**The People’s Music**

Centralized Database for Reviews on Musical Pieces

IST 659 M004 Fall 2021

Eamon Gallagher, Kevin Harmer, & Emmanuel Victor Kamya

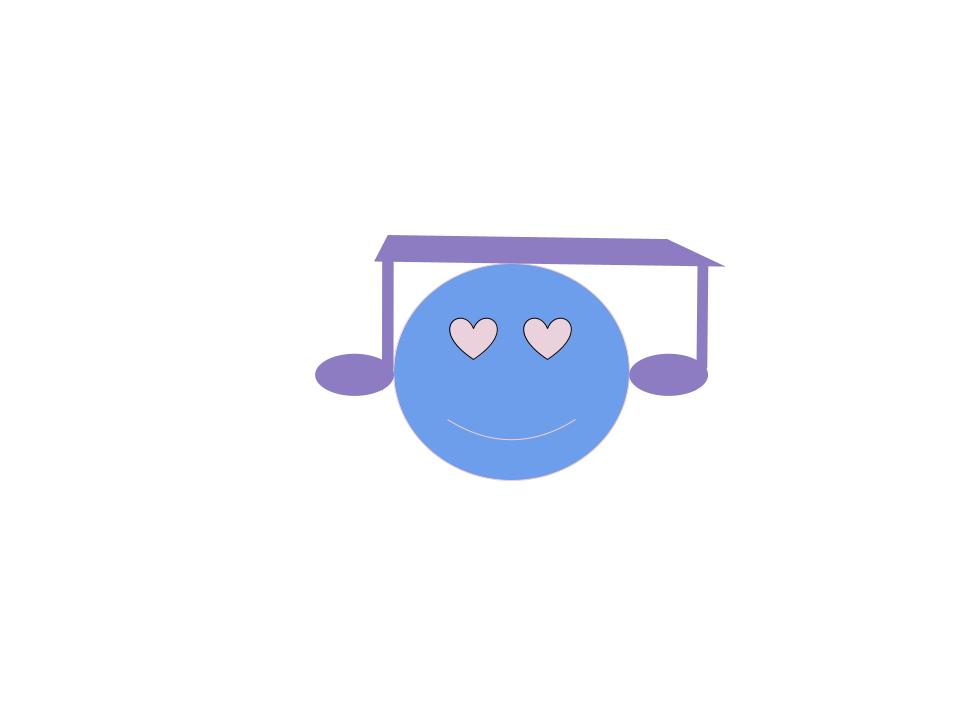
****

Table of Contents

1. Introduction …………………………………………………………………3
2. Problem Formulation ………………………………………………………..3
3. Relational Model ……………………………………………………………4
   1. Conceptual Model 4
   2. 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
14. 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.

