Project 1: Music Recommender

Sanket Hire **2016CS50402**
Yash Malviya **2016CS50403**
Vasudev Singh **2016MT60660**

Due date: March 2, 2020, 11:55pm IST

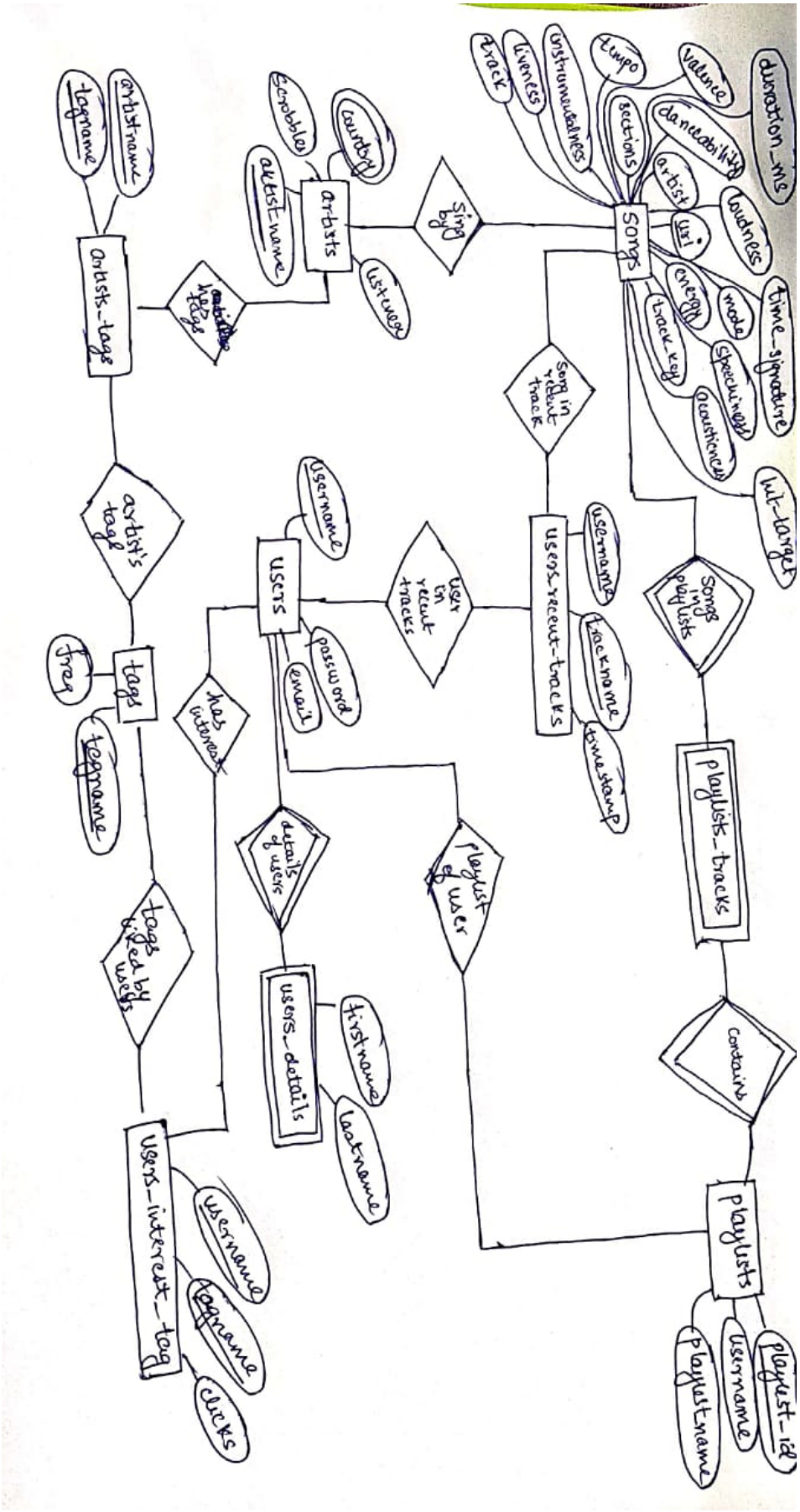
1 Section 1

This section provides brief description of our project.

