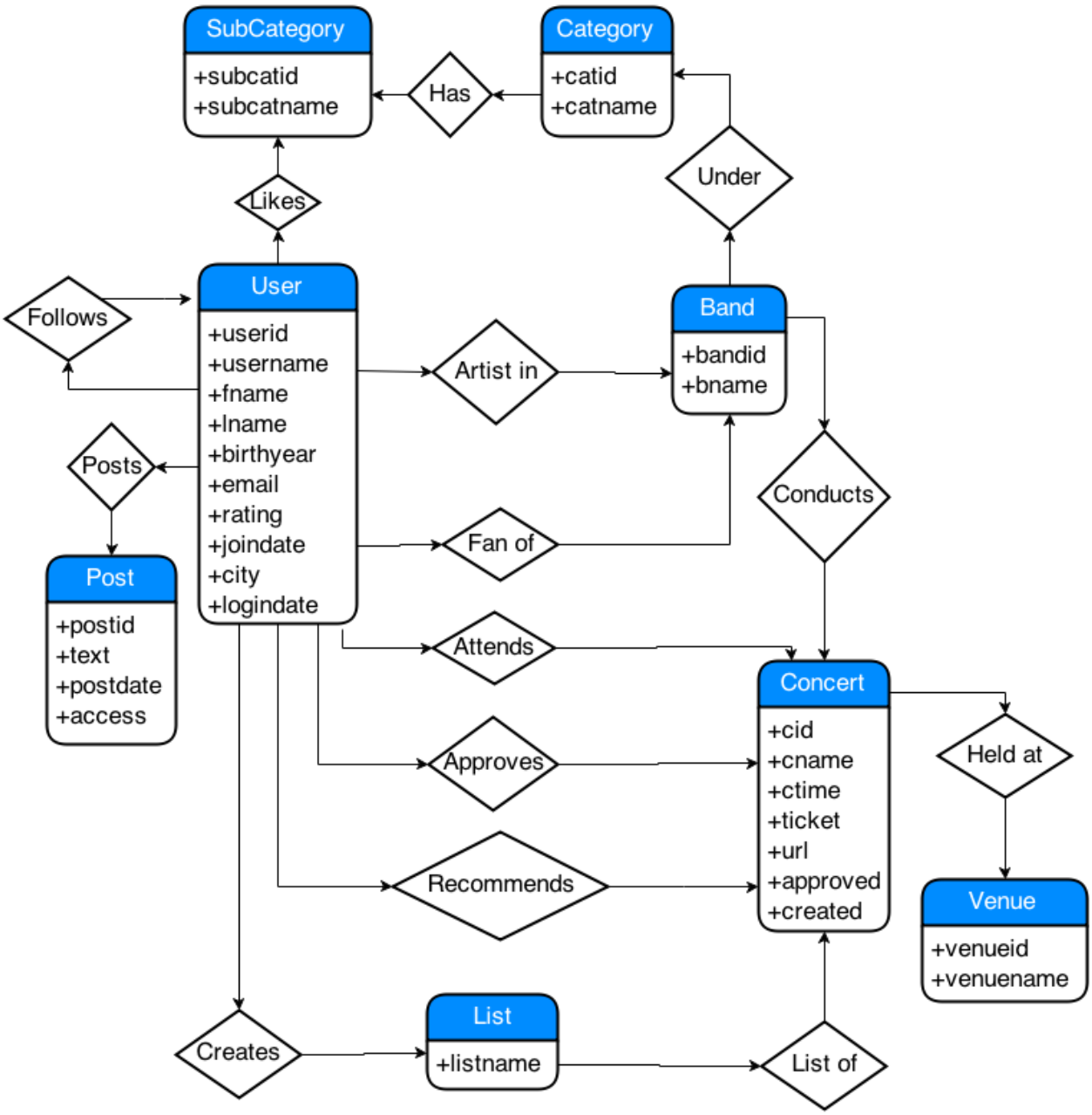
The system helps a user to find his favorite music by providing recommendations based on previous activities of the user and selected tastes and, rating and reviews on the concerts. The system gives the user a search tool for the user to find bands on his own.

These are the overall concepts that are involved in this project.

Approach

By analyzing the criteria, the main entities of the project are the User, Artist, Band and Concert, and other entities that finish up the criteria are List of Concerts by User, Venues, Posts by User, and Genres and Subgenres.

The relationships between the entities can be understood using the following Entity-Relationship Diagram.


SubCategory
+subcatid
+subcatname
Has
Category
+catid
+catname
Under
Likes
Follows
User
+userid
+username
+fname
+lname
+birthyear
+email
+rating
+joindate
+city
+logindate
Artist in
Band
+bandid
+bname
Posts
Conducts
Fan of
Post
+postid
+text
+postdate
+access
Attends
Concert
+cid
+cname
+ctime
+ticket
+url
+approved
+created
Approves
Held at
Recommends
Venue
+venueid
+venuename
Creates
List
+listname
List of

ER-Diagram

Assumptions

There are few assumptions made here in the schema. A band follows only a single genre; however the band may conduct concerts on different genres too. In Attend table, a record exists only if user RSVPs for the concert.

Tables and Constraints

From the ER-Diagram, the following tables are derived.

$$User\left(\begin{matrix}\boldsymbol{userid}, \boldsymbol{username}, password, \\ fname, lname, birthyear, email, rating, joindate, city, logindate\end{matrix}\right)$$

$User$ table stores the user's profile values. Here $userid$ is the primary key and $username$ is an alternate key for the table.

$$Band(\boldsymbol{bandid}, bname, \boldsymbol{catid})$$

$Band$ table stores list of bands. $bandid$ is the primary key and $catid$ is foreign key from $Category$ with $set\ null$ constraint on delete.

$$Concert(\boldsymbol{cid}, \boldsymbol{userid}, \boldsymbol{bandid}, cname, \boldsymbol{venueid}, ctime, ticket, url, approved, created)$$

$Concert$ table stores all the concerts that is posted on the application. cid is the primary key while $userid, bandid, venueid$ are foreign keys from $User, Band, Venue$ respectively. $venueid$ is with $set\ null$ constraint on delete.

