The People’s Music

Centralized Database for Reviews on Musical Pieces

IST 659 M004 Fall 2021

Eamon Gallagher, Kevin Harmer, & Emmanuel Victor Kamya

Table of Contents

1. Introduction …………………………………………………………3

2. Problem Formulation ……………………………………………………..3

3. Relational Model …………………………………………………………4
    a. Conceptual Model 4
    b. Logical Model 5

4. Data Dictionary ………………………………………………………..7

5. Business Rules …………………………………………………………8

6. Database System Infrastructure ………………………………………..9

7. SQL Codes …………………………………………………………….9

8. Major Data Questions …………………………………………………...23

9. Application Screens ……………………………………………………..27

10. Application Diagram …………………………………………………...36

11. Major Data Questions in the App………………………………………….36

12. Future Explorations ……………………………………………………….37

13. References ………………………………………………………………...37

I. Introduction

Since the dawn of the 21st century, portable, personalized music has swept over society. The need for music accessibility began in the early 1900's as live music expanded from the rich upper class to the modern working class. Popularity grew rapidly as music expanded into all areas of society, leading to a larger industrious need for musical outlets. This began with radio and forms of individual ownership (like records, cassettes, and eventually CD's). However, music accessibility took a major jump around the beginning of the 21st century as massive digital music libraries were introduced to the public.

Tech giants began investing in the musical entertainment industry. Musical libraries started to develop algorithms and playlists to appeal to listeners everywhere. These have finally led into the modern music industry as companies like Apple and Spotify dominate people's interest with their gigantic databases of music.

Despite the evolution of music accessibility, musical industries have no feedback system outside concert reviews and digital downloads. While artists and record companies surely have their own methods for analyzing performance, there is no centralized analysis for comparison among different artists, albums, genres and other musical entities. Our team has the solution: *The People's Music*. *The People's Music* is a musical review database implementation that gives the music industry a centralized resource for any audience feedback they would need. All they need to do is input their music into the system and let music listeners around the world give their feedback.

II. Problem Formulation

Because this database could evolve into several different things, our team formulated several problem solutions that could improve modern music listening. Due to *The People's Music* being in the initial stages of development, we narrowed down our possible objectives to 5 distinct goals.

1. Central database for music recommendations. All listeners, new or old, can browse the music review database for music that they have not heard before. They can compare between artists, albums, or genres to find newer recommendations that are perfect fits for their musical tastes.

2. Diverse mix of opinions. *The People's Music* provides a social platform for listeners to share their opinions with other listeners. The database can provide an outlet for musical minds to contribute to the industry.

3. Gives feedback for artists. Although artists know when their content gets a big achievement or hits a lot of views, they have no direct connection with their audience. Now, they can look at their different ratings and understand what their

audience likes and what they do not like, which could help them with new content.

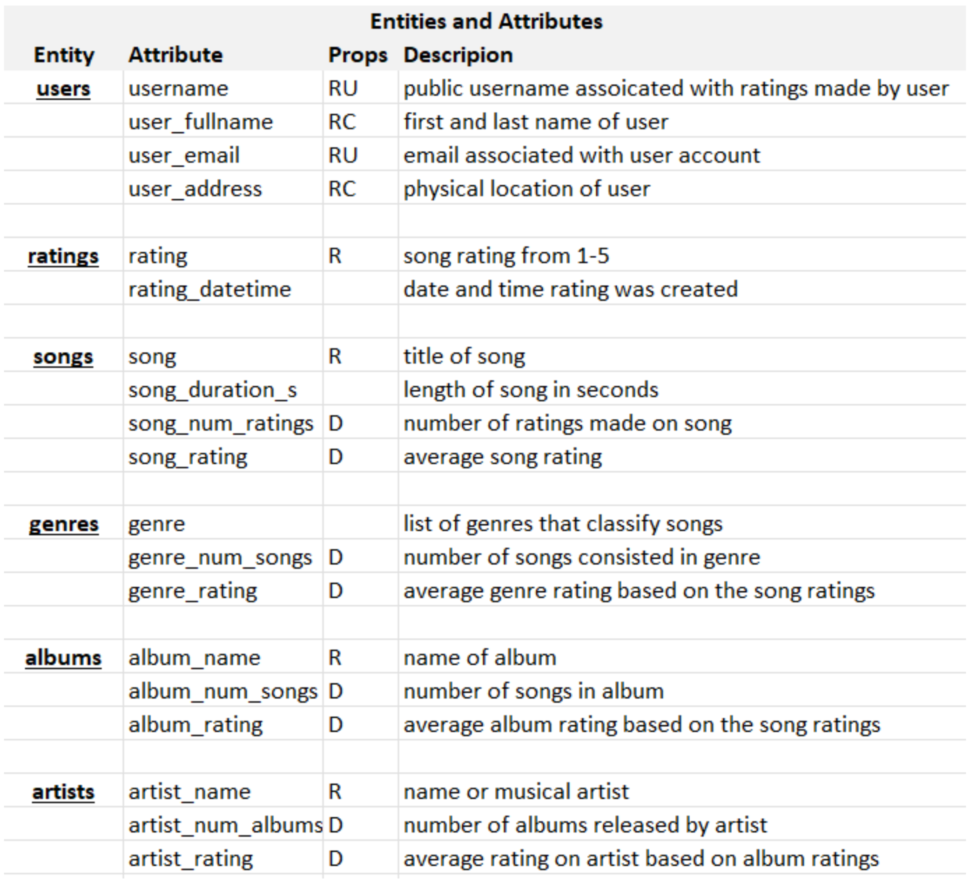
4. Compares popularity of similar artists. The only way musical listeners could compare artists is to look at music critiques, awards or public appearance. Now, they can check out *The People's Music* and learn the public's favorite musicians. This provides a new and unique objective analysis on different parts of the music industry.

5. Compares popularity between genres. There are many listeners who bounce around between genres. Another goal of *The People's Music* is to show rating differences between genres to give those listeners objective feedback on genre popularity.

III. Relational Model

1. Developing the Conceptual Model

ER Requirements:

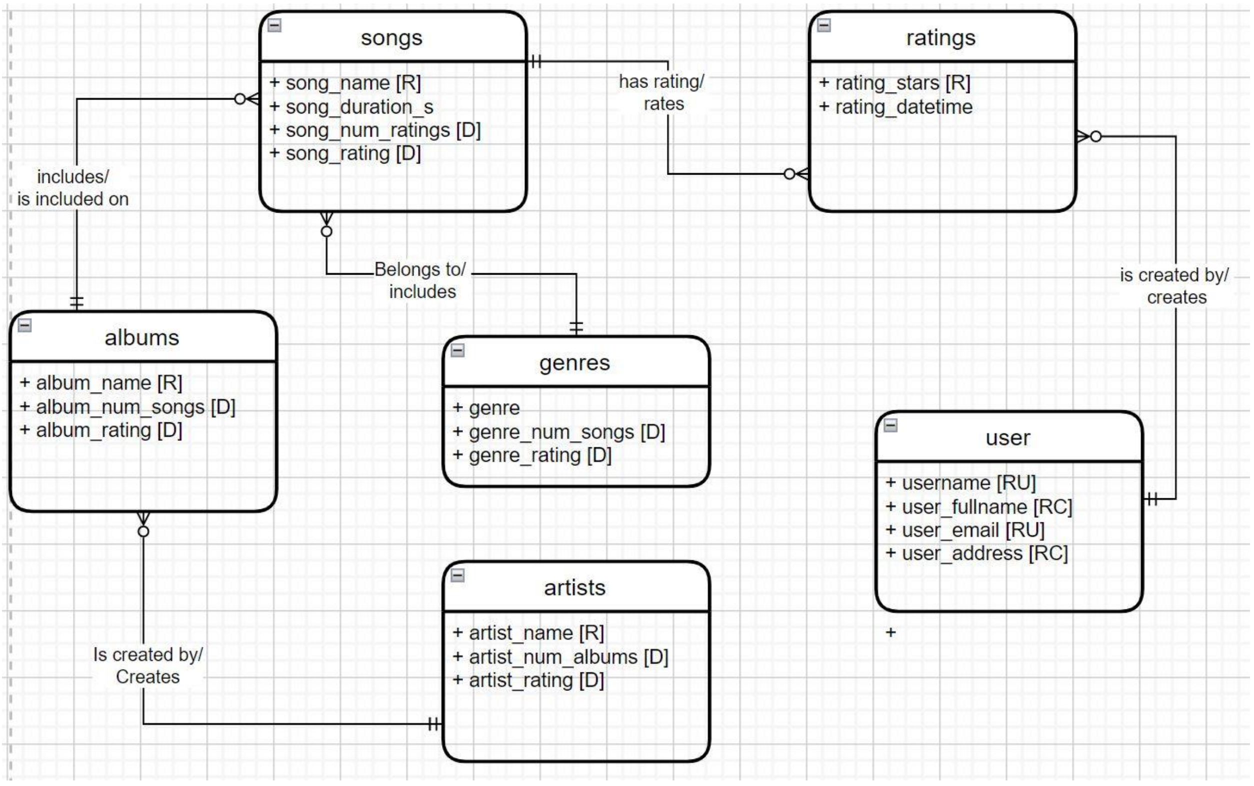


Entities and Attributes			
Entity	**Attribute**	**Props**	**Descripion**
users	username	RU	public username assoicated with ratings made by user
	user_fullname	RC	first and last name of user
	user_email	RU	email associated with user account
	user_address	RC	physical location of user
ratings	rating	R	song rating from 1-5
	rating_datetime		date and time rating was created
songs	song	R	title of song
	song_duration_s		length of song in seconds
	song_num_ratings	D	number of ratings made on song
	song_rating	D	average song rating
genres	genre		list of genres that classify songs
	genre_num_songs	D	number of songs consisted in genre
	genre_rating	D	average genre rating based on the song ratings
albums	album_name	R	name of album
	album_num_songs	D	number of songs in album
	album_rating	D	average album rating based on the song ratings
artists	artist_name	R	name or musical artist
	artist_num_albums	D	number of albums released by artist
	artist_rating	D	average rating on artist based on album ratings