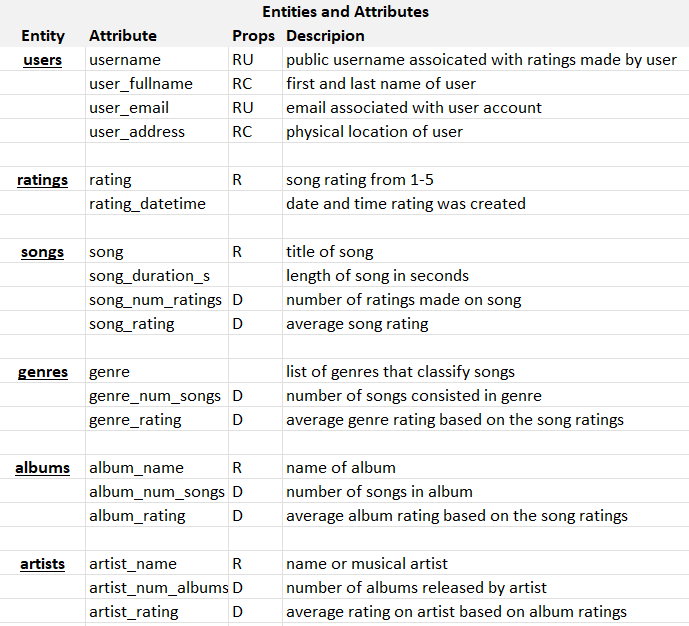
1. Problem Formulation

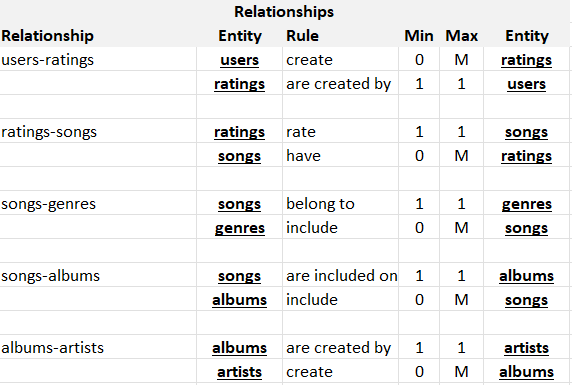
Because this database could evolve into several different things, our team formulated several problem solutions that could improve modern music listening. Due to *The People’s Music* being in the initial stages of development, we narrowed down our possible objectives to 5 distinct goals.

* 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.

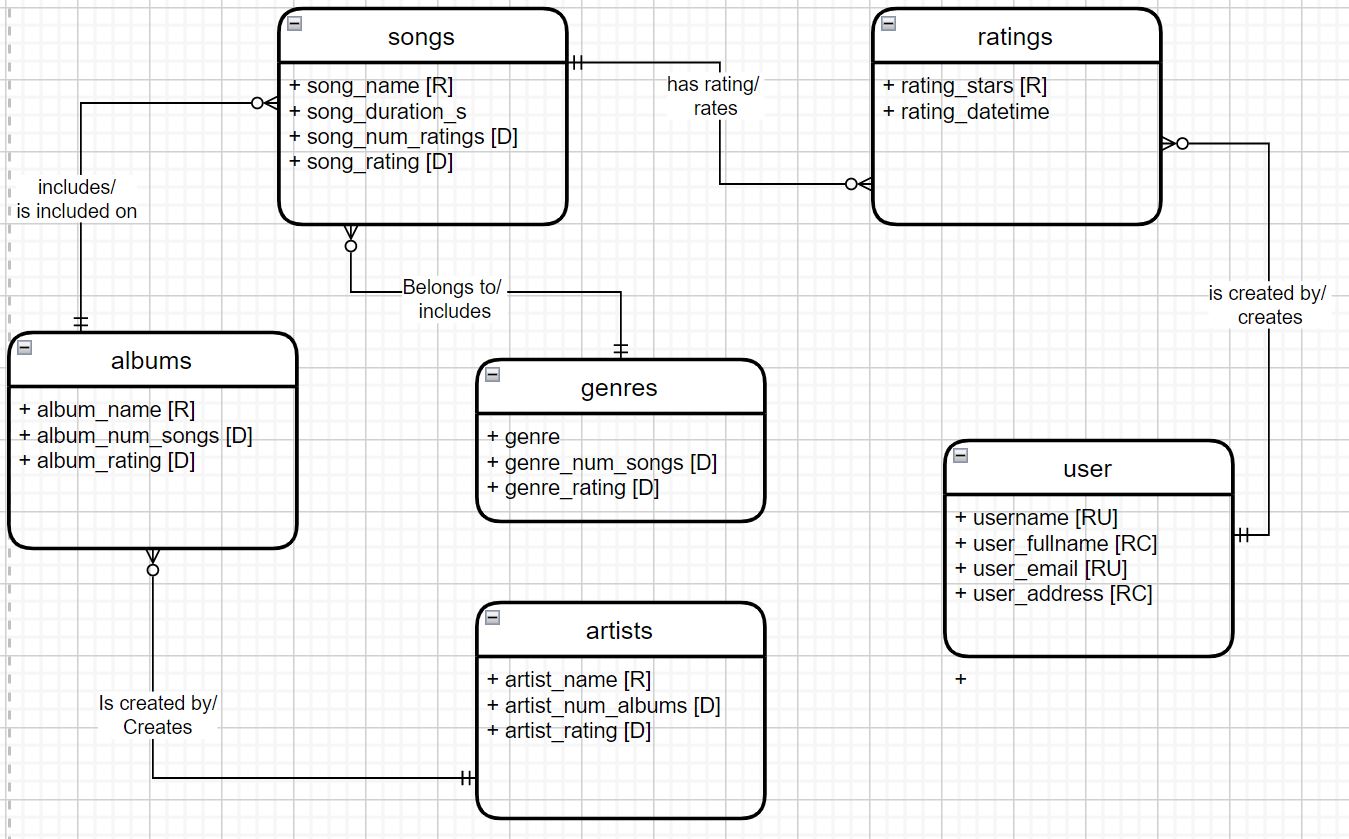
1. Relational Model
   1. Developing the Conceptual Model

