DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

|  |  |
| --- | --- |
| <<PES2201800486>> | <<ADITYA ABHISHEK>> |

|  |
| --- |
| Radio Station Management System The project titled Radio Station Management System is an online music streaming platform. It is a web application for providing services such as music streaming, playlists, discovering new songs, search songs and inspirational quotes .  The Radio Station Management System is a web application designed for all kinds of operating systems capable of running a web browser. This software is easy to use for all kinds of people with little or no knowledge of computer operations. It features a familiar and well thought-out, an attractive user interface, combined with strong searching, insertion and reporting capabilities. |

[**Introduction**](#_gwqiqapyxr0x) **2**

[**Data Model**](#_p451stoz4u81) **3**

[**FD and Normalization**](#_wc0v2611q54c) **5**

[**DDL**](#_pn441yprtk2f) **7**

[**Triggers**](#_a9bswny9ffj) **10**

[**SQL Queries**](#_exvna2br67rz) **11**

[**Conclusion**](#_k6bbxckcf51j) **13**

# Introduction

* The user can login using **UserProfile** and add details to username, user\_id, city, State, date\_of\_birth, last\_name, first\_name and last\_logged\_in.
* The user then gives data for **States** in which he/she specifies from where he/she belongs to by giving state\_name.
* The user then gives data for **Places** where he/she lives in by giving state\_name and city\_name.
* The user also has **Credentials** in the form of username and md5\_passwd.
* The user gives his/her **FavouriteGenres** by filling in the genre.
* The user listens to **Songs** which has img\_url, rating, url, genre, song\_id, name, singer and film.
* The songs have tags in the form of **SongTags** tag.
* The user gets inspired from **Quotes** which has a quote\_url and the personality it is coming from.
* And finally the user is logged in the **LoggedInUser** and gets a token.