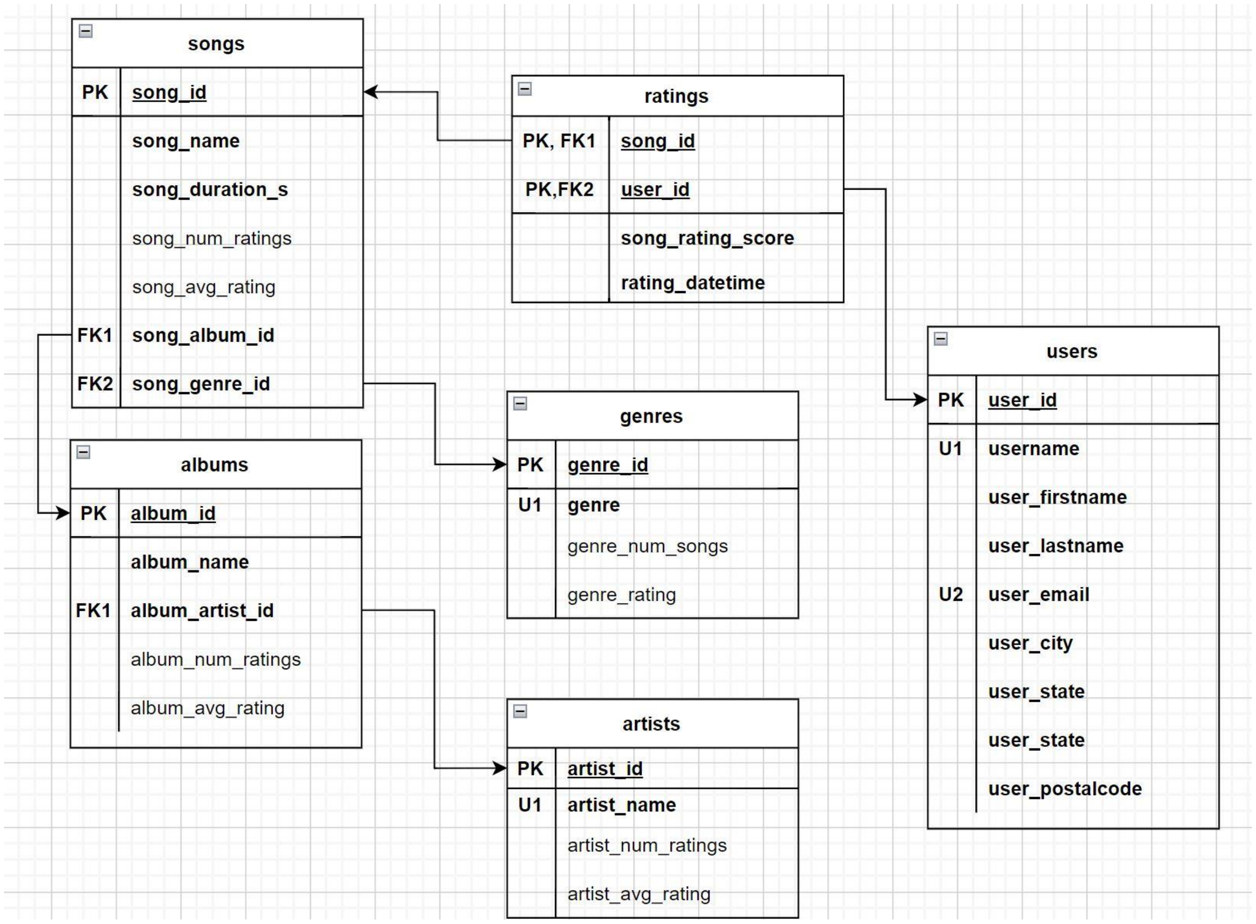
Relationships

Relationship	**Entity**	**Rule**	**Min**	**Max**	**Entity**
users-ratings	**users**	create	0	M	**ratings**
	ratings	are created by	1	1	**users**
ratings-songs	**ratings**	rate	1	1	**songs**
	songs	have	0	M	**ratings**
songs-genres	**songs**	belong to	1	1	**genres**
	genres	include	0	M	**songs**
songs-albums	**songs**	are included on	1	1	**albums**
	albums	include	0	M	**songs**
albums-artists	**albums**	are created by	1	1	**artists**
	artists	create	0	M	**albums**

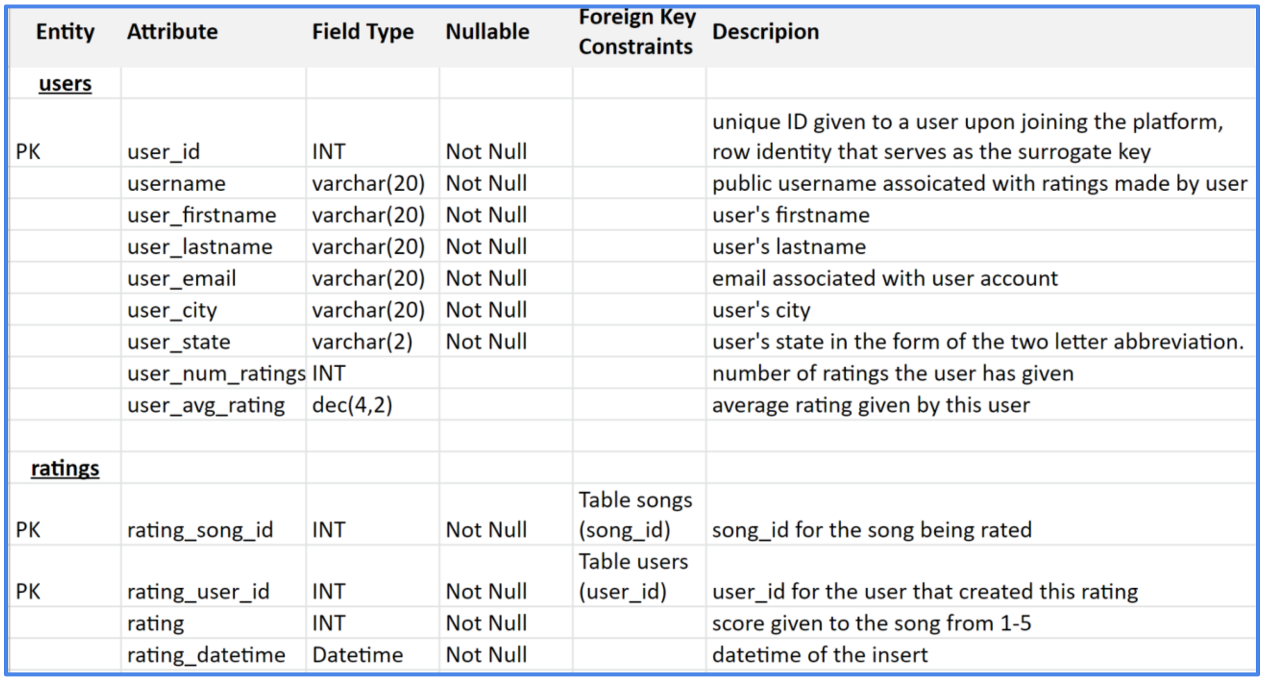
ERD Model:



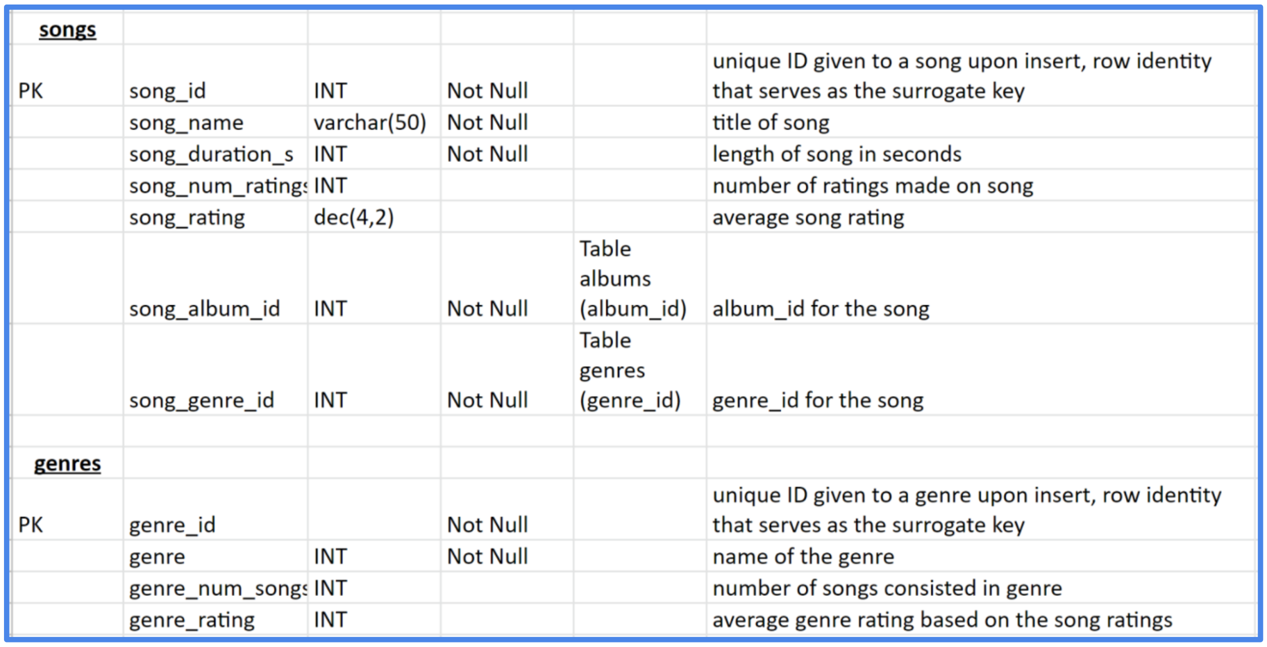
2. Logical Model



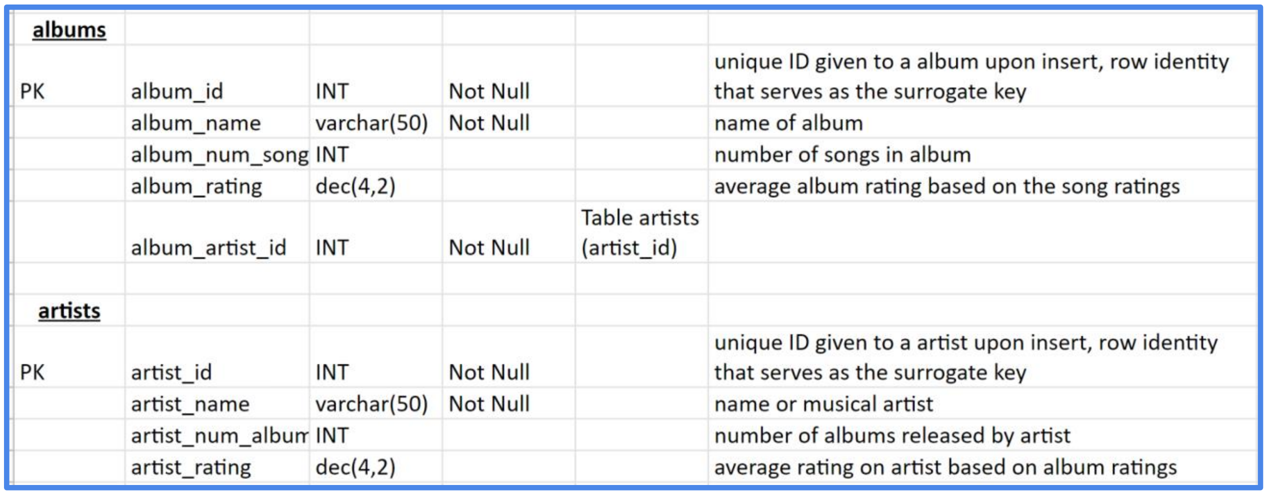
IV. Data Dictionary



Entity	Attribute	Field Type	Nullable	Foreign Key Constraints	Descripion
users					
PK	user_id	INT	Not Null		unique ID given to a user upon joining the platform, row identity that serves as the surrogate key
	username	varchar(20)	Not Null		public username assoicated with ratings made by user
	user_firstname	varchar(20)	Not Null		user's firstname
	user_lastname	varchar(20)	Not Null		user's lastname
	user_email	varchar(20)	Not Null		email associated with user account
	user_city	varchar(20)	Not Null		user's city
	user_state	varchar(2)	Not Null		user's state in the form of the two letter abbreviation.
	user_num_ratings	INT			number of ratings the user has given
	user_avg_rating	dec(4,2)			average rating given by this user
ratings					
PK	rating_song_id	INT	Not Null	Table songs (song_id)	song_id for the song being rated
PK	rating_user_id	INT	Not Null	Table users (user_id)	user_id for the user that created this rating
	rating	INT	Not Null		score given to the song from 1-5
	rating_datetime	Datetime	Not Null		datetime of the insert



songs					
PK	song_id	INT	Not Null		unique ID given to a song upon insert, row identity that serves as the surrogate key
	song_name	varchar(50)	Not Null		title of song
	song_duration_s	INT	Not Null		length of song in seconds
	song_num_ratings	INT			number of ratings made on song
	song_rating	dec(4,2)			average song rating
	song_album_id	INT	Not Null	Table albums (album_id)	album_id for the song
	song_genre_id	INT	Not Null	Table genres (genre_id)	genre_id for the song
genres					
PK	genre_id		Not Null		unique ID given to a genre upon insert, row identity that serves as the surrogate key
	genre	INT	Not Null		name of the genre
	genre_num_songs	INT			number of songs consisted in genre
	genre_rating	INT			average genre rating based on the song ratings



albums					
PK	album_id	INT	Not Null		unique ID given to a album upon insert, row identity that serves as the surrogate key
	album_name	varchar(50)	Not Null		name of album
	album_num_song	INT			number of songs in album
	album_rating	dec(4,2)			average album rating based on the song ratings
	album_artist_id	INT	Not Null	Table artists (artist_id)	
artists					
PK	artist_id	INT	Not Null		unique ID given to a artist upon insert, row identity that serves as the surrogate key
	artist_name	varchar(50)	Not Null		name or musical artist
	artist_num_album	INT			number of albums released by artist
	artist_rating	dec(4,2)			average rating on artist based on album ratings

V. Business Rules

Going into the internal model development, *The People's Music* has to stay consistent with a few business rules.