ER Requirements:

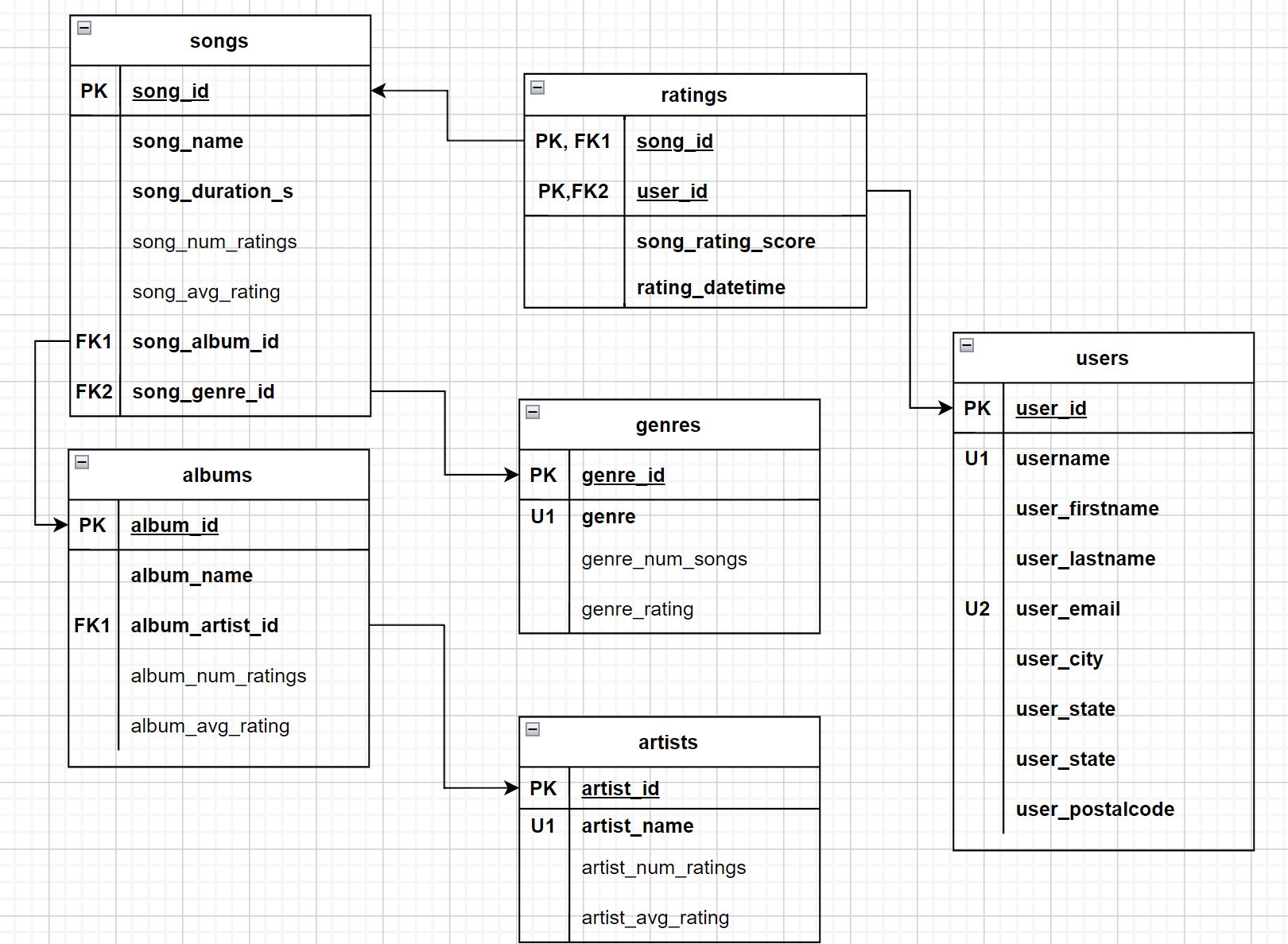