1.1 Motivation

The main motive of this project is to provide a platform for music enthusiasts and listeners.This platform helps users in recommending songs of their taste and allow them to share their playlists with others.For automating the user music taste, we are developing the music recommendation system which recommends songs based on their listening history.User's listening history is being analysed on artist based features(artist name, country,tags, listeners and scrobbles) and song based features(track, artist, danceability, energy, track key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duration, time signature, chorus hit, track,sections and hit target).

1.2 E-R Diagram



1.3 List of Entities and Attributes

1.3.1 artists

Attribute Name	Description
artist_name(**primary key**)	Name of the artist
country(**Multi-valued attribute**)	Country to which artist belongs to
listeners	Number of listeners on last.fm
scrobbles	Number of scrobbles on last.fm

Table 1: artists

```
create table artists(
  artist_name text ,
  country text,
  listeners integer,
  scrobbles integer,
  primary key (artist_name)
);
```

1.3.2 songs

Attribute Name	Description
track	Name of the artist
artist	Country to which artist belongs to
uri(**primary key**)	The resource identifier for the track.
danceability	Danceability describes how suitable a track is for dancing based on a combination of musical elements .
energy	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. .
track_key	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. .
loudness	The overall loudness of a track in decibels (dB).
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. .
speechiness	Speechiness detects the presence of spoken words in a track.
acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
instrumentalness	Predicts whether a track contains no vocals.
liveness	Detects the presence of an audience in the recording.
valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
tempo	The overall estimated tempo of a track in beats per minute (BPM).
duration_ms	The duration of the track in milliseconds.
time_signature	The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). .
chorus hit	This the the author's best estimate of when the chorus would start for the track.
sections	The number of sections the particular track has.
hit_target	It can be either '0' or '1'. '1' implies that this song has featured in the weekly list (Issued by Billboards) of Hot-100 tracks in that decade at least once and is therefore a 'hit'. '0' Implies . that the track is a 'flop'.

Table 2: songs

```
create table songs(
  track text,
  artist text,
  uri varchar(36),
  danceability decimal,
  energy decimal,
```

```
    track_key integer,
    loudness decimal,
    mode decimal,
    speechiness decimal,
    acousticness decimal,
    instrumentalness decimal,
    liveness decimal,
    valence decimal,
    tempo decimal,
    duration_ms integer,
    time_signature integer,
    chorus_hit decimal,
    sections integer,
    hit_target integer,
    primary key (uri),
    foreign key (artist) references artists(artist_name) on delete cascade on update
  cascade
);
```

1.3.3 tags

Attribute Name	Description
tag_name(**primary key**)	Tag of the artist
freq	Frequency of the tags

Table 3: tags

```
create table tags(
    tag_name text unique,
    freq integer,
    primary key(tag_name)
);
```

1.3.4 artist_tags

Attribute Name	Description
artist_name	Name of the artist
tag_name	Tag of the artist

Table 4: artist_tags

```

create table artist_tags(
    artist_name text,
    tag_name text,
    primary key (artist_name, tag_name),
    foreign key (artist_name) references artists(artist_name) on delete cascade on update
  cascade,
    foreign key (tag_name) references tags(tag_name) on delete cascade on update cascade
);
```

1.3.5 users

Attribute Name	Description
username	Username of User
email	Email of User
passwd	Password of User

Table 5: users

```
create table users(
  username text,
  email text unique,
  passwd text,
  primary key (username)
);
```

1.3.6 user_details

Attribute Name	Description
username	Username of User
email	Email of User
passwd	Password of User

Table 6: user_details

```
create table user_details(
  username text ,
  first_name varchar(32),
  last_name varchar(32),
  primary key (username),
  foreign key (username) references users(username) on delete cascade on update cascade
);
```