1. All genres and artists must be unique. If one genre or artist is duplicated, there would be problems in the display of artist and genre rating.
2. Similar to 1., all albums must be unique for any given artist. Again, multiple inputs for the same album would cause problems in the display of album rating.
3. Similar to 1. and 2., all songs in an album must be unique. Each song in an album must be unique to ensure that each displayed song rating is correct.
4. Users can only submit one rating on each song. This can be achieved in the internal model by a composite key in the ratings table
5. Users can rate as many songs as they would like
6. Artists are rated based on their unweighted albums, meaning each album has the same impact regardless of how many ratings it has.
7. Albums and genres are rated based on their unweighted songs, meaning each song has the same impact regardless of how many ratings it has.
8. Emails can only be linked to one account
9. Ratings must be an integer between 1 and 5
10. Songs/Genres/Albums/Artists ratings have null values until a song in each category receives a rating
11. Music information (songs, albums, artists, and genres) are primarily inputted into the internal model while ratings and users are primarily inputted into the physical/external model.

VI. Data System Infrastructure

We used the following to tools to create and implement *The People's Music*:

1. Microsoft Excel - Excel Spreadsheets were used to create and organize the ER requirements and the Data Dictionary
2. Draw.io - Drawings of the Logical and Conceptual Models were made in Draw.io
3. Azure Data Studio - The internal model (up/down script) was constructed in Azure Data Studio. Tables and constraints were created and data was inserted to run the functionality of the database.
4. Database is hosted on MS Azure servers.
5. Microsoft Power Apps - The database application was built in Power Apps. Data was imported into Power Apps from Azure and built into a simple functional application.

VII. SQL Code

```sql
if not exists(select * from sys.databases where name='music')
    create database music
GO

use music
go

-- DOWN
-- ratings table
if exists(select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_NAME='fk_ratings_rating_song_id')
    alter table ratings drop constraint fk_ratings_rating_song_id
if exists(select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_NAME='fk_ratings_rating_by_user')
    alter table ratings drop constraint fk_ratings_rating_by_user
drop table if exists ratings

-- songs table
if exists(select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_NAME='fk_songs_song_album_id')
    alter table songs drop constraint fk_songs_song_album_id
```

```
if exists(select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_NAME='fk_songs_song_genre_id')
    alter table songs drop constraint fk_songs_song_genre_id
drop table if exists songs

-- users table
drop table if exists users

-- albums table
if exists(select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_NAME='fk_albums_album_artist_id')
    alter table albums drop constraint fk_albums_album_artist_id
drop table if exists albums

-- genres table
drop table if exists genres

-- artists table
drop table if exists artists

-- UP Metadata
-- artists table
create table artists(
    artist_id int identity not null,
    artist_name varchar(50) not null, -- unique and not multivalued;
may be band with no first/last name
    constraint pk_artists_artist_id primary key (artist_id),
    constraint u_artists_artist_name unique (artist_name)
)

-- genres table
create table genres(
    genre_id int identity not null,
    genre varchar(20) not null,
    constraint pk_genres_genre_id primary key(genre_id),
    constraint u_genres_genre unique (genre) -- genres must be unique
)

-- albums table
create table albums(
    album_id int identity not null,
    album_name varchar(50) not null, -- different artists can have
same album names; no unique constraint needed
    album_artist_id int not null, -- foreign key to artist table
    constraint pk_albums_album_id primary key (album_id)
)
alter table albums
    add constraint fk_albums_album_artist_id foreign key
(album_artist_id)
    references artists(artist_id)

-- users table
create table users(
    user_id int identity not null,
    username varchar(20) not null,
    user_firstname varchar(20) not null,
    user_lastname varchar(20) not null,
    user_email varchar(50) not null,
    user_city varchar(20) not null,
    user_state varchar(2) not null,
    constraint pk_users_user_id primary key(user_id),
    constraint u_users_user_email unique(user_email),
    constraint u_users_username unique(username)
)
-- songs table
create table songs(
    song_id int identity not null,
    song_name varchar(50) not null, -- titles limited to 50
```

```
characters
    song_duration_s int not null, -- not sure if time is right
    song_album_id int not null, -- foreign key to album table
    song_genre_id int not null, -- foreign key to genre table
    constraint pk_songs_song_id primary key(song_id)
)
alter table songs
    add constraint fk_songs_song_genre_id foreign key (song_genre_id)
    references genres(genre_id)
alter table songs
    add constraint fk_songs_song_album_id foreign key (song_album_id)
    references albums(album_id)

-- ratings table
create table ratings(
    rating_song_id int not null, -- foreign key to song table
    rating int not null, -- can be decimal if want better analysis
    rating_by_user int not null, -- foreign key to user table
    rating_datetime smalldatetime not null default current_timestamp,
    constraint pk_ratings_by_user_on_song primary key
(rating_song_id, rating_by_user),
    constraint ck_ratings_min_max_rating check (rating >= 1 and
rating <= 5) -- ratings are between 1 and 5
)
alter table ratings
    add constraint fk_ratings_rating_by_user foreign key
(rating_by_user)
    references users(user_id)
alter table ratings
    add constraint fk_ratings_rating_song_id foreign key
(rating_song_id)
    references songs(song_id)

-- Derived Columns
-- count of ratings & avg ratings for songs
drop function if exists CountSongRatings
go
CREATE FUNCTION dbo.CountSongRatings (@SongID INT)
RETURNS INT
AS BEGIN
    DECLARE @RatingCount INT
    SELECT @RatingCount = COUNT(*) FROM ratings WHERE rating_song_id
= @SongID
    RETURN @RatingCount
END
go

ALTER TABLE songs
ADD song_num_ratings AS dbo.CountSongRatings(song_id)

Drop function if exists AvgSongRating
go
CREATE FUNCTION dbo.AvgSongRating (@SongID INT)
RETURNS dec(4,3)
AS BEGIN
    DECLARE @AvgRating dec(4,3)
    SELECT @AvgRating = avg(cast(rating as decimal(4,3))) FROM
ratings WHERE rating_song_id = @SongID
    RETURN @AvgRating
END
go

ALTER TABLE songs
ADD song_rating AS dbo.AvgSongRating(song_id)

-- count of songs and avg song rating for albums and genres
```

```
drop function if exists CountSongs
go
CREATE FUNCTION dbo.CountSongs (@AlbumID INT)
RETURNS INT
AS BEGIN
    DECLARE @SongCount INT
    SELECT @SongCount = COUNT(*) FROM songs WHERE song_album_id =
@AlbumID
    RETURN @SongCount
END
go

ALTER TABLE albums
ADD album_num_songs AS dbo.CountSongs(album_id)

ALTER TABLE genres
ADD genre_num_songs AS dbo.CountSongs(genre_id)

drop function if exists AlbumRating
go
CREATE FUNCTION dbo.AlbumRating (@AlbumID INT)
RETURNS dec(4,3)
AS BEGIN
    DECLARE @AvgRating dec(4,3)
    SELECT @AvgRating = avg(song_rating) FROM songs WHERE
song_album_id = @AlbumID
    RETURN @AvgRating
END
go

ALTER TABLE albums
ADD album_rating AS dbo.AlbumRating(album_id)

ALTER TABLE genres
```

```
ADD genre_rating as dbo.AlbumRating(genre_id)

-- number of albums and avg album rating for artists

drop function if exists CountAlbums
go
CREATE FUNCTION dbo.CountAlbums (@ArtistID INT)
RETURNS int
AS BEGIN
    DECLARE @AlbumCount int
    SELECT @AlbumCount = COUNT(*) FROM albums WHERE album_artist_id =
@ArtistID
    RETURN @AlbumCount
END
go

ALTER TABLE artists
ADD artist_num_albums AS dbo.CountAlbums(artist_id)

drop function if exists ArtistRating
go
CREATE FUNCTION dbo.ArtistRating (@ArtistID INT)
RETURNS dec(4,3)
AS BEGIN
    DECLARE @AvgRating dec(4,3)
    SELECT @AvgRating = avg(album_rating) FROM albums WHERE
album_artist_id = @ArtistID
    RETURN @AvgRating
END
go

ALTER TABLE artists
ADD artist_rating AS dbo.ArtistRating(artist_id)
```

```
-- number of ratings and avg rating for users

drop function if exists CountUserRatings
go
CREATE FUNCTION dbo.CountUserRatings (@UserID INT)
RETURNS INT
AS BEGIN
    DECLARE @RatingCount INT
    SELECT @RatingCount = COUNT(*) FROM ratings WHERE rating_by_user
= @UserID
    RETURN @RatingCount
END
go

ALTER TABLE users
ADD user_num_ratings AS dbo.CountUserRatings(user_id)

Drop function if exists UserAvgRating
go
CREATE FUNCTION dbo.UserAvgRating (@UserID INT)
RETURNS dec(4,3)
AS BEGIN
    DECLARE @AvgRating dec(4,3)
    SELECT @AvgRating = avg(cast(rating as decimal(4,3))) FROM
ratings WHERE rating_by_user = @UserID
    RETURN @avgRating
END
go

ALTER TABLE users
ADD user_avg_rating AS dbo.UserAvgRating(user_id)

-- UP Data
```

```sql
insert into artists -- may add, but do not reorder; fill mess up
foreign key
    (artist_name)
    values
    ('Two Door Cinema Club'),
    ('Mac Demarco'),
    ('Drake'),
    ('Billy Joel'),
    ('Taylor Swift'),
    ('Luke Combs'),
    ('Jordan Davis'),
    ('Avicii'),
    ('Bruno Mars'),
    ('Ed Sheeran'),
    ('Green Day'),
    ('Kayne West'),
    ('Queen'),
    ('Coldplay')

insert into genres -- may add but do not reorder; will mess up
foreign key
    (genre)
    values
    ('Pop'),
    ('Rock'),
    ('Country'),
    ('Hip-Hop/Rap'),
    ('Dance/Electronic'),
    ('Latin'),
    ('Alternative')

insert into albums
    (album_name, album_artist_id)
    values
    ('Tourist History', 1),
    ('2', 2),
    ('Certified Lover Boy',3),
    ('Scorpion', 3),
    ('An Innocent Man', 4),
    ('Glass Houses', 4),
    ('52nd Street', 4),
    ('The Stranger', 4),
    ('Fearless', 5),
    ('Red', 5),
    ('1989', 5),
    ('This One''s for You Too', 6),
    ('Home State', 7),
    ('Buy Dirt', 7),
    ('True', 8),
    ('Doo-Wops & Hooligans', 9),
    ('Unorthodox Jukebox', 9),
    ('24K Magic', 9),
    ('Divide', 10),
    ('American Idiot', 11),
    ('My Beautiful Dark Twisted Fantasy', 12),
    ('Donda', 12),
    ('The Game', 13),
    ('A Night at the Opera', 13),
    ('A Rush of Blood to the Head', 14)

insert into users
    (username, user_firstname, user_lastname, user_email, user_city,
user_state)
    values
    ('eamong_musicman', 'Eamon', 'Gallagher', 'etgallag@syr.edu',
'Syracuse', 'NY'),
    ('joey_beats',  'Joseph', 'Baloney', 'joeyb@mail.org', 'New York
City', 'NY'),
    ('notKanyeWest', 'Kayne', 'East', 'kwest@rap.org', 'Los Angeles',
'CA'),
    ('jgyl', 'Jake', 'Gyllenhaal','jgyl@hollywood.com', 'Los
Angeles', 'CA'),
    ('rapsfacts', 'Aubrey', 'Graham', 'drake@rap.org', 'Toronto',
'ON')

insert into songs -- may add but do not reorder; ratings based on
ordered song id
    (song_name, song_duration_s, song_album_id, song_genre_id)
    values
    ('I Can Talk', 177, 1, 1),
    ('Freaking Out the Neighborhood', 173, 2, 2),
    ('Fair Trade', 291, 3, 4),
    ('God''s Plan', 198, 4, 4),
    ('The Longest Time', 220, 5, 2),
    ('Uptown Girl', 198, 5, 2),
    ('You May Be Right', 255, 6, 2),
    ('My Life', 230, 7, 2),
    ('Vienna', 214, 8, 2),
    ('Love Story', 234, 9, 1),
    ('You Belong With Me', 231, 9, 1),
    ('All Too Well (10 minute version)', 613, 10, 1),
    ('All Too Well', 329, 10, 1),
    ('I Knew You Were Trouble', 219, 10, 1),
    ('We Are Never Getting Back Together', 193, 10, 1),
    ('Shake It Off', 219, 11, 1),
    ('Honky Tonk Highway', 213, 12, 3),
    ('Beautiful Crazy', 193, 12, 3),
    ('Slow Dance in a Parking Lot', 193, 13, 3),
    ('Buy Dirt', 167, 14, 3),
    ('Wake Me Up', 249, 15, 5),
    ('Grenade', 222, 16, 1),
    ('Just The Way You Are', 221, 16, 1),
    ('When I Was Your Man', 214, 17, 1),
    ('24K Magic', 226, 18, 1),
    ('That''s What I Like', 206, 18, 1),
    ('Castle on the Hill', 261, 19, 1),
    ('Shape of You', 233, 19, 1),
    ('Holiday', 232, 20, 2),
    ('Boulevard of Broken Dreams', 260, 20, 2),
    ('Runaway', 339, 21, 4),
    ('Power', 292, 21, 4),
    ('Off the Grid', 339, 22, 4),
    ('Crazy Little Thing Called Love', 162, 23, 2),
    ('Another One Bites the Dust', 215, 23, 2),
    ('Bohemian Rhapsody', 355, 24, 2),
    ('The Scientist', 266, 25, 7),
    ('Clocks', 250, 25, 7)

insert into ratings
    (rating_song_id, rating, rating_by_user)
    values
    (1, 3, 1),
    (1, 2, 2),
    (2, 5, 1),
    (12, 1, 4),
    (13, 1, 4),
    (3, 1, 3),
    (4, 1, 3),
    (31, 5, 3),
    (32, 5, 3),
    (33, 5, 3),
    (3, 4, 5),
    (4, 5, 5),
    (31, 3, 5),
    (32, 3, 5),
    (33, 2, 5)
```

```
-- Verfify
/*select * from artists order by artist_id
select * from genres order by genre_id
select * from albums order by album_id
select * from songs order by song_id
select * from users order by user_id
select * from ratings order by rating_datetime desc*/

-- Data Questions
-- 1: Music Recommendations
-- album view with artists
drop view if exists album_artists
go
create view album_artists as (
    select al.album_id, al.album_name, al.album_num_songs,
al.album_rating, ar.artist_name
        from albums al
        left join artists ar on al.album_artist_id = ar.artist_id
)
go
-- song view with albums, artists and genres
drop view if exists song_album_artist_genre
go
create view song_album_artist_genre as (
    select s.song_id, s.song_name, s.song_duration_s,
s.song_num_ratings, s.song_rating, aa.album_name, aa.artist_name,
g.genre
        from songs s
            left join album_artists aa on s.song_album_id =
aa.album_id
            left join genres g on s.song_genre_id = g.genre_id
)
go
select * from song_album_artist_genre -- song comparison with all
relevant information
select * from album_artists -- album comparison with all relevant
informaiton
select * from artists -- artist comparison
select * from genres -- genre comparison

-- 2: User opinions
-- specific ratings
drop view if exists user_ratings
go
create view user_ratings as
    (select u.user_id, u.username, u.user_num_ratings,
u.user_avg_rating, rating_song_id, rating
        from users u
            left join ratings r on u.user_id = r.rating_by_user)
go
-- specific ratings on songs
drop view if exists user_rating_songs
go
create view user_rating_songs as (
    select ur.user_id, ur.username, ur.rating, s.song_name,
ur.user_num_ratings, ur.user_avg_rating
        from user_ratings ur
            left join songs s on ur.rating_song_id = s.song_id
)
go
select * from user_rating_songs

-- 3: Artist Feedback; example = Kayne West
select * from song_album_artist_genre where artist_name = 'Kayne
West'
```