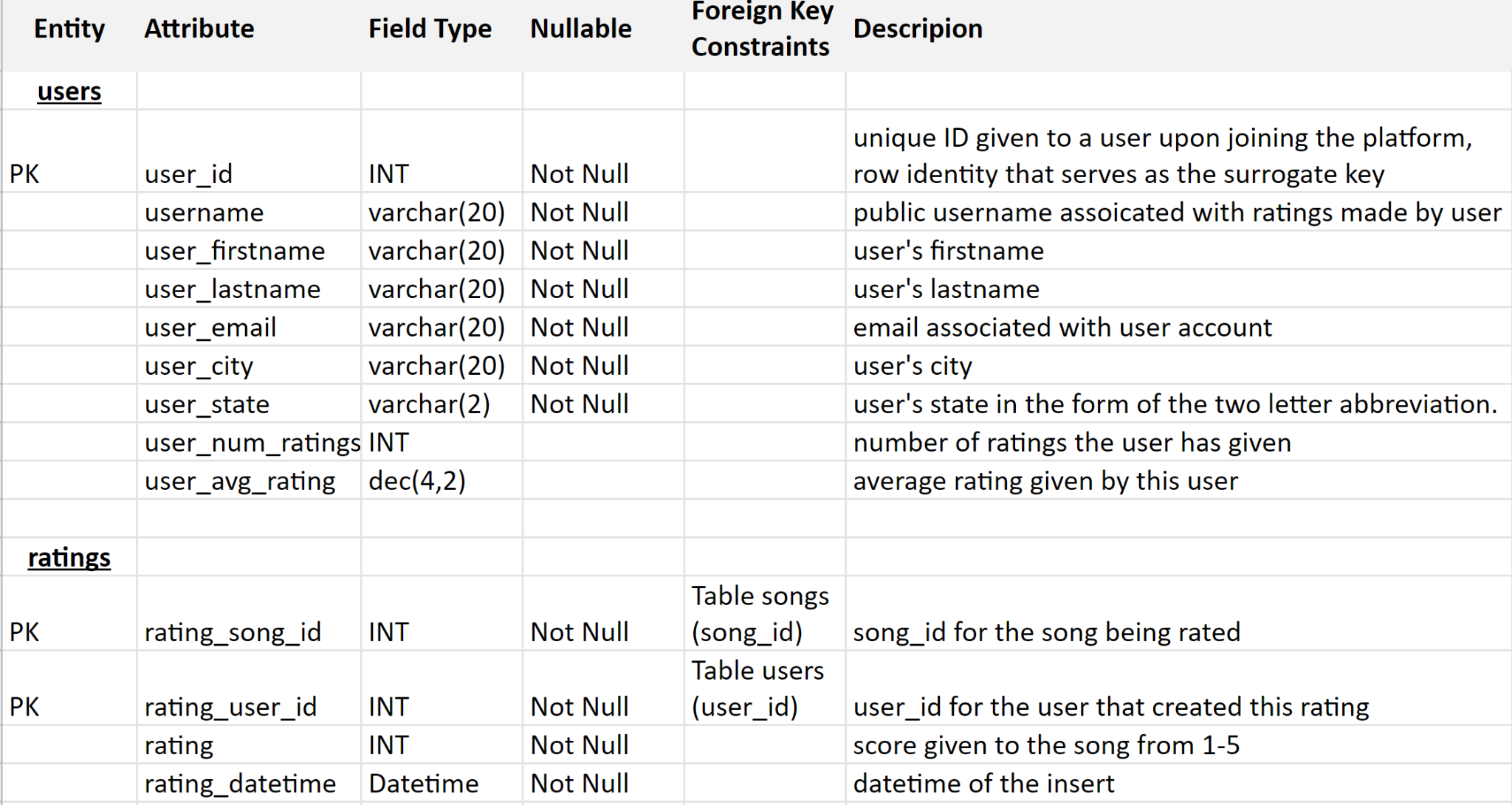
ERD Model:



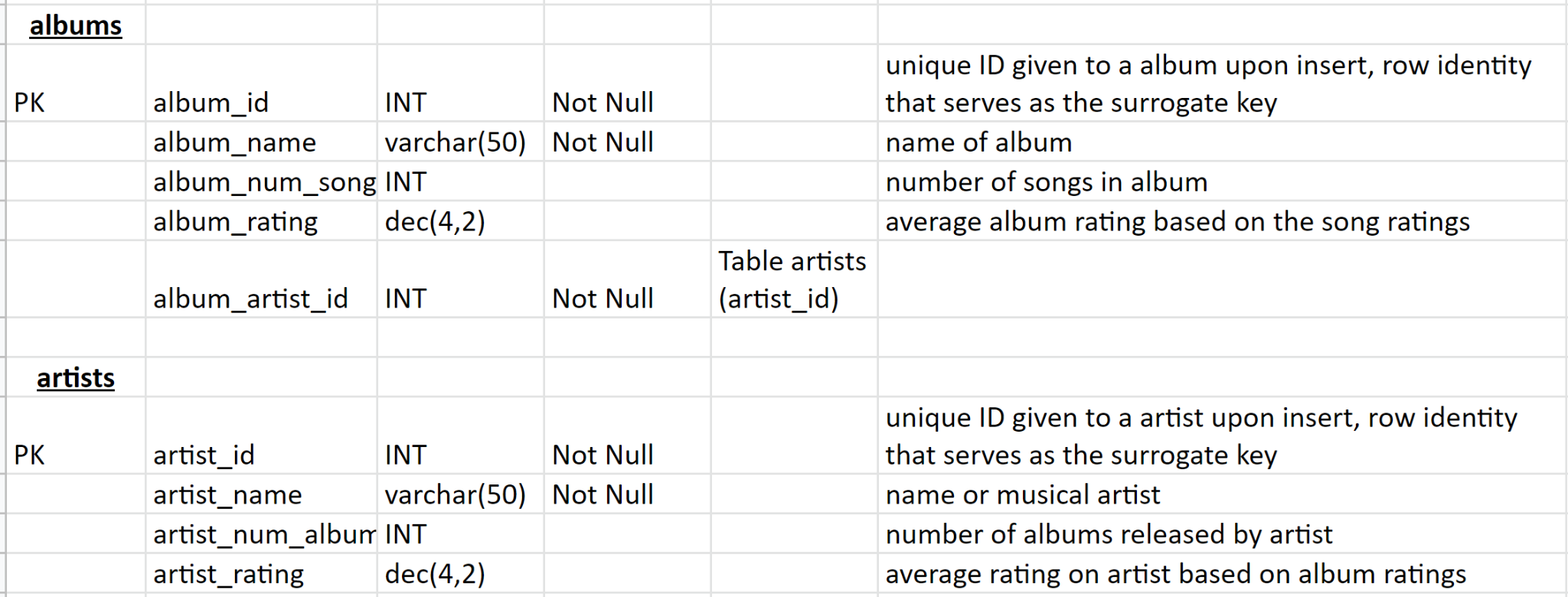
* 1. Logical Model



1. Data Dictionary







1. 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.