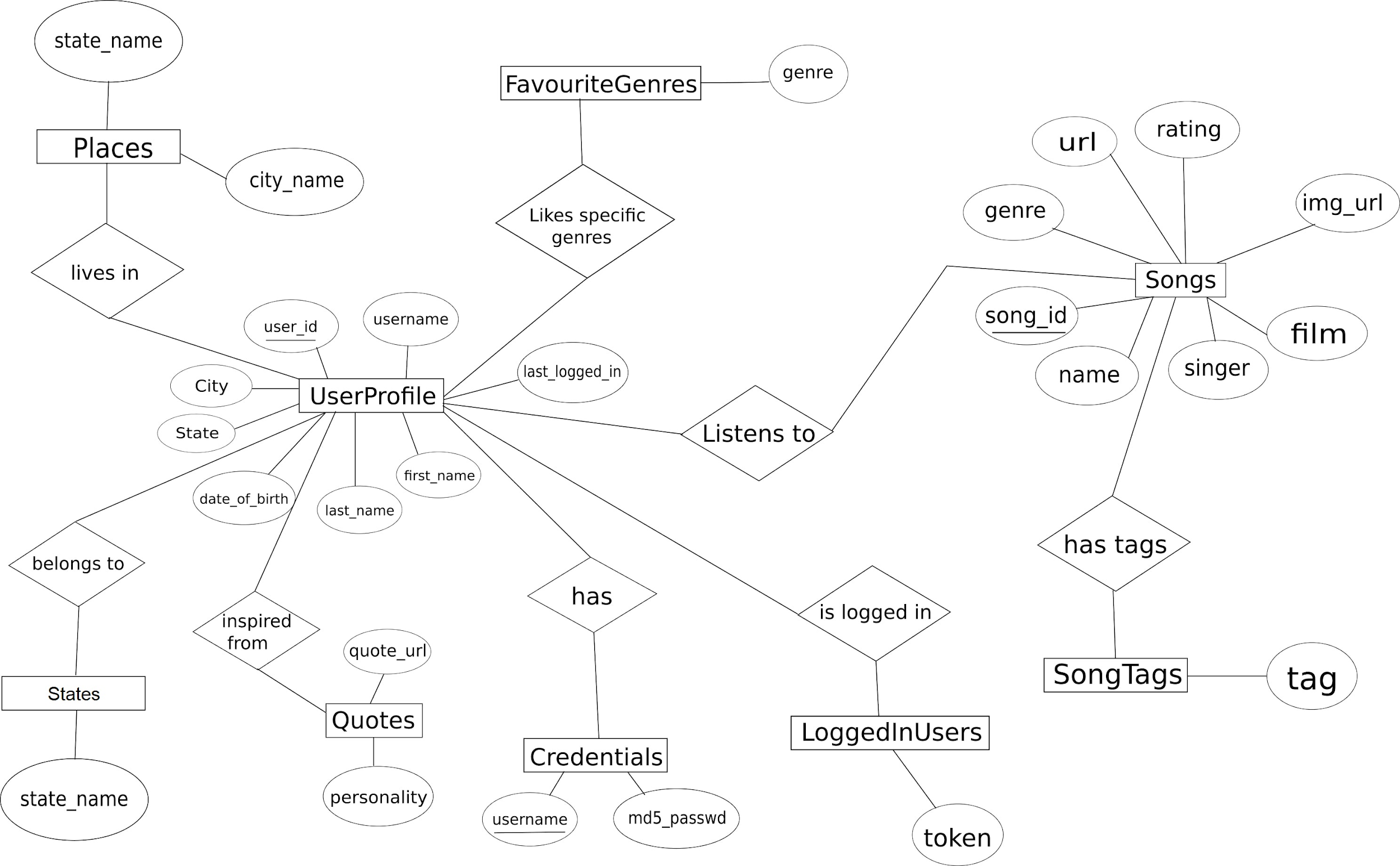
# 

# Data Model

## E.R. Diagram

## 

## 



## Relational Schema

## 

## Keys

### Primary Keys

* state\_name in table **States**
* city\_name in table **Places**
* user\_id in table **UserProfile**
* song\_id in table **Songs**
* User\_id in table **LoggedInUsers**

### Foreign Keys

* state\_name in table **Places** with references to state\_name in **States**.
* city in table **UserProfile** with references to city\_name in **Places**.
* state in table **UserProfile** with references to state\_name in **States**.
* username in **Credentials** with references to username in **UserProfile**.
* user\_id in **Credentials** with references to user\_id in **UserProfile**.
* user\_id in **FavouriteGenres** with references to user\_id in **UserProfile**.
* song\_id in **SongTags** with references to song\_id in **Songs**.
* user\_id in **LoggedInUsers** with references to user\_id in **UserProfile**.

### Data Types

For most of the text attributes we have used varchar data type.For number attributes we have used integer data type and numeric data type and finally we have used date data type for dates.

# FD and Normalization

## FUNCTIONAL DEPENDENCIES

**PLACES**

(city\_name) → state\_name

**USER PROFILE**

(user\_id) → username, first\_name, last\_name, date\_of\_birth, city, state, last\_logged\_in

**SONGS**

(song\_id ) → name, singer, film, genre, rating, url, img\_url

**FAVOURITE GENRES**

(user\_id) → genres

**QUOTES**

(quote\_url) → personality, count

**SONG TAGS**

(song\_id) → tag

**LOGGED IN USER**

(user\_id) → token

**CREDENTIALS**

(user\_id) → username, md5\_passwd

## NORMALISATION

**1NF:**Remove all multivalued attributes and nested relations

Since there are no multivalued and composite attributes as we can see in our ER diagram. Hence all the values are atomic and in first normal form.

1NF can get violated if we didn't convert the attribute ‘type’ into its own relation [non atomic]

**2NF:**Ensure full functional dependency

As we can see in the above functional dependencies, none of the relations have partial functional dependency. Hence all the given relations are in second normal form.

Let us discuss a case when 2NF can be violated.

Assume if we add user\_id attribute to relation SongTag, then this attribute will not depend on primary key song\_id hence it will violate 2NF.

**3NF:**Eliminate transitive dependency

As we can see there is no transitive dependency, all of our relation is in third normal form.

**BCNF**: Every dependency in a relation is either trivial or is dependent on a superkey

From the schema diagram, we can see the above rules for BCNF are abided in every relation. Thus, our relations are in BCNF.

# DDL

-- Table structure for table States

--

CREATE TABLE States(

state\_name varchar(100) PRIMARY KEY

);

-- Table structure for table Places

--

CREATE TABLE Places(

city\_name varchar(100) PRIMARY KEY,

state\_name varchar(100) NOT NULL,

FOREIGN KEY(state\_name) REFERENCES States(state\_name)

ON DELETE CASCADE ON UPDATE CASCADE

);

-- Table structure for table UserProfile

--

CREATE TABLE UserProfile(

user\_id int PRIMARY KEY,

username varchar(100) NOT NULL UNIQUE,

first\_name varchar(100) NOT NULL,

last\_name varchar(100),

date\_of\_birth date NOT NULL,

city varchar(100),

state varchar(100),

last\_logged\_in NUMERIC,

FOREIGN KEY(city) REFERENCES Places(city\_name)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY(state) REFERENCES States(state\_name)

);