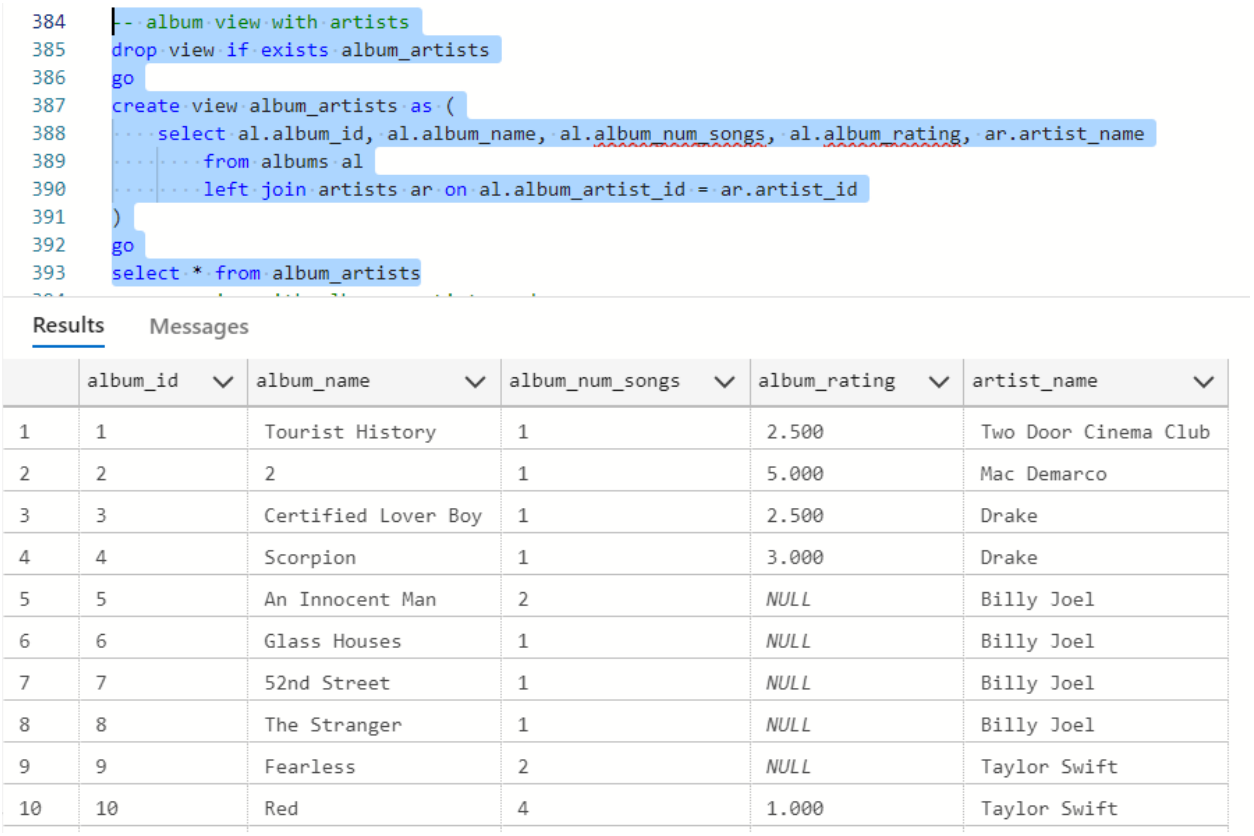
```
-- or by album
select * from album_artists where artist_name = 'Kayne West'

-- 4: Evaluating Artists; example = comparing Drake and Kayne West
select * from artists where artist_name = 'Drake' or artist_name =
'Kayne West'

-- 5: Genre Comparison; example = Country, Rock, Pop
select * from genres where genre = 'Pop' or genre = 'Country' or
genre = 'Rock'
```

VIII. Major Data Questions

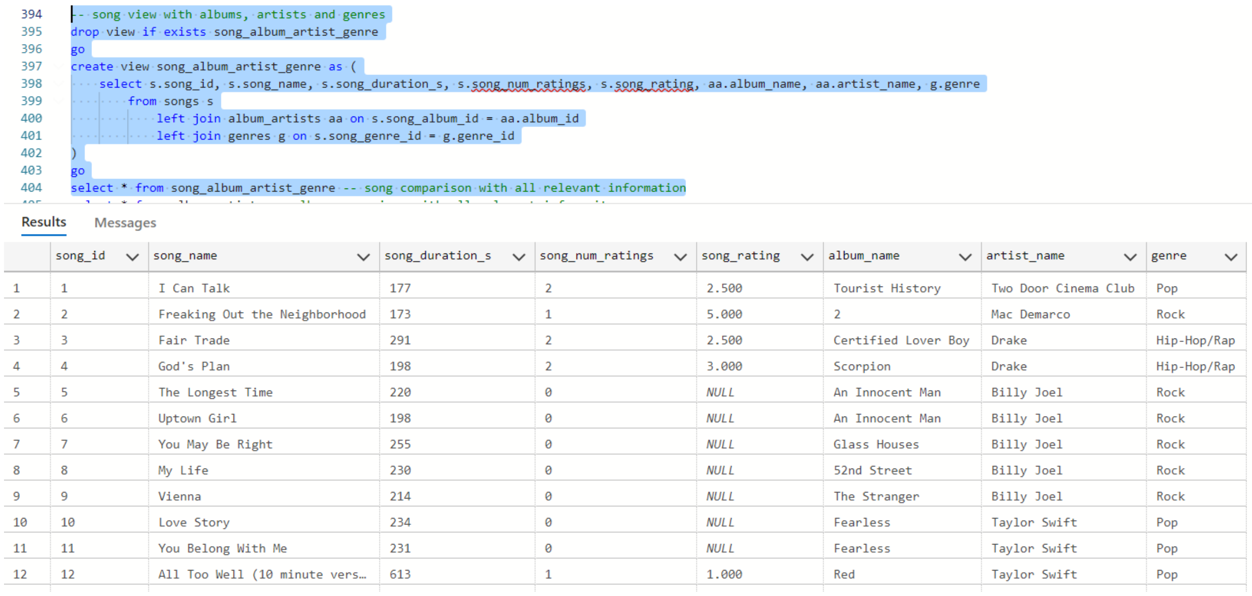
1. Music Recommendations. Depending on the user, they may want recommendations by song, album, artist or genre. But whatever their preference, *The People's Music* is the perfect platform to explore music once users give their input.



```
-- album view with artists
drop view if exists album_artists
go
create view album_artists as (
    select al.album_id, al.album_name, al.album_num_songs, al.album_rating, ar.artist_name
        from albums al
        left join artists ar on al.album_artist_id = ar.artist_id
)
go
select * from album_artists
```