When new concert is created, two actions are triggered before and after its creation: a $ConcertInsertBefore$ that calls $UpdateUserRating$ procedure to recalculate the posting user's rating and a $ConcertInsertAfter$ that decides whether to insert a record into $Approve$ table based on its $approved$ attribute.

When a concert is deleted a $ConcertDeleteAfter$ is triggered that redeems the rating for posting user for unapproved concert and reduces few points from the user's rating for approved concerts.

$Approve(\boldsymbol{cid}, \boldsymbol{userid1}, \boldsymbol{userid2})$

A concert is straight away approved if the posting user has a rating of 8 or more, otherwise the concert is inserted here in $Approve$ table for approval from two users $userid1$ and $userid2$, whose ratings at least 7. When both are approved, a $ConcertApproveAfter$ is triggered, that toggles $approved$ of concert and calls $UpdateUserRating$.

$Artist(\boldsymbol{userid}, \boldsymbol{bandid}, url)$

$Artist$ is a user in a band. Thus, $userid$ and $bandid$ are foreign key, both with $set\ null$ constraint on delete.

$Category(\boldsymbol{catid}, catname)$

It stores a list of genres for the application.

$SubCategory(\boldsymbol{subcatid}, \boldsymbol{catid}, subcatname)$

It stores a list of subgenres for the application.

$ConcertGenre(\boldsymbol{cid}, \boldsymbol{subcatid})$

Although a band is dedicated to only one genre, it's concert may be conducted in many subgenres. $ConcertGenre$ associates the concert with subgenres through foreign keys cid and $subcatid$.

$Venue(\boldsymbol{venueid}, venuename)$

This is a list predefined venues by the company.

$Post(\boldsymbol{postid}, \boldsymbol{userid}, text, postdate, access)$

This contains list of posts by user on his profile with $postid$ as primary key and $userid$ as foreign key with $cascade$ constraint.

$UserList(\boldsymbol{listid}, \boldsymbol{userid}, listname, listdate)$

This maintains the concert lists created by the user.

$List(\boldsymbol{listid}, \boldsymbol{cid})$

This table helps the application to associate concerts with $userlists$ with both $listid$ and cid as foreign keys from $List$ and $Concert$ respectively.

$Follow(\boldsymbol{userid}, \boldsymbol{followerid}, followdate)$

This table maintains the records of who is following who. Both $userid$ and $followerid$ are foreign keys from different instances of $User$ table.

$Fan(\boldsymbol{userid}, \boldsymbol{bandid}, fandate)$

This table holds the records of who is a fan of which band. $userid$ and $bandid$ are foreign keys.

$Recommend(\boldsymbol{userid}, \boldsymbol{cid}, recdate)$

Whenever a user recommends a concert to his followers, it gets added here.

$UserGenre(\boldsymbol{userid}, \boldsymbol{subcatid})$

This maintains the user's taste of music. $userid$ and $subcatid$ are foreign keys here.

$Attend(\boldsymbol{userid}, \boldsymbol{cid}, attended)$

If a user is attending a concert, a record is placed here. Once he has attended it he can update the $attended$ column.

$Review(\boldsymbol{reviewid}, \boldsymbol{userid}, \boldsymbol{cid}, rating, review, reviewdate)$

This last table contains all the reviews written by the users with ratings on a given concert. Here $reviewid$ is primary key while $userid$ and cid are foreign keys.

Triggers and Procedures

$ConcertInsertBefore$

This is triggered before a concert is inserted. If the posting user's rating is at least 8, the new row's $approved$ is set to 1 else to 0. And then $UpdateUserRating$ is called.

$ConcertInsertAfter$

This is triggered after the concert is inserted. If the inserted concert has $approved$ as 0 then create a new row in $Approve$ table to facilitate two for two users to approve it.

$ConcertDeleteAfter$

This trigger is called after a concert is deleted just to readjust user's rating by calling $UpdateUserRating$.

$ConcertApproveAfter$

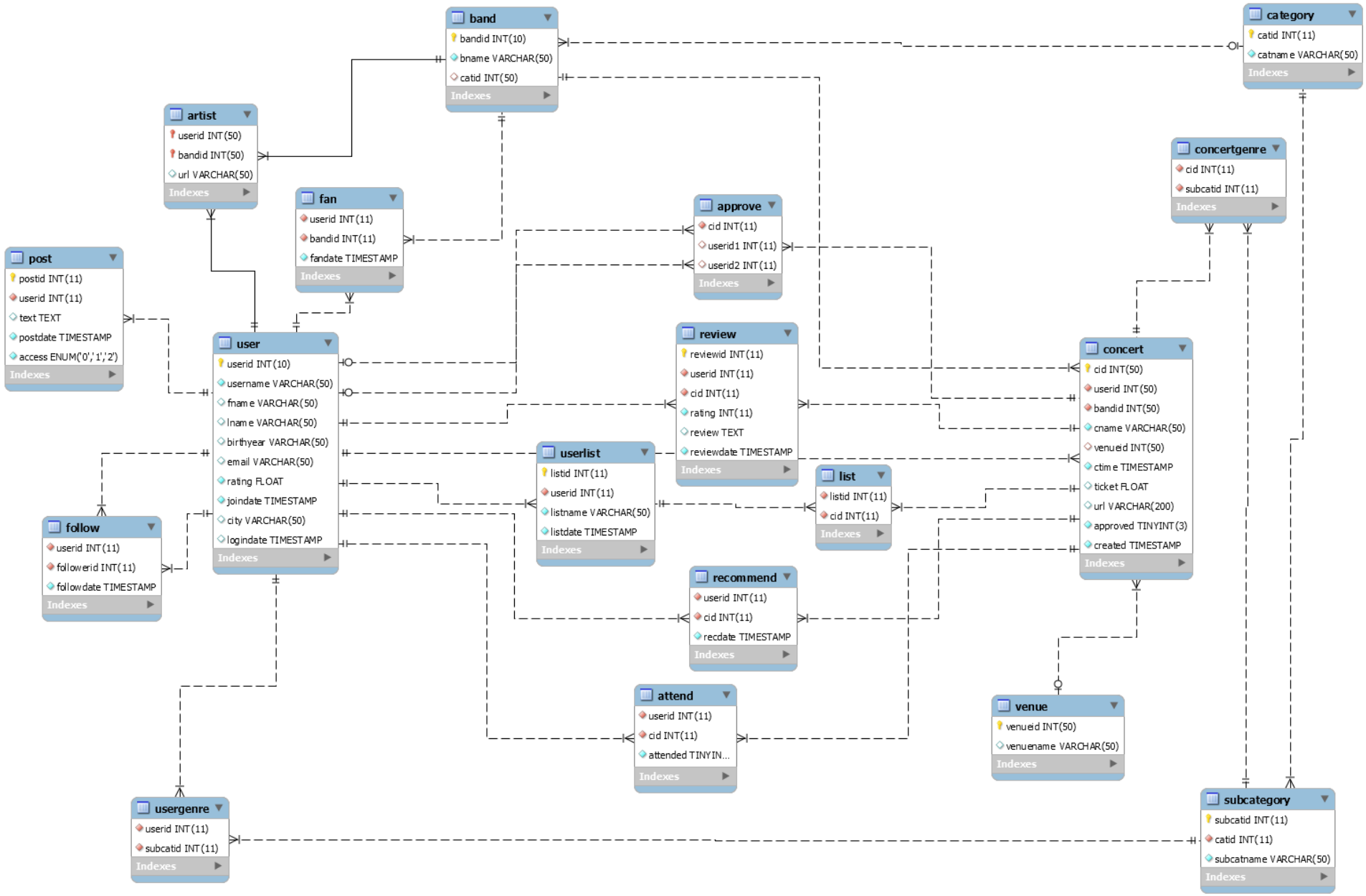
After two users approve concert i.e. two $userid$s are updated into approve table for given concert, the trigger sets $approved$ of $Concert$ to 1. And this also calls $UpdateUserRating$ to recalculate the user's rating.