-- Table structure for table Credentials

--

CREATE TABLE Credentials(

user\_id int,

username varchar(100),

md5\_passwd varchar(100) NOT NULL,

FOREIGN KEY(username) REFERENCES UserProfile(username)

ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY(user\_id) REFERENCES UserProfile(user\_id)

);

-- Table structure for table Songs

--

CREATE TABLE Songs(

song\_id int PRIMARY KEY,

name varchar(100) NOT NULL,

singer varchar(100),

film varchar(100),

genre varchar(100),

rating int,

url varchar(100) NOT NULL UNIQUE,

img\_url varchar(100) default '/images/defaultsong.jpg'

);

-- Table structure for table FavouriteGenres

--

CREATE TABLE FavouriteGenres(

user\_id int,

genre varchar(100) NOT NULL,

FOREIGN KEY(user\_id) REFERENCES UserProfile(user\_id),

PRIMARY KEY(user\_id,genre)

);

-- Table structure for table Quotes

--

CREATE TABLE Quotes(

quote\_url varchar(100) PRIMARY KEY,

personality varchar(100),

count int

);

-- Table structure for table SongTags

--

CREATE TABLE SongTags(

song\_id int,

tag varchar(100),

foreign key(song\_id) REFERENCES Songs(song\_id)

ON DELETE CASCADE ON UPDATE CASCADE

);

-- Table structure for table LoggedInUsers

--

CREATE TABLE LoggedInUsers(

user\_id int PRIMARY KEY,

token varchar(100) NOT NULL UNIQUE,

FOREIGN KEY(user\_id) REFERENCES UserProfile(user\_id)

);

# Triggers

We can apply a trigger in such a way that whenever a user logs into the website, last\_logged\_in for the user gets affected.

CREATE TRIGGER trgAfterUpdate ON LoggedInUsers

After Insert

AS

declare @user\_id int;

select @user\_id=i.user\_id from inserted i;

UPDATE UserProfile

SET last\_logged\_in=getdate()

WHERE user\_id=@user\_id;

GO



Here, we can see that the last\_logged\_in is changed to the current date.

# SQL Queries

* List all the songs which belong to the favourite genre of first name “aditya” .

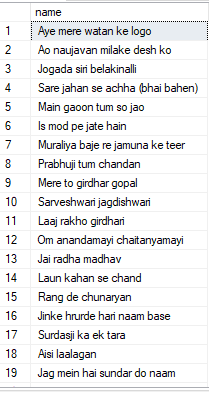
SELECT name

FROM Songs

WHERE genre IN (SELECT f.genre

FROM FavouriteGenres AS f, UserProfile AS u

WHERE u.first\_name=’aditya’ AND u.user\_id=f.user\_id);



* List the personality whose quotes are the most liked.

SELECT personality

FROM Quotes

WHERE count=(SELECT MAX(count)

FROM Quotes);

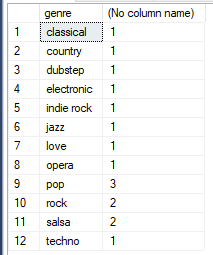


* List all the genres along with the number of users who like them.

SELECT genre, count(\*)

FROM FavouriteGenres

GROUP BY genre ;



* List out the songs with their name, id, genre, rating, url, img\_url,tag whose tag is present and the song\_id is between 1-100.

SELECT t.song\_id, name, genre, rating, url, img\_url,tag

FROM Songs AS s JOIN SongTags AS t ON s.song\_id=t.song\_id

WHERE t.song\_id BETWEEN 1 AND 100;



* Find all the inactive users and print when they were last logged in

SELECT last\_logged\_in, username

FROM UserProfile

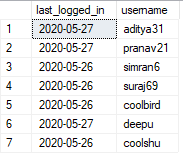
WHERE user\_id IN ( (SELECT user\_id

FROM UserProfile)

EXCEPT

(SELECT user\_id

FROM LoggedInUsers));



# Conclusion

As we have seen throughout this Report, very valuable statistics can be obtained by studying and querying the database in hand. With this database design, a Radio Station Service company can write similar queries periodically, and study the trends of the content they serve, the efficiency they provide with streaming,

From this project the user can either listen to songs based on his/her choice.

The users can update their profile and mention their favourite genres.

Listening and streaming songs are free but to download in high quality format the user has to pay.

Users can register for our website free of any charge. Users can enrich their mind and get inspired by the thoughts of great personalities.

Register with minimum details and then update profile

Many fields are optional which can be updated in their profile.

User mode is the default mode provided to the user. Users can listen to or buy songs in this mode. The user can select developer mode if required. The queries executed at the backend will be shown. The user will have an option to create a personal playlist of songs or select from the recently played.