Results Messages

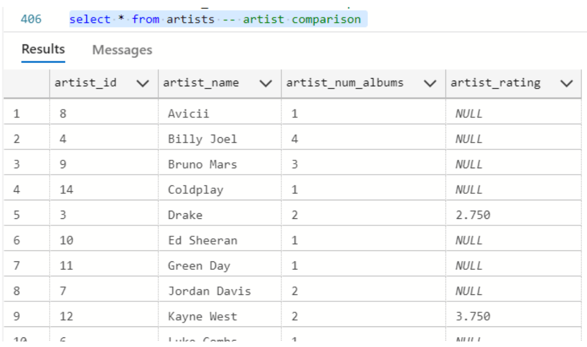
	album_id	album_name	album_num_songs	album_rating	artist_name
1	1	Tourist History	1	2.500	Two Door Cinema Club
2	2	2	1	5.000	Mac Demarco
3	3	Certified Lover Boy	1	2.500	Drake
4	4	Scorpion	1	3.000	Drake
5	5	An Innocent Man	2	*NULL*	Billy Joel
6	6	Glass Houses	1	*NULL*	Billy Joel
7	7	52nd Street	1	*NULL*	Billy Joel
8	8	The Stranger	1	*NULL*	Billy Joel
9	9	Fearless	2	*NULL*	Taylor Swift
10	10	Red	4	1.000	Taylor Swift



```
-- song view with albums, artists and genres
drop view if exists song_album_artist_genre
go
create view song_album_artist_genre as (
    select s.song_id, s.song_name, s.song_duration_s, s.song_num_ratings, s.song_rating, aa.album_name, aa.artist_name, g.genre
        from songs s
            left join album_artists aa on s.song_album_id = aa.album_id
            left join genres g on s.song_genre_id = g.genre_id
)
go
select * from song_album_artist_genre -- song comparison with all relevant information
```

Results Messages

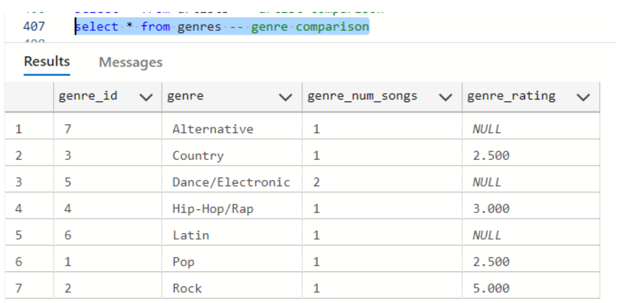
	song_id	song_name	song_duration_s	song_num_ratings	song_rating	album_name	artist_name	genre
1	1	I Can Talk	177	2	2.500	Tourist History	Two Door Cinema Club	Pop
2	2	Freaking Out the Neighborhood	173	1	5.000	2	Mac Demarco	Rock
3	3	Fair Trade	291	2	2.500	Certified Lover Boy	Drake	Hip-Hop/Rap
4	4	God's Plan	198	2	3.000	Scorpion	Drake	Hip-Hop/Rap
5	5	The Longest Time	220	0	*NULL*	An Innocent Man	Billy Joel	Rock
6	6	Uptown Girl	198	0	*NULL*	An Innocent Man	Billy Joel	Rock
7	7	You May Be Right	255	0	*NULL*	Glass Houses	Billy Joel	Rock
8	8	My Life	230	0	*NULL*	52nd Street	Billy Joel	Rock
9	9	Vienna	214	0	*NULL*	The Stranger	Billy Joel	Rock
10	10	Love Story	234	0	*NULL*	Fearless	Taylor Swift	Pop
11	11	You Belong With Me	231	0	*NULL*	Fearless	Taylor Swift	Pop
12	12	All Too Well (10 minute vers…	613	1	1.000	Red	Taylor Swift	Pop



```sql
select * from artists -- artist comparison
```

Results Messages

	artist_id	artist_name	artist_num_albums	artist_rating
1	8	Avicii	1	NULL
2	4	Billy Joel	4	NULL
3	9	Bruno Mars	3	NULL
4	14	Coldplay	1	NULL
5	3	Drake	2	2.750
6	10	Ed Sheeran	1	NULL
7	11	Green Day	1	NULL
8	7	Jordan Davis	2	NULL
9	12	Kayne West	2	3.750
10	6	Luke Combs	1	NULL

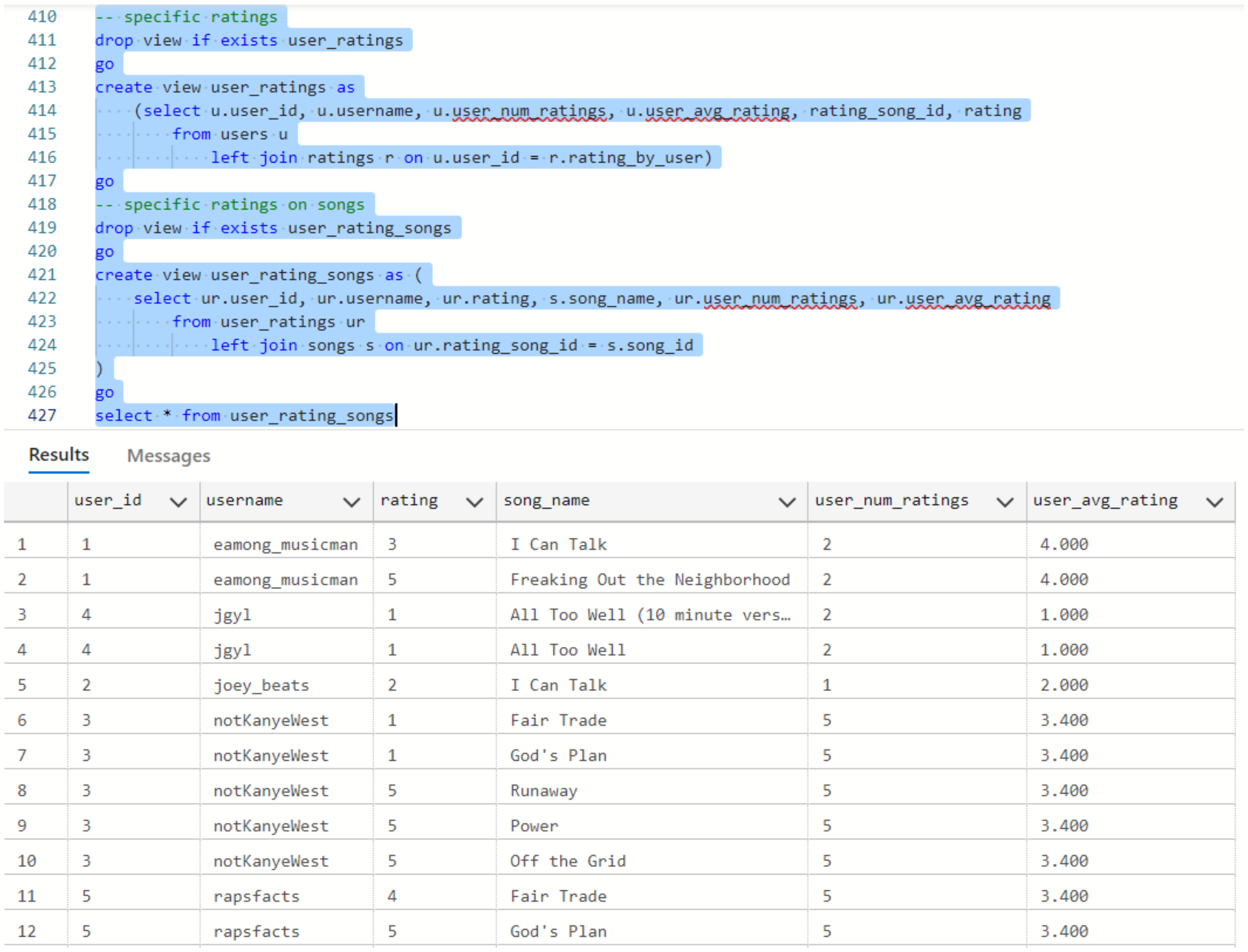


```sql
select * from genres -- genre comparison
```

Results Messages

	genre_id	genre	genre_num_songs	genre_rating
1	7	Alternative	1	NULL
2	3	Country	1	2.500
3	5	Dance/Electronic	2	NULL
4	4	Hip-Hop/Rap	1	3.000
5	6	Latin	1	NULL
6	1	Pop	1	2.500
7	2	Rock	1	5.000

2. Diverse Mix of Opinions. Some users may want to follow the ratings of others. Whether they would like to follow their friends are music industry icons, they can individual rating by each user in the database.

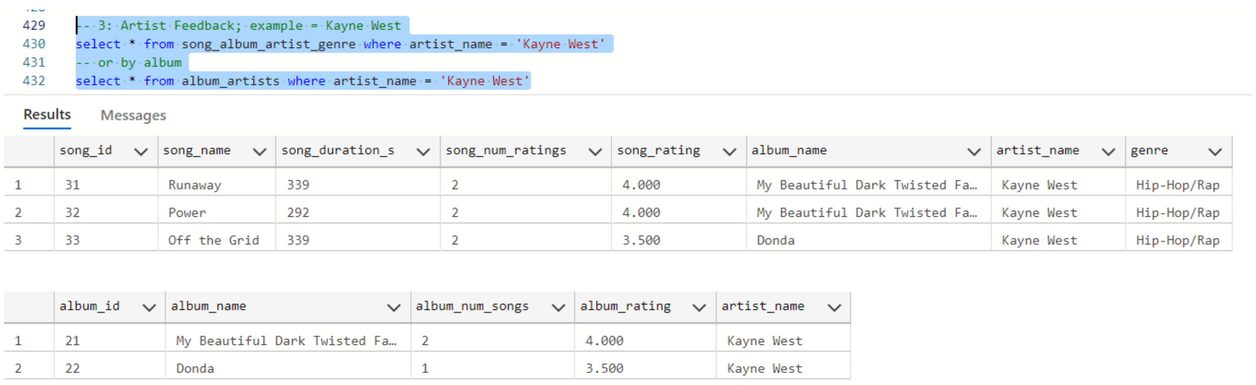


```sql
-- specific ratings
drop view if exists user_ratings
go
create view user_ratings as
    (select u.user_id, u.username, u.user_num_ratings, u.user_avg_rating, rating_song_id, rating
        from users u
            left join ratings r on u.user_id = r.rating_by_user)
go
-- specific ratings on songs
drop view if exists user_rating_songs
go
create view user_rating_songs as (
    select ur.user_id, ur.username, ur.rating, s.song_name, ur.user_num_ratings, ur.user_avg_rating
        from user_ratings ur
            left join songs s on ur.rating_song_id = s.song_id
)
go
select * from user_rating_songs
```

Results Messages

	user_id	username	rating	song_name	user_num_ratings	user_avg_rating
1	1	eamong_musicman	3	I Can Talk	2	4.000
2	1	eamong_musicman	5	Freaking Out the Neighborhood	2	4.000
3	4	jgyl	1	All Too Well (10 minute vers…	2	1.000
4	4	jgyl	1	All Too Well	2	1.000
5	2	joey_beats	2	I Can Talk	1	2.000
6	3	notKanyeWest	1	Fair Trade	5	3.400
7	3	notKanyeWest	1	God's Plan	5	3.400
8	3	notKanyeWest	5	Runaway	5	3.400
9	3	notKanyeWest	5	Power	5	3.400
10	3	notKanyeWest	5	Off the Grid	5	3.400
11	5	rapsfacts	4	Fair Trade	5	3.400
12	5	rapsfacts	5	God's Plan	5	3.400

3. Feedback for Artists. *The People's Music* is not limited to public ratings. Artists can also view the ratings of all of their music to compare their songs and albums to know what's hot and what's not.