$UpdateUserRating$

This recalculates the user's rating based on the given parameters i.e. whether to increase or decrease the rating value.

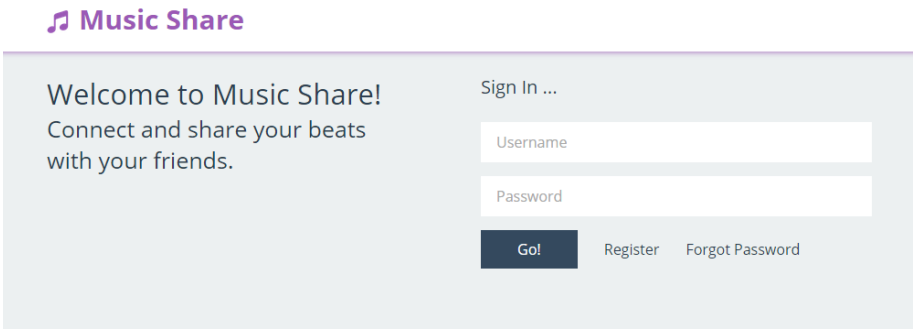
Entity Relationships

The relationships and constraints between tables, procedures and triggers can be illustrated using the following class diagram.

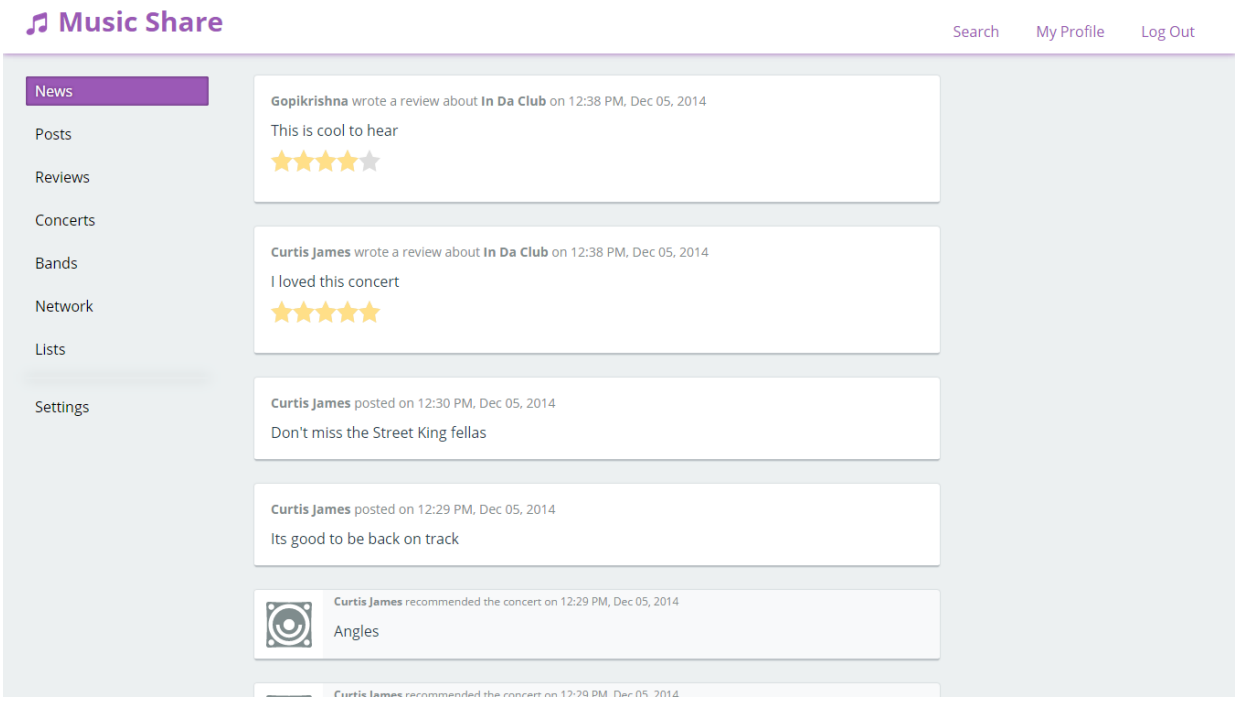

band
bandid INT(10)
bname VARCHAR(50)
catid INT(50)
Indexes
category
catid INT(11)
catname VARCHAR(50)
Indexes
artist
userid INT(50)
bandid INT(50)
url VARCHAR(50)
Indexes
fan
userid INT(11)
bandid INT(11)
fandate TIMESTAMP
Indexes
approve
cid INT(11)
userid1 INT(11)
userid2 INT(11)
Indexes
concertgenre
cid INT(11)
subcatid INT(11)
Indexes
post
postid INT(11)
userid INT(11)
text TEXT
postdate TIMESTAMP
access ENUM('0','1','2')
Indexes
user
userid INT(10)
username VARCHAR(50)
fname VARCHAR(50)
lname VARCHAR(50)
birthyear VARCHAR(50)
email VARCHAR(50)
rating FLOAT
joindate TIMESTAMP
city VARCHAR(50)
logindate TIMESTAMP
Indexes
review
reviewid INT(11)
userid INT(11)
cid INT(11)
rating INT(11)
review TEXT
reviewdate TIMESTAMP
Indexes
concert
cid INT(50)
userid INT(50)
bandid INT(50)