1.3.7 playlists

Attribute Name	Description
playlist_id(**primary key**)	Playlist Id
username	Username of User
playlist_name	Name Of Playlist

Table 7: playlists

```
create table playlists(
  playlist_id serial,
  username text,
  playlist_name text,
  primary key (playlist_id),
  foreign key (username) references users(username) on delete cascade on update cascade
);
```

1.3.8 playlist_tracks

Attribute Name	Description
playlist_id	Playlist Id
track_uri	link for the track

Table 8: playlist_tracks

```
create table playlist_tracks(
  playlist_id integer,
  track_uri varchar(36),
  primary key (playlist_id, track_uri),
  foreign key (playlist_id) references playlists(playlist_id) on delete cascade on
update cascade,
  foreign key (track_uri) references songs(uri) on delete cascade on update cascade
);
```

Attribute Name	Description
username	Username of user
tag_name	Tag of the artist
clicks	Number of times the tag got queried

Table 9: user_interest_tags

1.3.9 user_interest_tags

```
create table user_interest_tags(
  username text,
  tag_name text,
  clicks integer,
  primary key (username, tag_name),
  foreign key (username) references users(username) on delete cascade on update cascade,
  foreign key (tag_name) references tags(tag_name) on delete cascade on update cascade
);
```

1.3.10 user_recent_tracks

Attribute Name	Description
username	Username of user
track_uri	link of the track
time_stamp	timestamp at which the song got requested to play

Table 10: user_recent_tracks

```
create table user_recent_tracks(
  username text,
  track_uri varchar(36),
  time_stamp timestamp,
  primary key (username, track_uri),
  foreign key (username) references users(username) on delete cascade on update cascade,
  foreign key (track_uri) references songs(uri) on delete cascade on update cascade
);
```

2 Section 2

2.1 Sources of our data

- Music artists popularity: **https://www.kaggle.com/pieca111/music-artists-popularity**
- The Spotify Hit Predictor Dataset (1960-2019): **https://www.kaggle.com/theoverman/the-spotify-hit-predictor-dataset**

2.2 How we downloaded it

We downloaded the "READYMADE" datasets from the links given in sources section**(subsection 2.1)**.

2.3 Cleanup Steps

- We first downloaded the datasets as per **(subsection 2.1)**
  - **Music artists popularity:** Artists dataset comprising the attributes - artist_name, country, tags, listeners, and scrobbles
    - **"artists_raw" table**
  - **The Spotify Hit Predictor Dataset (1960-2019):** Songs dataset comprising the attributes for tracks such as track, artist, danceability, energy, track key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duration, time signature, chorus hit, track sections, and hit target.

    - **dataset-of-00s.csv**
    - **dataset-of-10s.csv**
    - **dataset-of-60s.csv**
    - **dataset-of-70s.csv**
    - **dataset-of-80s.csv**
    - **dataset-of-90s.csv**

  We compiled the above csv files of songs dataset in one table **"songs_raw" table**

- **artists table**
    - Then we extracted the column attributes of **"artists_raw" table** - artist_name, country, listeners, and scrobbles.
    - and created **"artists" table** as shown in **(subsubsection 1.3.1)**.
    - With artist_name as a distinct attribute.
    - Additionally, keeping only those artists which have sung the songs as per **"songs" table** by using **join**.
- **songs table**
    - We extracted all the column attributes of **"songs_raw" table**.
    - Keeping those rows in which songs have a artist existing in **"artists_raw" table** by using **join**.
    - Stored the table in **"songs" table** as shown in **(subsubsection 1.3.2)**.
- **tags table**
    - This table contains (tag_name $\rightarrow$ freq(tag_frequency)).
    - tag_name is distinct attribute extracted from **"artists_tags" table's** tags attribute column.
    - freq(tag_frequency) is the frequency of each tag present in **"artists_tags" table's** tags attribute column.
    - Stored the table in **"tags" table** as shown in **(subsubsection 1.3.3)**.
- **artists_tags table**
    - Further extracted the column attributes of **"artists_raw" table** - artist_name and tags (keeping artist_name distinct across the rows).
    - Here tags were multi-valued for each artist.
    - So we created 1NF (artist_name $\rightarrow$ tags) table and stored the table as **"artist_tags" table** as shown in **(subsubsection 1.3.4)**.
- The tables below are for user queries which can be updated as per the user's input (Here, cleaning steps are not required at the database making stage):
    - users
    - user_details
    - playlists
    - playlist_tracks
    - user_interest_tags
    - user_recent_tracks