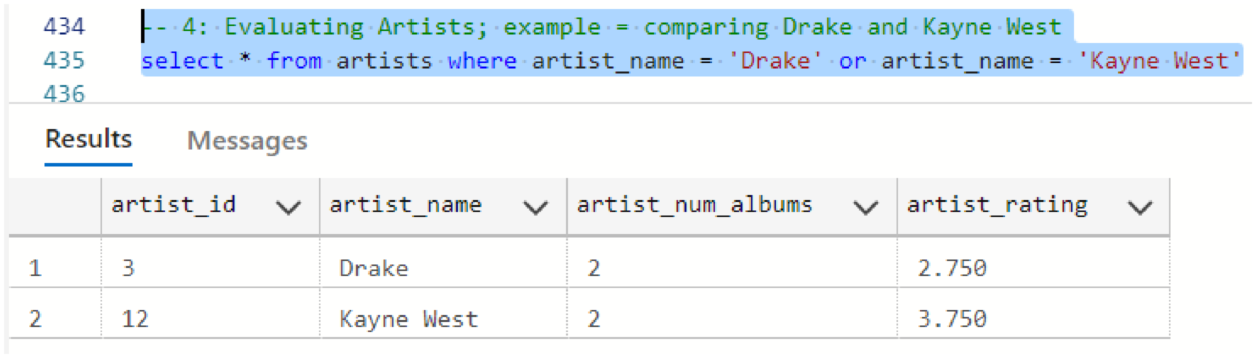
```sql
-- 3: Artist Feedback; example = Kayne West
select * from song_album_artist_genre where artist_name = 'Kayne West'
-- or by album
select * from album_artists where artist_name = 'Kayne West'
```

Results Messages

	song_id	song_name	song_duration_s	song_num_ratings	song_rating	album_name	artist_name	genre
1	31	Runaway	339	2	4.000	My Beautiful Dark Twisted Fa…	Kayne West	Hip-Hop/Rap
2	32	Power	292	2	4.000	My Beautiful Dark Twisted Fa…	Kayne West	Hip-Hop/Rap
3	33	Off the Grid	339	2	3.500	Donda	Kayne West	Hip-Hop/Rap

	album_id	album_name	album_num_songs	album_rating	artist_name
1	21	My Beautiful Dark Twisted Fa…	2	4.000	Kayne West
2	22	Donda	1	3.500	Kayne West

4. Centralized Rankings. Artist popularity can now be directly calculated from user ratings. *The People's Music* provides a competitive environment for musicians who are searching to make the best music.

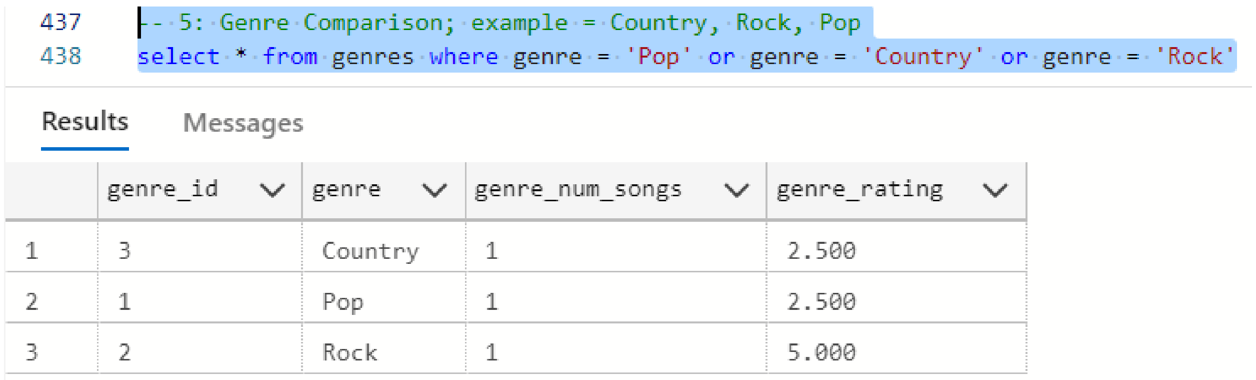


```sql
-- 4: Evaluating Artists; example = comparing Drake and Kayne West
select * from artists where artist_name = 'Drake' or artist_name = 'Kayne West'
```

Results Messages

	artist_id	artist_name	artist_num_albums	artist_rating
1	3	Drake	2	2.750
2	12	Kayne West	2	3.750

5. Genre Comparison. Never before has there been a database to determine which genres are best. Now, with *The People's Music*, music listeners can settle their debates based on the world's feedback.



```sql
-- 5: Genre Comparison; example = Country, Rock, Pop
select * from genres where genre = 'Pop' or genre = 'Country' or genre = 'Rock'
```

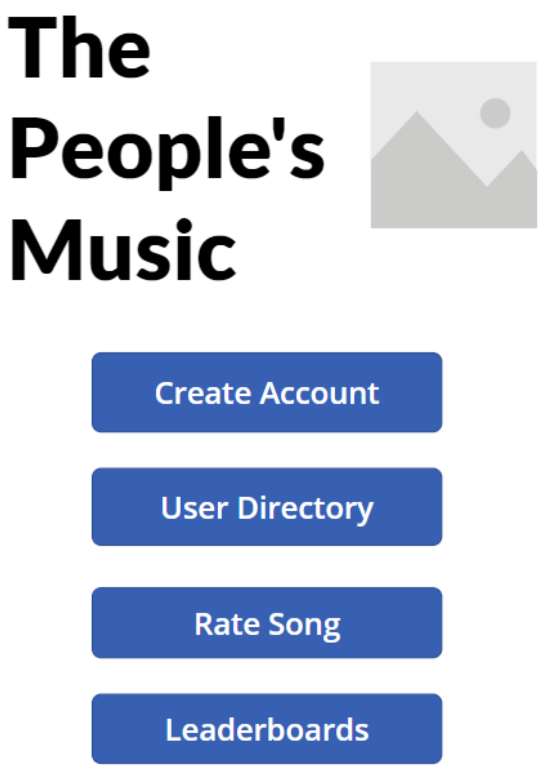
Results Messages

	genre_id	genre	genre_num_songs	genre_rating
1	3	Country	1	2.500
2	1	Pop	1	2.500
3	2	Rock	1	5.000

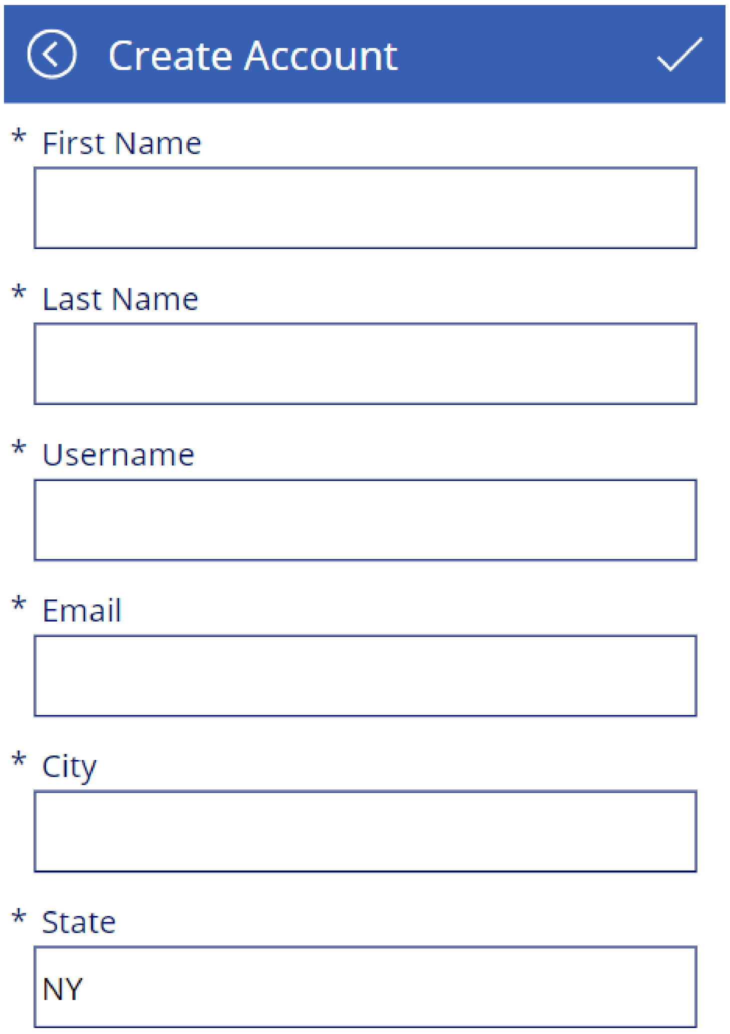
Note: These examples will not be accurate until the database gets a significant amount of user feedback!