cname VARCHAR(50)
venueid INT(50)
ctime TIMESTAMP
ticket FLOAT
url VARCHAR(200)
approved TINYINT(3)
created TIMESTAMP
Indexes
userlist
listid INT(11)
userid INT(11)
listname VARCHAR(50)
listdate TIMESTAMP
Indexes
list
listid INT(11)
cid INT(11)
Indexes
follow
userid INT(11)
followerid INT(11)
followdate TIMESTAMP
Indexes
recommend
userid INT(11)
cid INT(11)
recdate TIMESTAMP
Indexes
attend
userid INT(11)
cid INT(11)
attended TINYIN...
Indexes
venue
venueid INT(50)
venuename VARCHAR(50)
Indexes
usergenre
userid INT(11)
subcatid INT(11)
Indexes
subcategory
subcatid INT(11)
catid INT(11)
subcatname VARCHAR(50)
Indexes

Control Flow

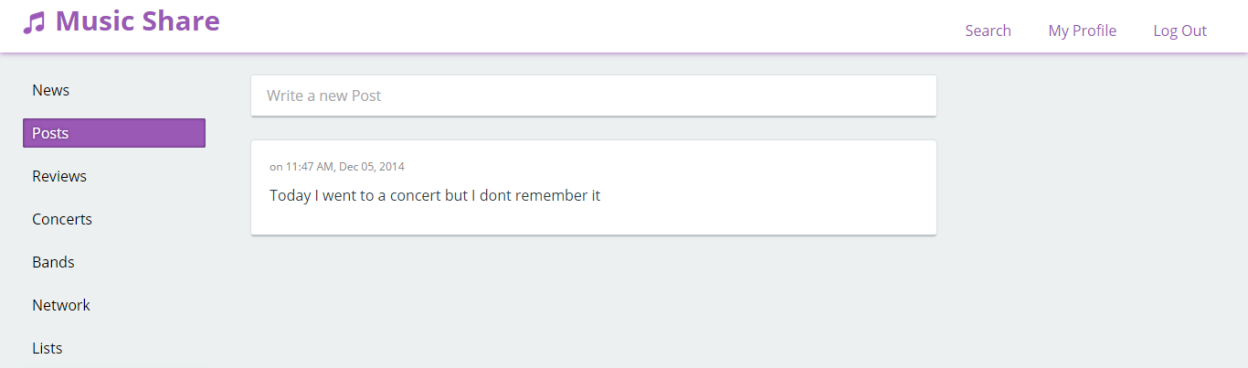
The application starts with a Login page for the user to enter his username and password, or to create a new account by registering.



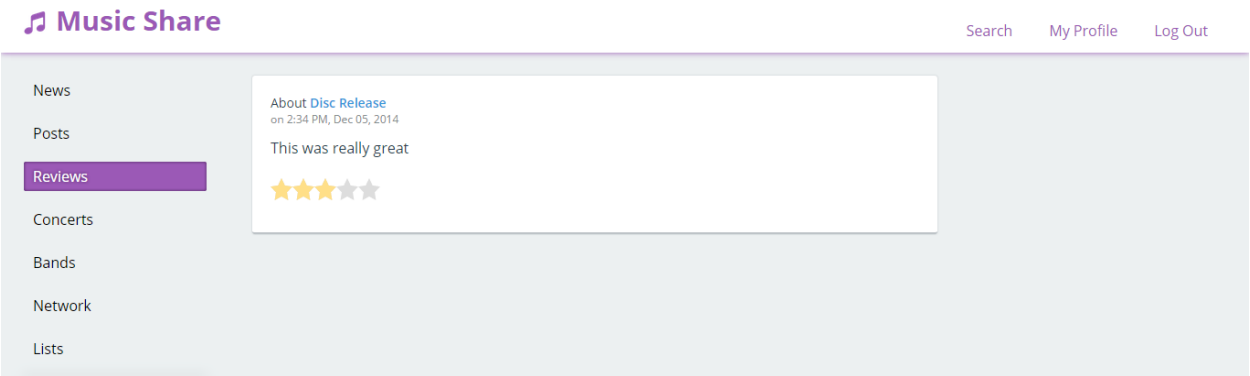
Once the user is logged in, he can see his profile with news of last 24 hours of happenings. If he is a new user, he will not see anything until he has selected some music tastes or followed some users.



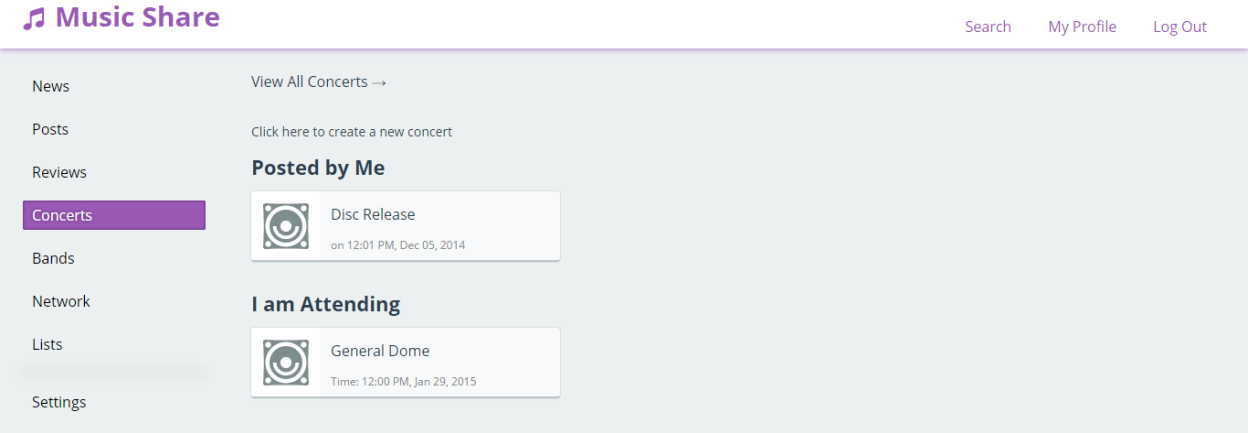
He can create or delete posts from posts tab.