2.4 Statistics

Table	No. of Tuples	Time to Load	Raw Dataset Size	Raw Dataset Size (Clean)
artists_raw	1466083	3675.145 ms	201.1 MB	-
songs_raw	41106	264.162 ms	7728 kB	-
artists	7559	*	-	1024 kB
songs	32025	*	-	7872 kB
tags	7229	*	-	616 kB
artist_tags	221212	*	-	21 MB
users	1	-	-	24 kB
user_details	1	-	-	16 kB
playlists	2	-	-	16 kB
playlist_tracks	40	-	-	8192 bytes
user_interest_tags	786	-	-	16 kB
user_recent_tracks	10	-	-	16 kB

Table 11: Statistics

[*] These tables are not loaded from the csv but are the derived from the postgresql tables artist_raw and songs_raw that's why we have not written the time to load from them. Only query time can be deduced for these tables.

3 Section 3

3.1 User's view of the system

- Login
  - User is authenticated using a form to fill their username and password at the start
- SignUp
  - Interface for new users to create an account
  - User is required to fill certain details such as his/her first name, last name along with username, email and password.
- Update User's details
  - User details has email, First Name, Last Name
  - There is an interface to update this
- Create Playlist
  - This gives users option to save songs as a list and name the list
- Add song to playlist
  - Adding the a song the list described above
- Play / Mark song
  - This is most basic option of the application, where user can a song info or play it
- Search
  - A box to write in to search for songs and artists and various filtering and sorting options for the output
- Recommendation
  - from the perspective of the user, they will be able to see songs similar to the songs they hear the most

3.2 System view

- Signup
  - A new entry is being inserted into users table whenever a user signs up
  - Insert user data in user related entities such as users, user_details.
- Update User's Details
  - User privilege to update user_details attributes like user's username, first name and last name if requested.
- Create Playlist
  - A new entry is being added into the playlists table signifying the playlist id and the owner of that playlist.
- Add song to playlist
  - A new entry is being inserted into the playlist_tracks signifying the song name added to which playlist_id.
- Search
  - System administrator has a privilege to do search queries using "WHERE" clause for getting required tuples of attributes for a given set of conditions.
  - Example: Return the list of songs sung by an artist.

```
select track from songs where artist = 'Drake';

```

- Recommendation
  - Music is recommended using various information collected from the user as they operate the website
  - Type 1: User interest tags entire history
    - Artist with most matching tags with user interest tags are selected. The songs these artist created are recommended to them. Note that this is based on which tags they are interested in not how much they are interested in it.
    - Query:

```
    select
    *
from
    (select
    *
    from (
            select
            arts.artist_name,
            count(arts.artist_name)
            from (
                    select
                    artist_name
                    from (
                            select
                            tag_name
                            from user_interest_tags
                            where
                            username = 'yash98'
                    ) as in_tr,
                    artist_tags
                    where
                    artist_tags.tag_name = in_tr.tag_name
            ) as arts
            group by
            arts.artist_name
    ) as arts_count
```

```
    order by
    count desc) as arts_desc,
    songs
where arts_desc.artist_name = songs.artist;