IX. Application Screens

1. Main Menu



2. Create Account



Create Account

* First Name

* Last Name

* Username

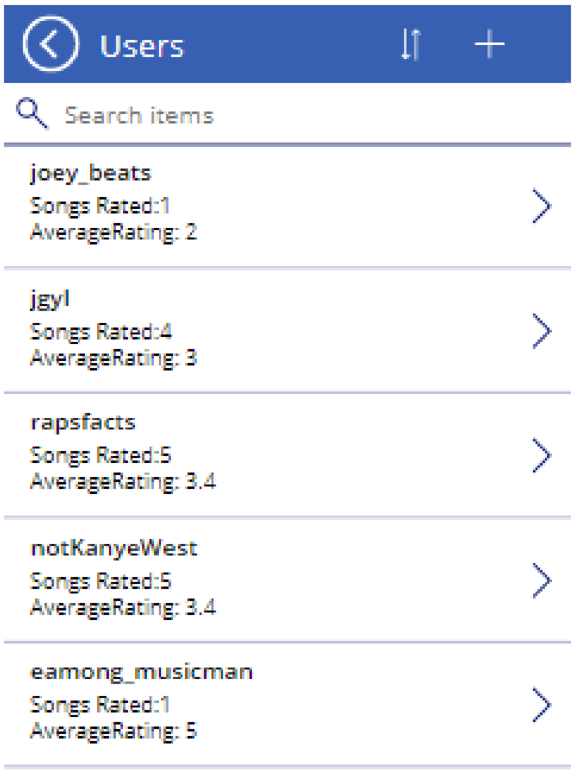
* Email

* City

* State

NY

3. Users



joey_beats
Songs Rated:1
AverageRating: 2

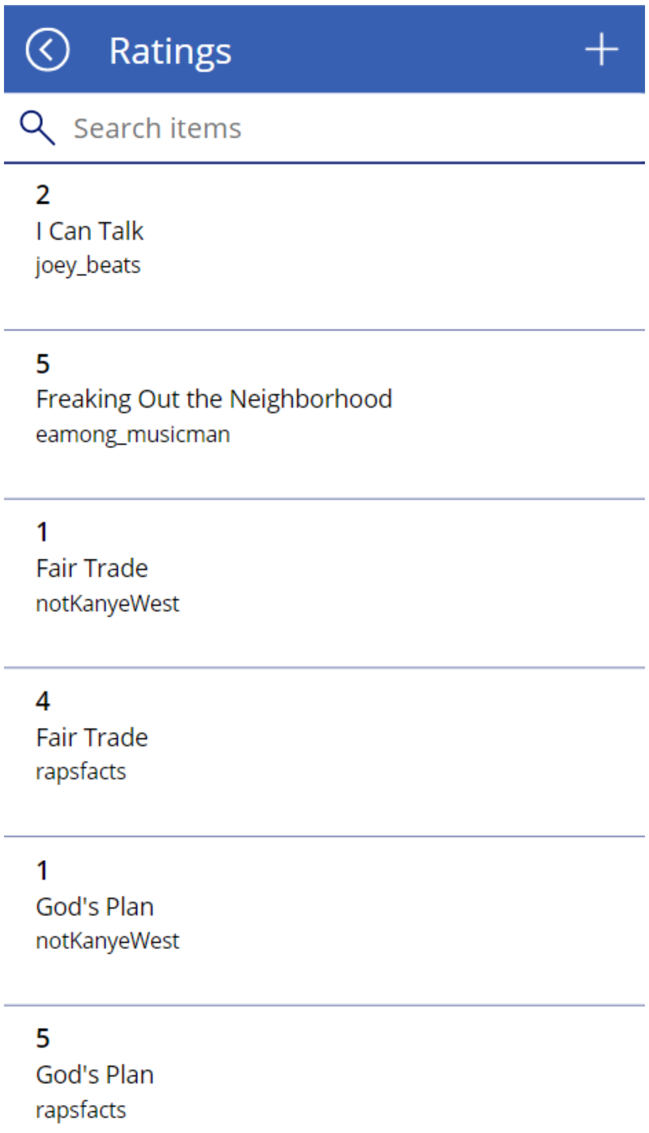
jgyl
Songs Rated:4
AverageRating: 3

rapsfacts
Songs Rated:5
AverageRating: 3.4

notKanyeWest
Songs Rated:5
AverageRating: 3.4

eamong_musicman
Songs Rated:1
AverageRating: 5

4. Ratings Page



2
I Can Talk
joey_beats

5
Freaking Out the Neighborhood
eamong_musicman

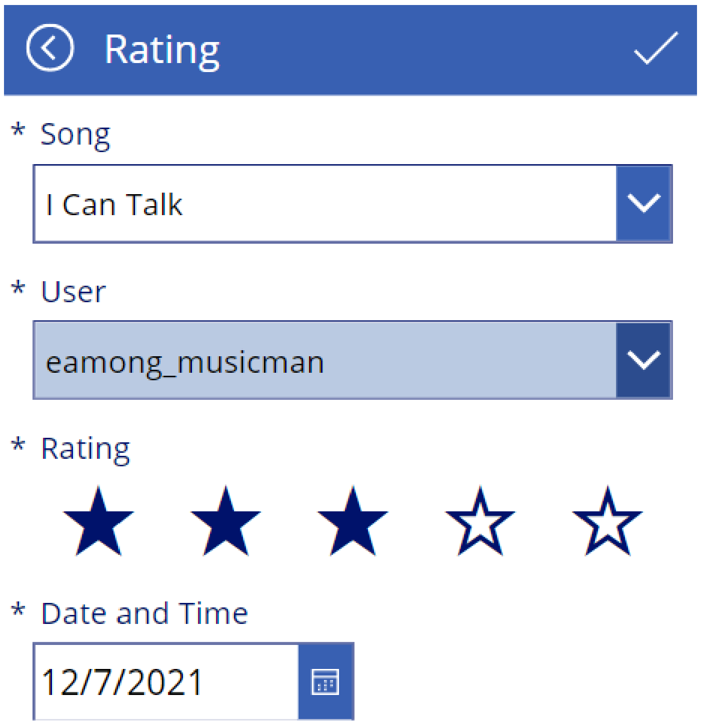
1
Fair Trade
notKanyeWest

4
Fair Trade
rapsfacts

1
God's Plan
notKanyeWest

5
God's Plan
rapsfacts

5. New Ratings Page



Rating

* Song

I Can Talk

* User

eamong_musicman

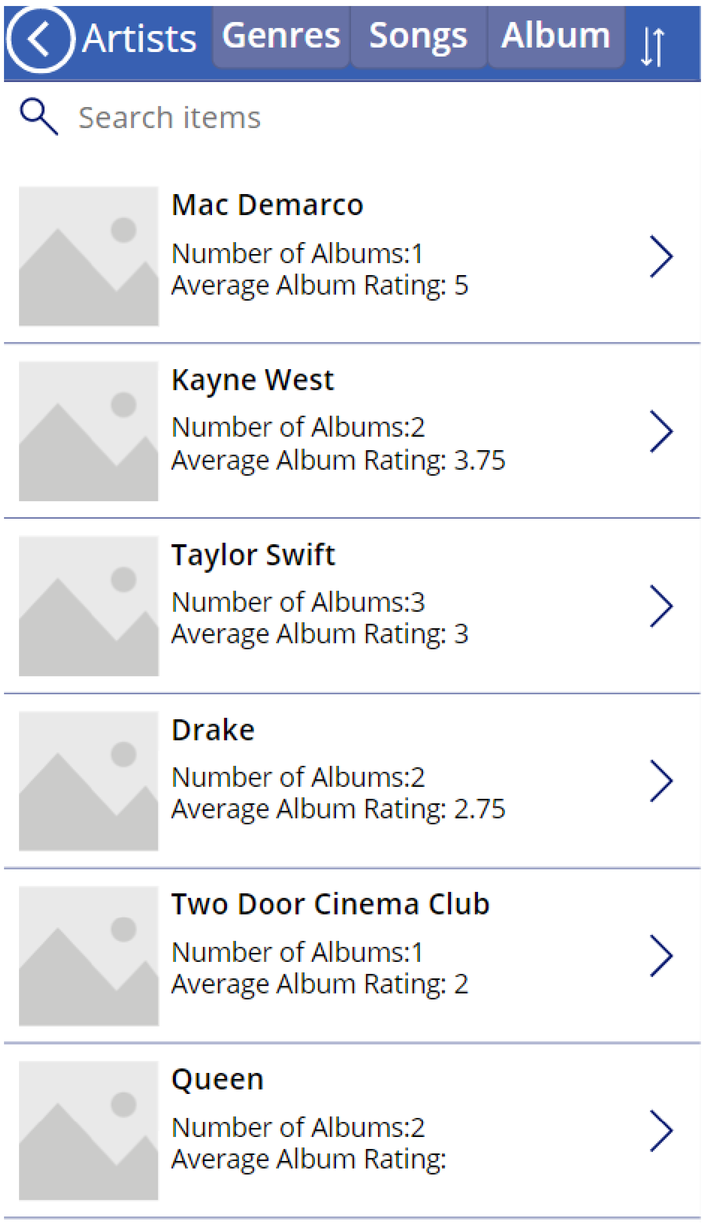
* Rating

★ ★ ★ ☆ ☆

* Date and Time

12/7/2021

6. Artists Leaderboard



Mac Demarco
Number of Albums:1
Average Album Rating: 5

Kayne West
Number of Albums:2
Average Album Rating: 3.75

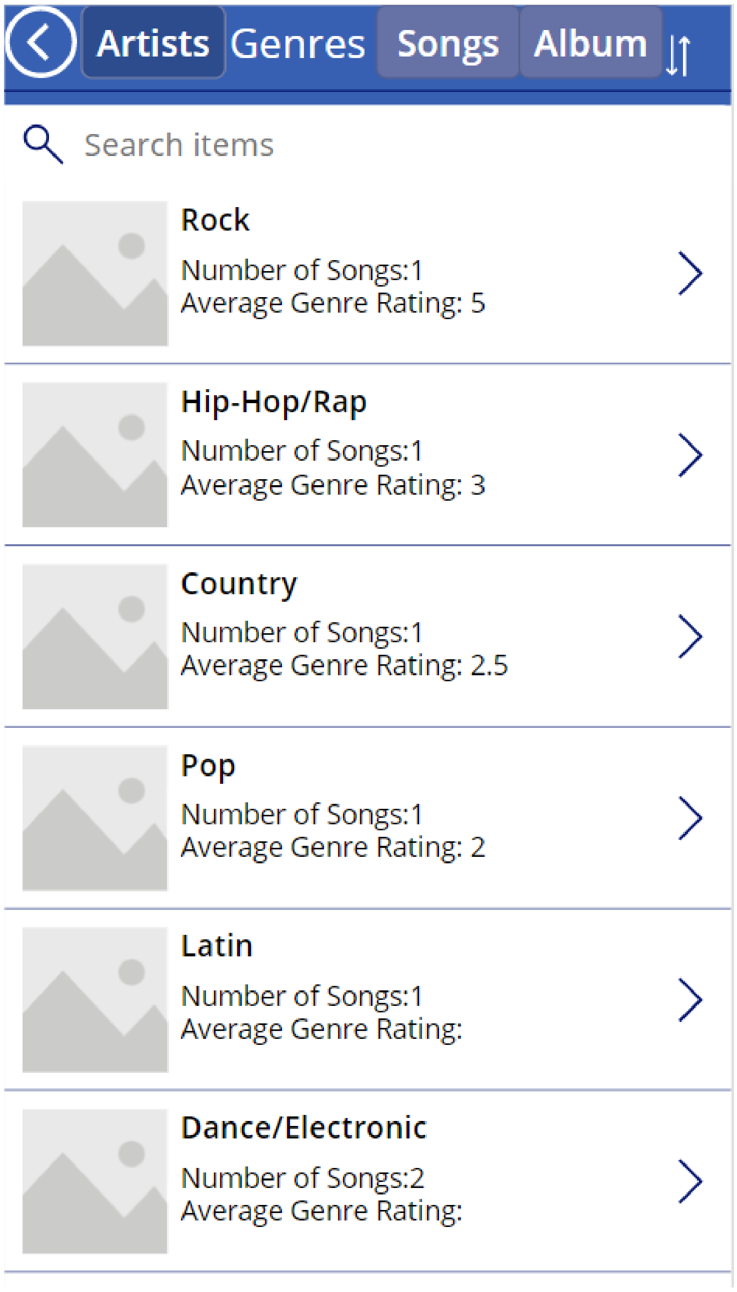
Taylor Swift
Number of Albums:3
Average Album Rating: 3

Drake
Number of Albums:2
Average Album Rating: 2.75

Two Door Cinema Club
Number of Albums:1
Average Album Rating: 2

Queen
Number of Albums:2
Average Album Rating:

7. Genres Leaderboard



Rock
Number of Songs:1
Average Genre Rating: 5

Hip-Hop/Rap
Number of Songs:1
Average Genre Rating: 3

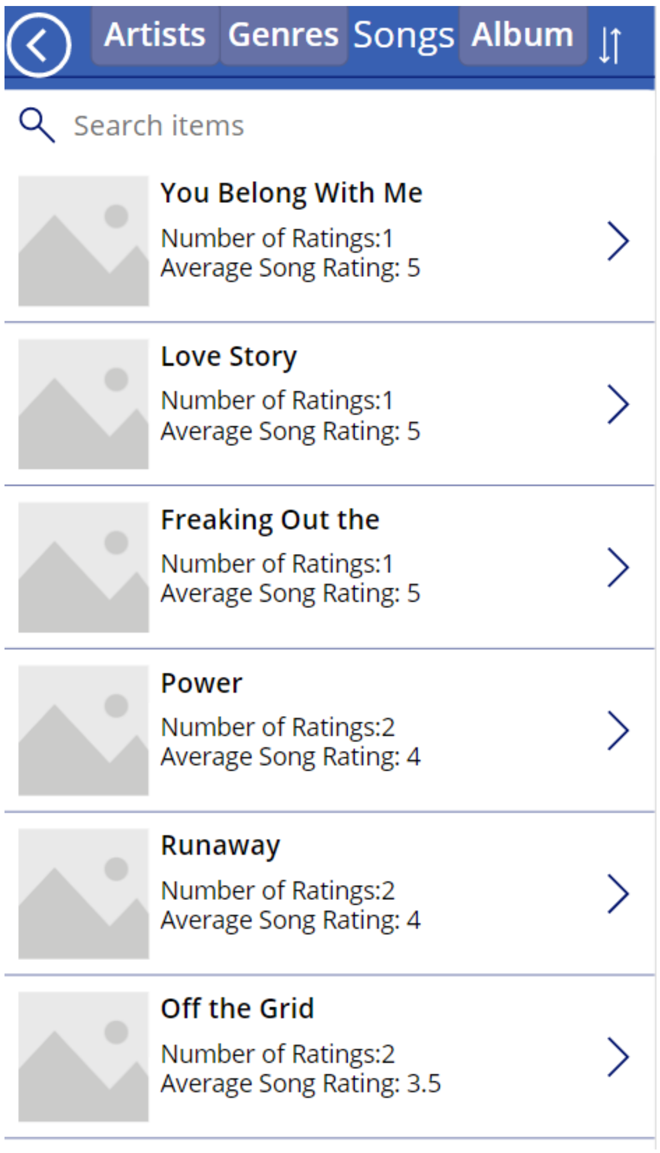
Country
Number of Songs:1
Average Genre Rating: 2.5

Pop
Number of Songs:1
Average Genre Rating: 2

Latin
Number of Songs:1
Average Genre Rating:

Dance/Electronic
Number of Songs:2
Average Genre Rating:

8. Songs Leaderboard



You Belong With Me
Number of Ratings:1
Average Song Rating: 5

Love Story
Number of Ratings:1
Average Song Rating: 5

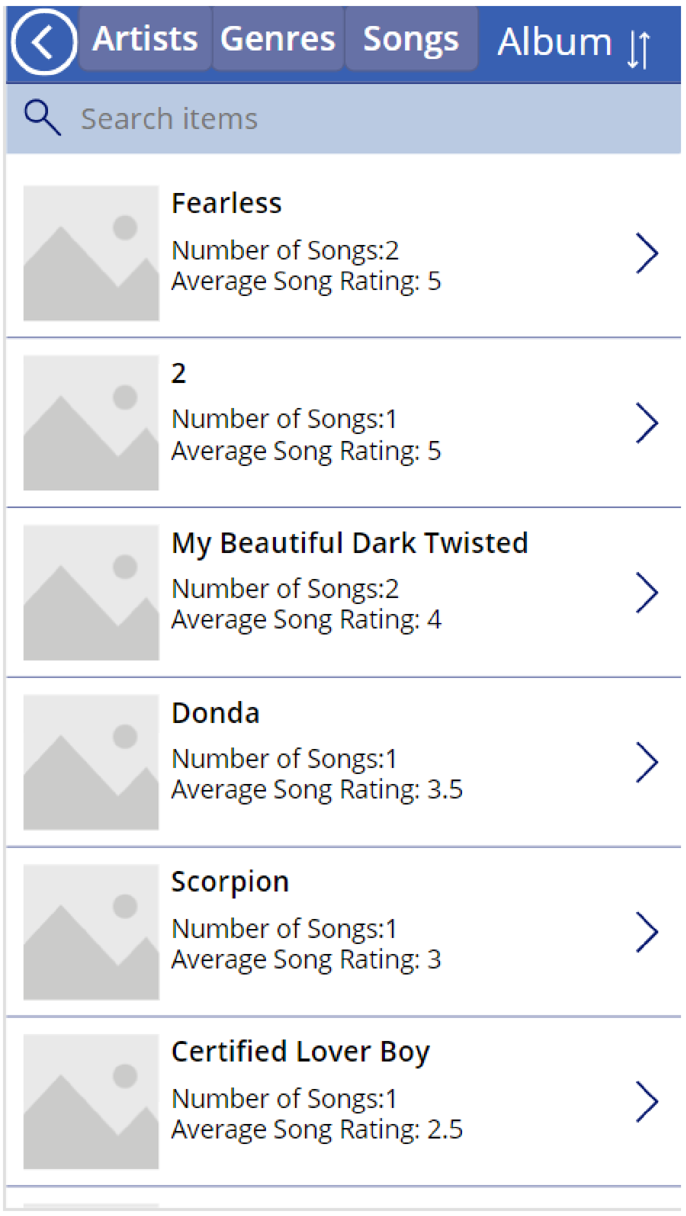
Freaking Out the
Number of Ratings:1
Average Song Rating: 5

Power
Number of Ratings:2
Average Song Rating: 4

Runaway
Number of Ratings:2
Average Song Rating: 4

Off the Grid
Number of Ratings:2
Average Song Rating: 3.5

9. Albums Leaderboard



Fearless
Number of Songs:2
Average Song Rating: 5

2
Number of Songs:1
Average Song Rating: 5

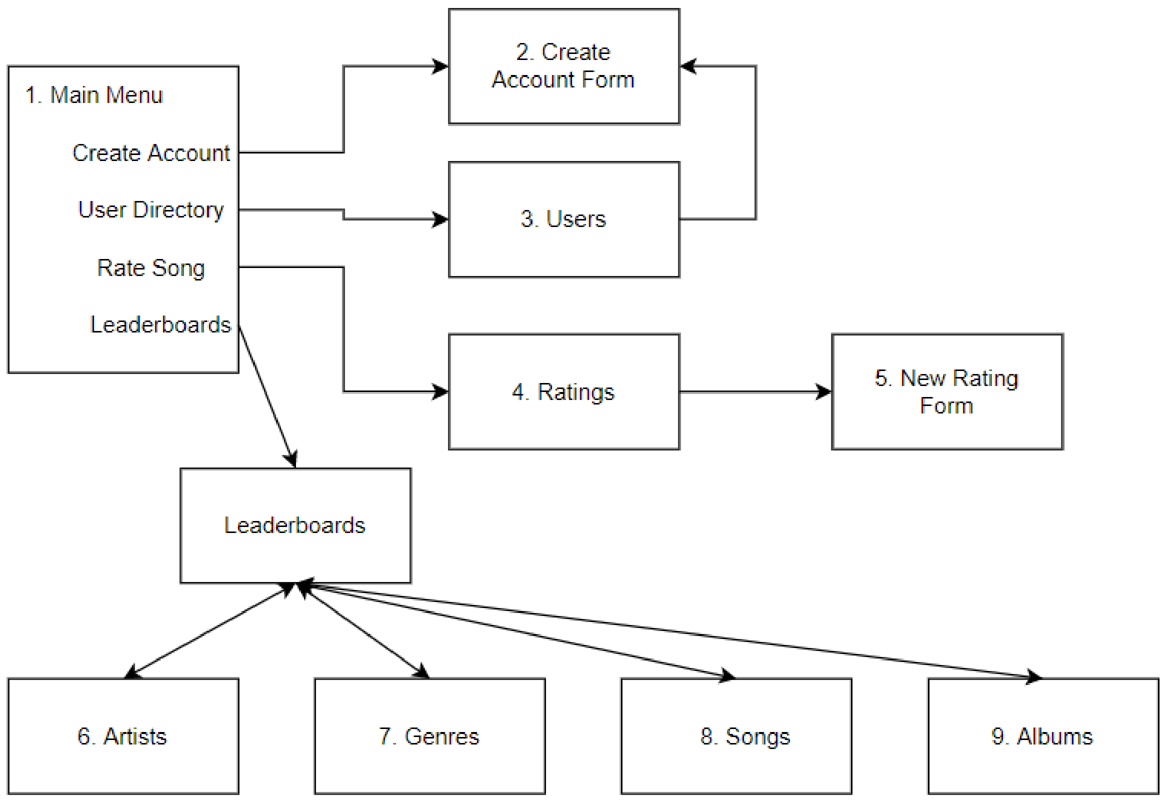
My Beautiful Dark Twisted
Number of Songs:2
Average Song Rating: 4

Donda
Number of Songs:1
Average Song Rating: 3.5

Scorpion
Number of Songs:1
Average Song Rating: 3

Certified Lover Boy
Number of Songs:1
Average Song Rating: 2.5

X. Screen Diagram



XI. Major Data Questions in the App

1. Music Recommendations. To avoid redundancy, we will not repost the screens used for our first major question. Instead, we instruct the reader to look at screens 6-9 and examine our artist, genre, song and album leaderboards for the recommendations.
2. Diverse Mix of Opinions. Similar to the music recommendations, user opinions can be found on screen 4. Here, you can see all of the ratings made by each user, ordered by song id.
3. Feedback for artists (example: Kanye West). Using screens 8 and 9, one can search "Kanye West" on the search bar and get all of the related songs and albums. From here, you can view each of their ratings and number of ratings.
4. Comparing Artists. Using screen 6, one may search for any artists that they want. Once the average ratings and number of ratings are viewed, the user can conclude which artist has better ratings from *The People's Music*.
5. Comparing Genres. Using Screen 7, one may search for any genres that they want. Once the average ratings and number of ratings are viewed, the user can conclude which genre has better ratings from *The People's Music.*

XII. Future Explorations

Although *The People's Music* covers much of what it was set out to do, there are many other factors that can be added to improve the application implementation and database. In future updates, our team would like to incorporate several ideas that would elevate the value of *The People's Music*.

With separate artists and songs, future updates will include a featured artist column in the songs table. Taken directly from the database, users can see their favorite artists featured in other songs.

Because our database is in the beginning stages of development, we focused on the ratings system for our physical model. Consequently, our team would like to add the option for users to add songs, artists and albums to the database. This way, artists of all kinds can get their content out there for the world to listen to.

Before we release *The People's Music*, it would be important to create a password for different users so their accounts can be secure. Users would have the option to enter their passwords in registration and it would be required each time they log in.

To help the experience on *The People's Music*, we would add group features to our app. This would provide a social media feature for groups of friends to rate similar music and share newer music with each other.

XII. References

- Song Lengths: https://en.wikipedia.org/
- PowerApp: https://apps.powerapps.com/play/f018660e-0b4a-443b-b8db-d061c20c0a03?source=portal