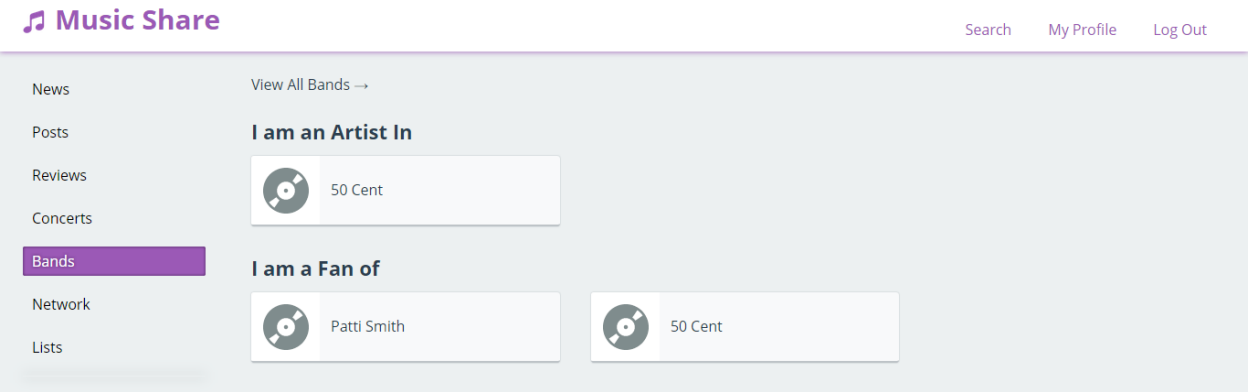
He can manage his reviews from reviews tab. To create a new review, he should go to a concert and write a one.



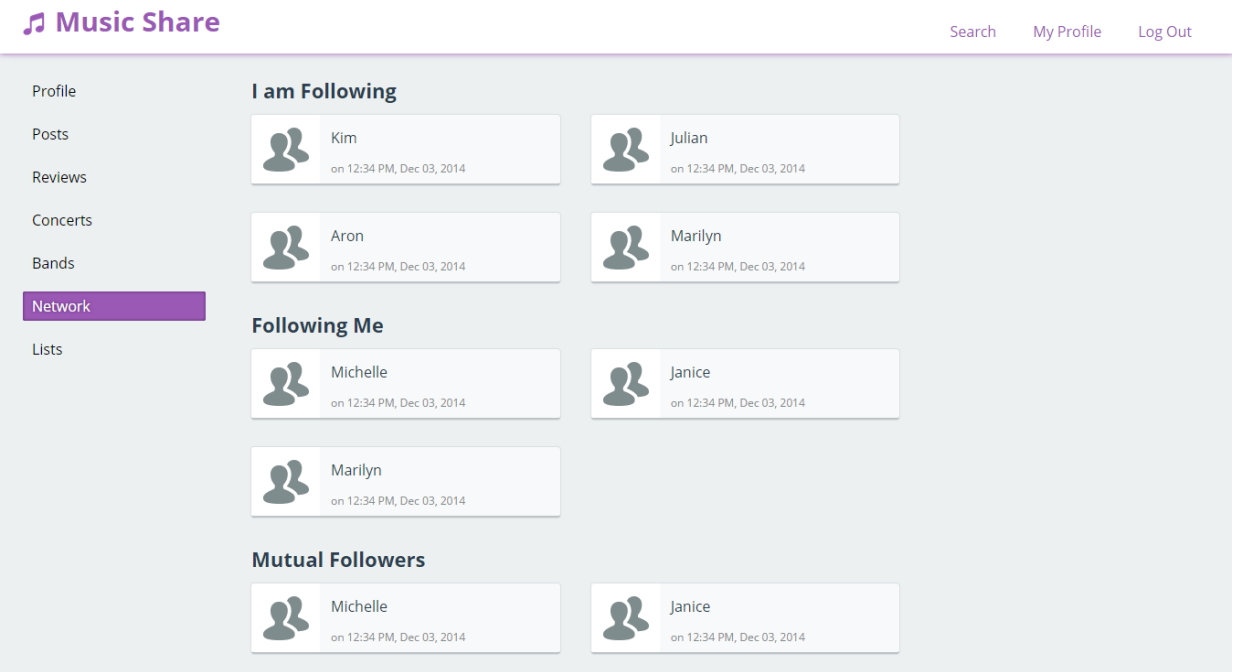
He can create concerts and see which concerts he is attending or have attended from concerts tab.



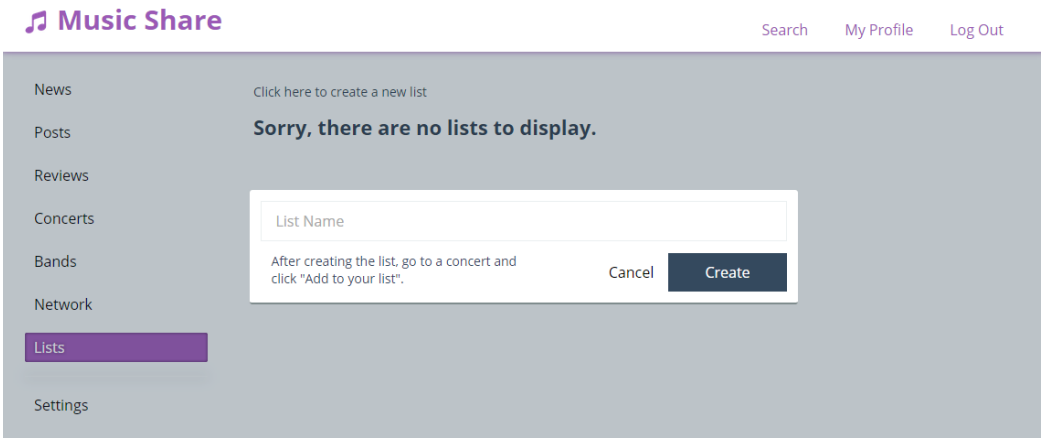
And he can view his bands from bands tab. The band is shown if he is a fan of or an artist in the band.