```

* time: 4 sec 754 msec

- Type 2: User interest tags clicks based
  * Clicks denote how much user is interested in a particular tag. The query for this type of recommendation calculates clicks for the tags an artist has. Based on this artists priority for being recommended is calculated.
  * Query:

```
select
    *
from
    (select
    *
    from (
            select
            arts.artist_name,
            sum(arts.clicks)
            from (
                    select
                    artist_name, in_tr.tag_name, clicks
                    from (
                            select
                            tag_name, clicks
                            from user_interest_tags
                            where
                            username = 'yash98'
                    ) as in_tr,
                    artist_tags
                    where
                    artist_tags.tag_name = in_tr.tag_name
            ) as arts
            group by
            arts.artist_name) as arts_sum
    order by
    sum desc) as arts_desc,
    songs
where arts_desc.artist_name = songs.artist;

```

  * time: 4 sec 719 msec
- Type 3: Recent Tracks based
  * This extract tags from artist of a song then proceed to rank other artists the same way as in type 1.

3.2.1 Special functionality

- **Indexing**

1. **artist_country_index**

```
create index artist_country_index on artists using brin(country);

```

Query:

```
select artist_name from artists where country = 'India';

```

(a) index activate → **Time: 2.421 ms**

(b) index deactivate → **Time: 2.798 ms**

2. **artist_tag_index**

```
create index artist_tag_index on artist_tags using brin(tag_name);

```

Query:

```
select * from artist_tags where tag_name = 'rock';

```

(a) index activate → **Time: 45.183 ms**

(b) index deactivate → **Time: 46.649 ms**

3. **track_artist_index**

```
create index track_artist_index on songs using brin(artist);

```

(a) **Query 1**:

```
select track, artist from songs where artist = 'AC/DC';

```

(6 Rows)

i. index activate → **Time: 7.187 ms**

ii. index deactivate → **Time: 6.806 ms**

(b) **Query 2**:

```
select track, artist from songs where artist = 'Taylor Swift';

```

(49 Rows)

i. index activate → **Time: 6.251 ms**

ii. index deactivate → **Time: 6.768 ms**

- **Trigger**
  - It was used for two purposes
    - Maintain the upper limit of stored recent track for each user. This maintains fixed number of songs that the user listened last i.e. it maintains first in first out
    - Update tags user is interested as they listen to / mark a track. Increment clicks for a tag that user showed interest for.
  - **Query:**

```
create or replace function trigger_add_recent_track() returns trigger as $body$
 begin if (
  select
    count(*)
  from user_recent_tracks
  where
    user_recent_tracks.username = new.username
) >= 20 then begin
delete from user_recent_tracks
where
  new.username = user_recent_tracks.username
  and user_recent_tracks.time_stamp = (
    select
      min(time_stamp)
    from user_recent_tracks
    where
      new.username = user_recent_tracks.username
  );
end;
end if;
update user_interest_tags
set
  clicks = clicks + 1
where
  user_interest_tags.tag_name in (
    (
```

```
26        select
27          user_interest_tags.tag_name
28        where
29          user_interest_tags.username = new.username
30      )
31      intersect
32        (
33          select
34            artist_tags.tag_name
35          from artist_tags
36          where
37            artist_tags.artist_name = (
38              select
39                artist
40              from songs
41              where
42                songs.uri = new.track_uri
43            )
44        )
45    );
46  insert into user_interest_tags
47  select
48    new.username,
49    q1.tag_name,
50    1
51  from (
52      (
53        select
54          artist_tags.tag_name
55        from artist_tags
56        where
57          artist_tags.artist_name = (
58            select
59              artist
60            from songs
61            where
62              songs.uri = new.track_uri
63          )
64      )
65      except
66        (
67          select
68            user_interest_tags.tag_name
69          from user_interest_tags
70          where
71            user_interest_tags.username = new.username
72        )
73    ) as q1;
74  return new;
75  end $body$ language plpgsql;
76
77
78  create trigger add_recent_track before
79  insert on user_recent_tracks for each row execute procedure
    trigger_add_recent_track();
80
81
```

3.3 Timings

Timings have been stated along with queries