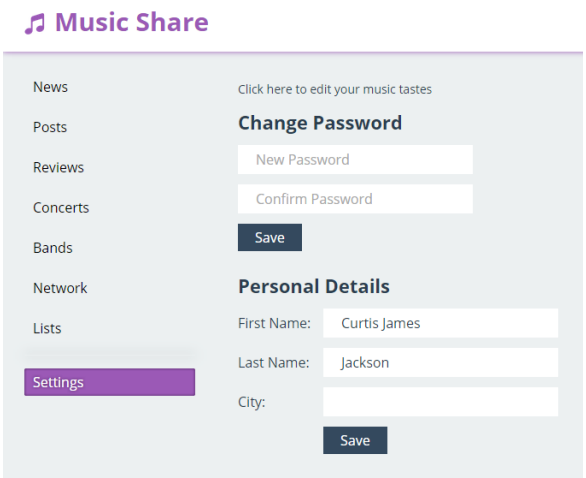
Network tab shows the user, who he is following, and who is following him. If he is viewing another user's profile then he can also see mutual followers.

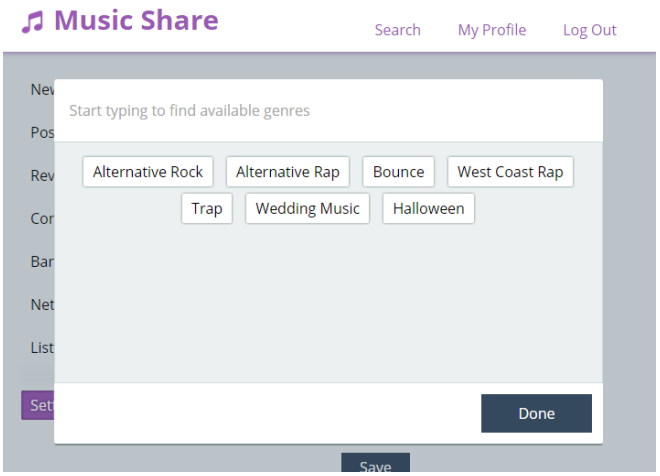


He can create and view concert lists from lists tab.

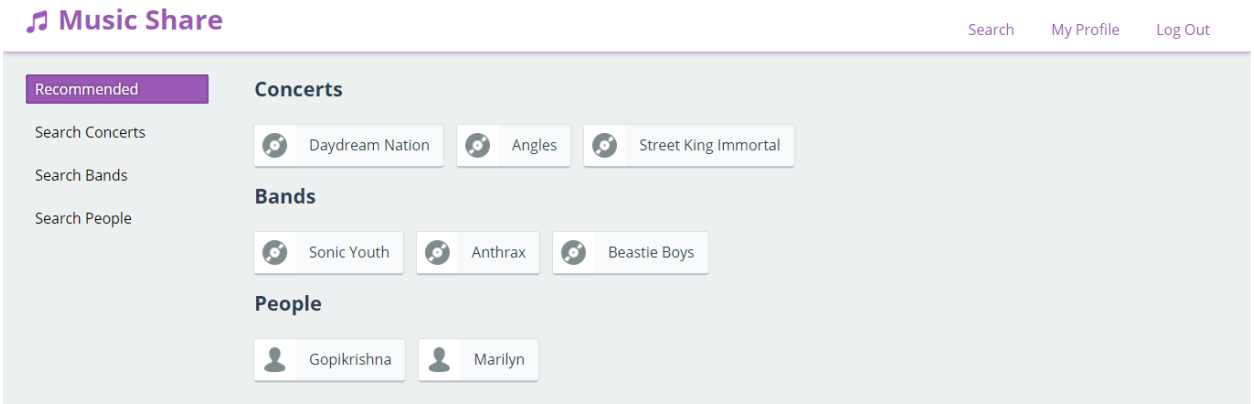


Through settings, the user can change password, set profile details and change his music tastes.

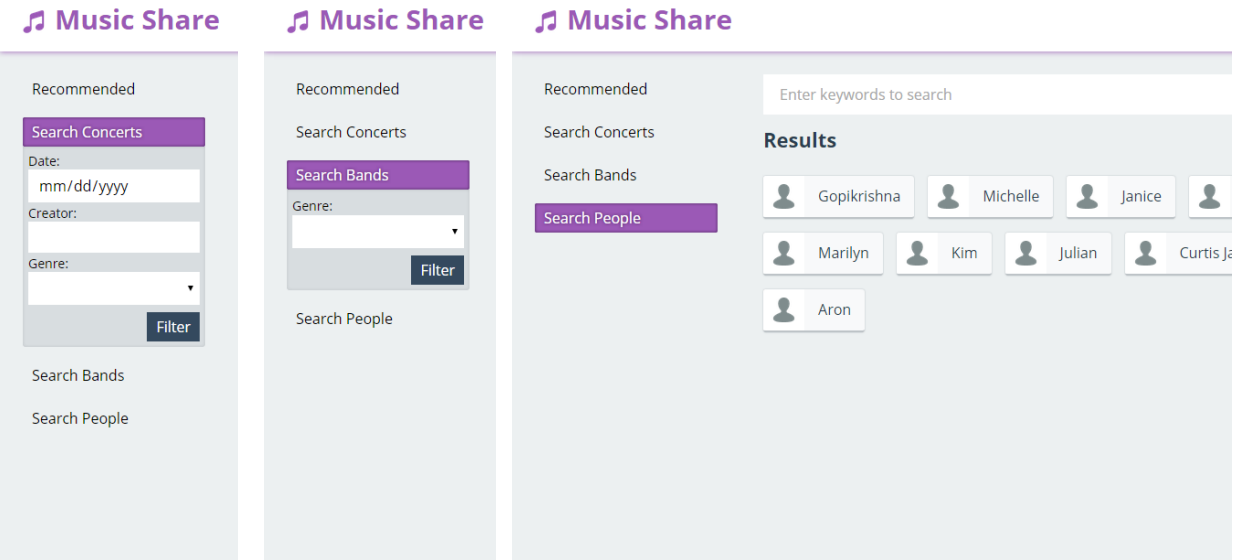




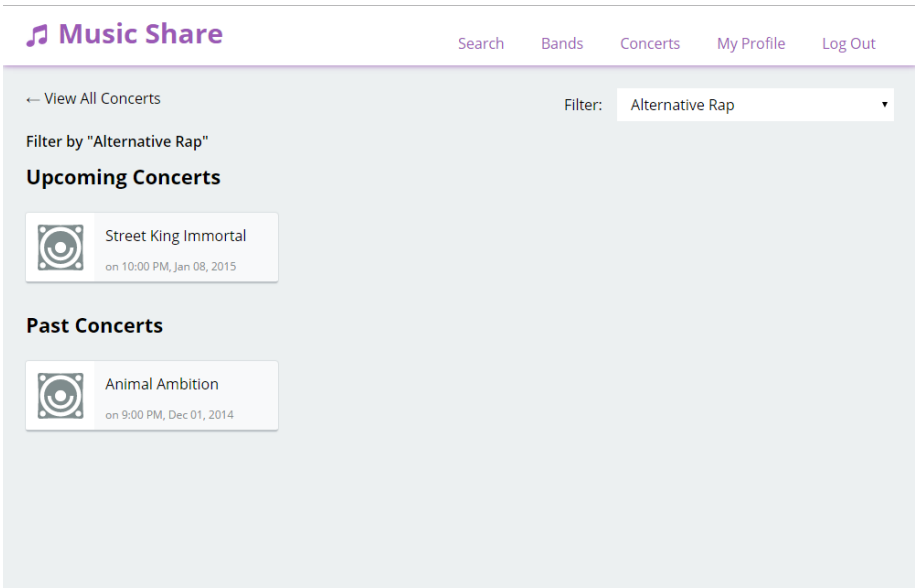
From search he can see recommended concerts, bands and people to follow.



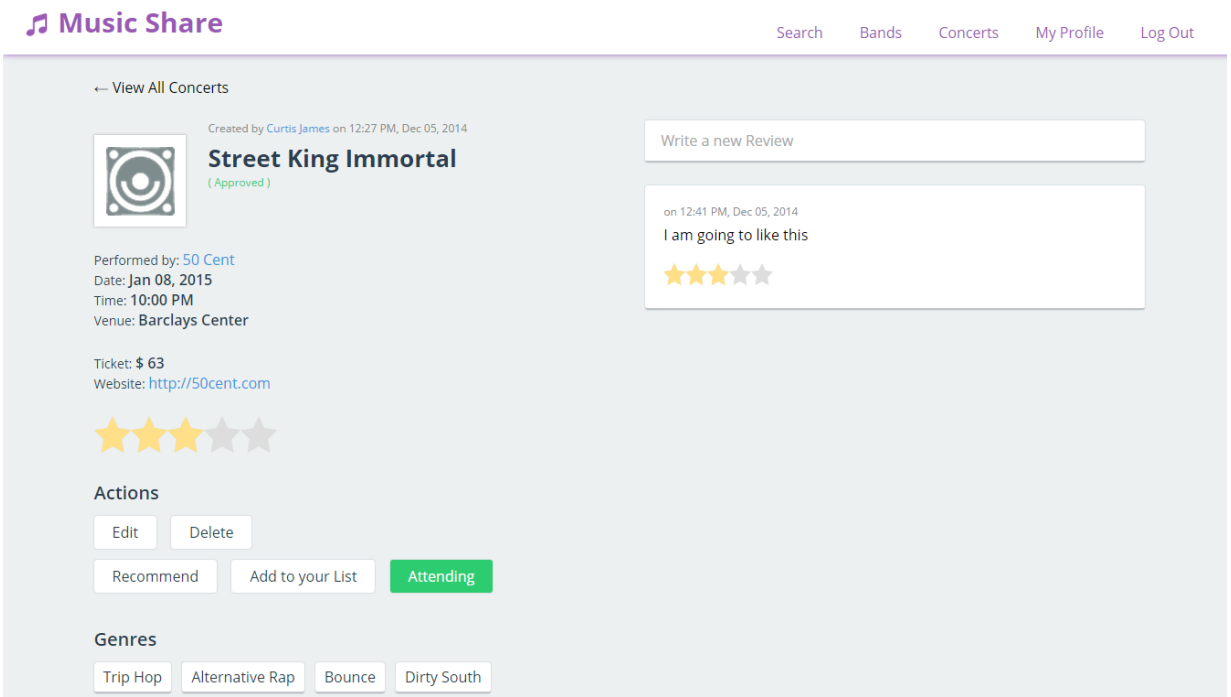
He can search concerts, bands and people through various options: date, concert creator, person name, genre etc.



He can view upcoming concerts and past concerts from concerts page. Also he can filter than based on their genres.

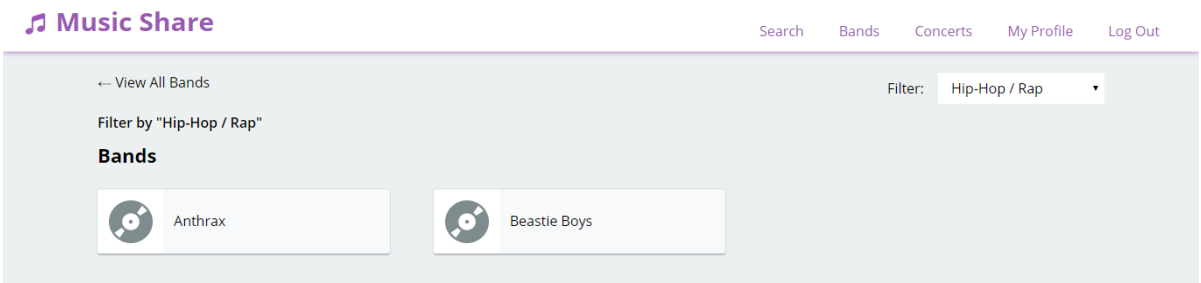


Clicking a concert will bring it to its detailed view.

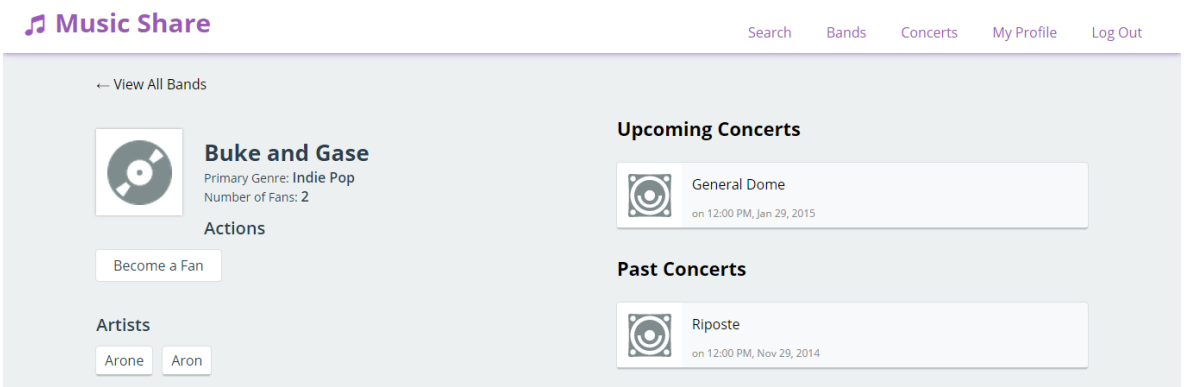


Same goes for the bands.

List of bands:



Detailed view:



Sample Queries

Following are some sample queries that is used to retrieve appropriate date for the application

1) User Data
   a. Signup: (the user id is 4)

      $insert\ into\ user(username, password)$
      $values('GK1991',' d165c85e6d6eacfb535e054ae88f5daa')$

userid	username	password	fname
4	GK1991	d165c85e6d6eacfb535e054ae88f5daa	NULL

lname	birthyear	email	rating	joindate	city
NULL	NULL	NULL	6	2014-11-24 12:53:33	NULL

   b. Create Profile:

      $update\ user\ set\ fname = `Gopi`, lname = `Krishna`,$
      $birthyear = `1991`, email = `gs1922@nyu.edu`,$
      $city = `NYC`\ where\ username = `GK1991`$

userid	username	fname	lname	birthyear	email
4	GK1991	Gopi	Krishna	1991	gs1922@nyu.edu

   c. Fan of a band:

      $insert\ into\ fan(userid, bandid)\ values(4,2)$

      + Options

userid	bandid	fandate
1	2	2014-11-24 11:51:02
2	1	2014-11-24 11:51:02
3	1	2014-11-24 11:51:02
3	2	2014-11-24 11:51:02

   d. Post a rating and review:

      $insert\ into\ review(cid, userid, rating, review)$
      $values(3,1,4.5, `This\ concert\ is\ awesome, you\ should\ come\ next\ time`)$

reviewid	userid	cid	rating	review	reviewdate
1	1	3	5	This concert is awesome,you should come next time	2014-11-24 13:06:16

2) Band and Concert Data

a. New concert

$insert\ into\ concert(userid, bandid, cname, venueid, ctime, ticket, url)$
$values(4,2,`Welcome\ Rockers`, 4, `2014-12-09\ 09{:}00{:}00`, 0, \text{NULL})$

cid	userid	bandid	cname	venueid	ctime	ticket	url	approved	created
3	1	1	Jazz on Moday	1	2014-11-24 11:29:34	123	*NULL*	1	2014-11-24 11:53:51
4	2	2	Rocking Songs	2	2014-11-27 11:20:00	*NULL*	*NULL*	0	2014-11-24 11:53:51
5	4	2	Welcome Rockers	4	2014-12-09 09:00:00	0	*NULL*	0	2014-11-24 13:12:19

b. New list of concerts by user

$insert\ into\ userlist(userid, listname)\ values(4, `Likes`)$

listid	userid	listname	listdate
1	3	My Fav List	2014-11-24 11:55:14
2	4	Likes	2014-11-24 13:14:27

c. Insert concerts into a user list (adding two concerts 3 and 5 into 'Likes' list)

$insert\ into\ list(listid, cid)\ values(2,3), (2,5)$

+ Options

listid	cid
1	3
1	4
2	3
2	5

3) Search data

a. Search Jazz concerts:

$select\ cid, cname, ctime\ from\ concert$
$join\ subcategory\ using(subcatid)$
$join\ category\ using(catid)$
$where\ catname = `Jazz`;$

cid	cname	ctime
3	Jazz on Moday	2014-11-24 11:29:34

b. Search concerts recommended by people they follow: (user id is 2)

$select\ cid, cname, ctime\ from\ follow$
$join\ recommend\ using\ (userid)$
$join\ concert\ using\ (cid)\ where\ followerid = 2$

+ Options

cid	cname	ctime
4	Rocking Songs	2014-11-27 11:20:00

c. Search new concerts since the users last login: (user id is 2)

$select\ cid, cname, ctime\ from\ concert$
$where\ created\ >\ (select\ logindate\ from\ user\ where\ userid\ =\ 2)$

cid	cname	ctime
3	Jazz on Moday	2014-11-24 11:29:34
4	Rocking Songs	2014-11-27 11:20:00
5	Welcome Rockers	2014-12-09 09:00:00

4) System Recommendations:

a. Recommendations based on user taste: (user id is 2)

$select\ bandid, bname\ from\ band$
$join\ subcategory\ using(catid)$
$where\ subcatid\ in\ (select\ subcatid\ from\ usergenre\ where\ userid\ =\ 2)$
$group\ by\ bandid, bname$

bandid	bname
2	Rock All Time

b. Recommendations based on taste of persons who the user follow: (user id is 3):

$select\ bandid, bname\ from\ band$
$join\ subcategory\ using(catid)$
$where\ subcatid\ in$
$\quad (select\ subcatid\ from\ usergenre$
$\quad join\ follow\ using(userid)$
$\quad where\ followerid\ =\ 3)$
$group\ by\ bandid, bname$

bandid	bname
1	GK Jazz
2	Rock All Time