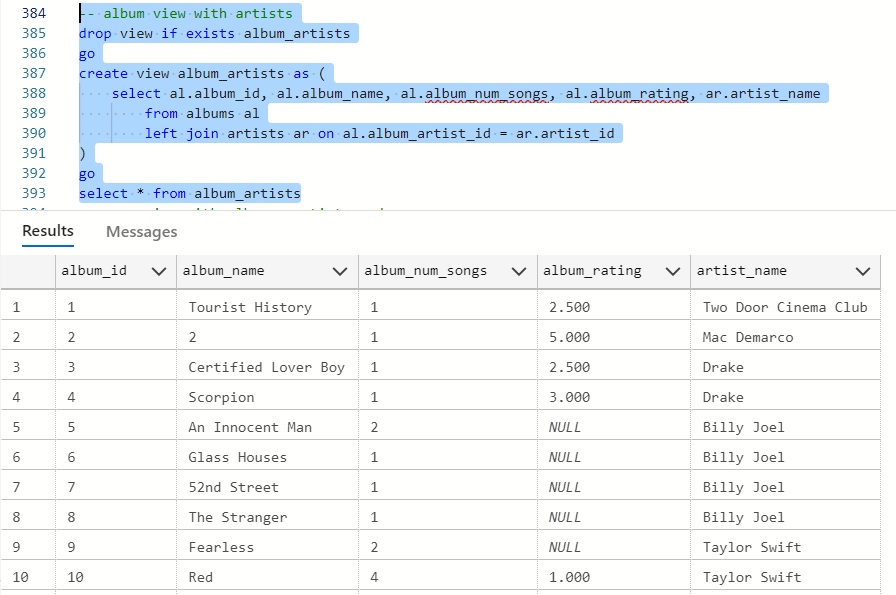
1. Data System Infrastructure

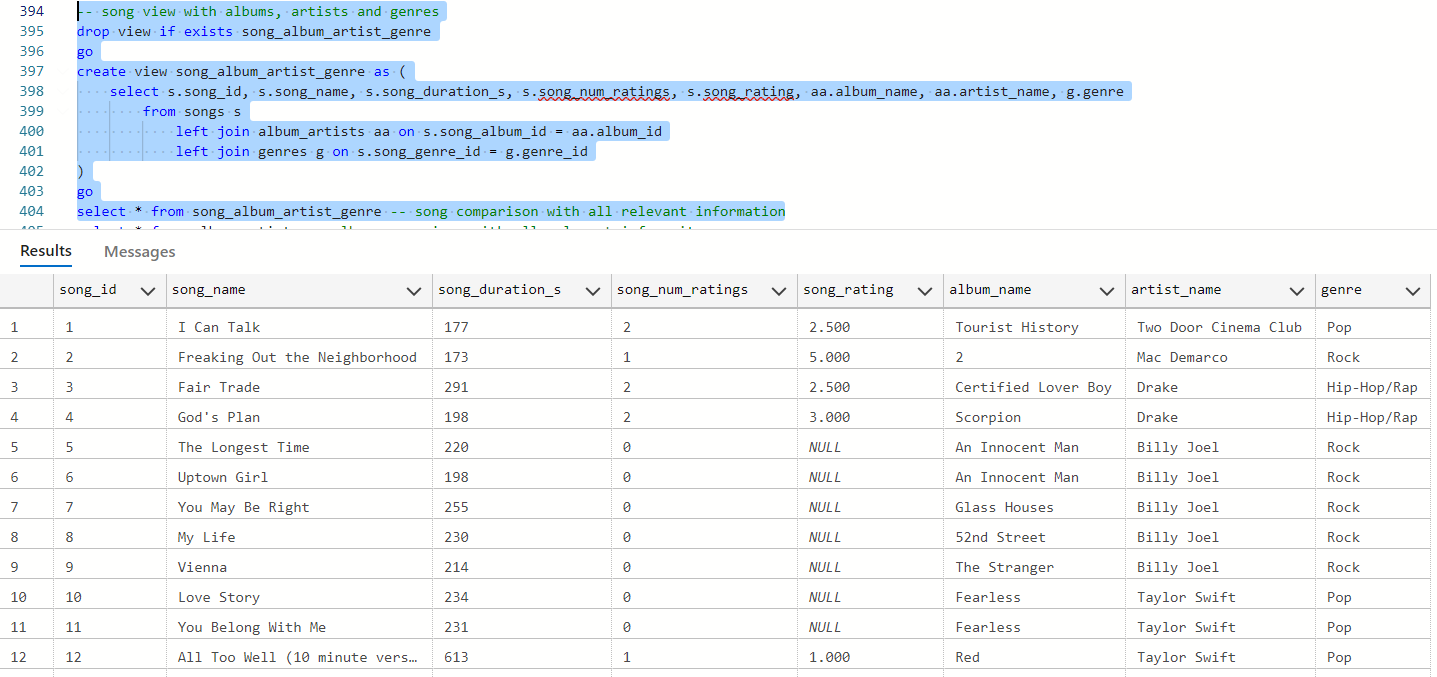
We used the following to tools to create and implement *The People’s Music*:

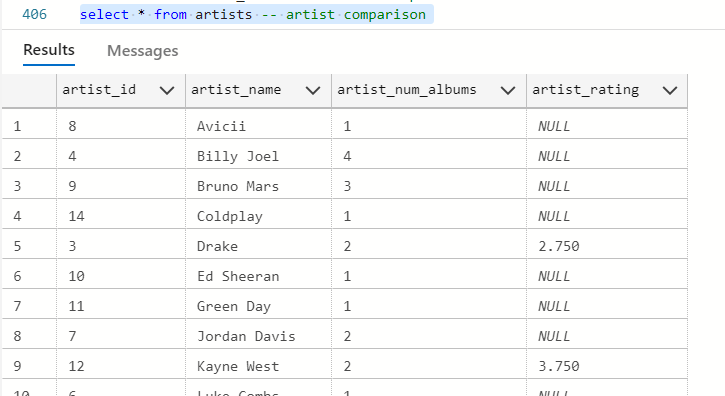
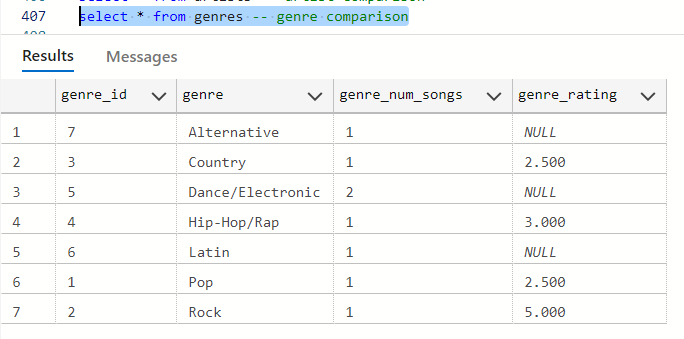
1. Microsoft Excel - Excel Spreadsheets were used to create and organize the ER requirements and the Data Dictionary
2. Draw.io - Drawings of the Logical and Conceptual Models were made in Draw.io
3. Azure Data Studio - The internal model (up/down script) was constructed in Azure Data Studio. Tables 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.
6. SQL Code

| 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  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' |
| --- |

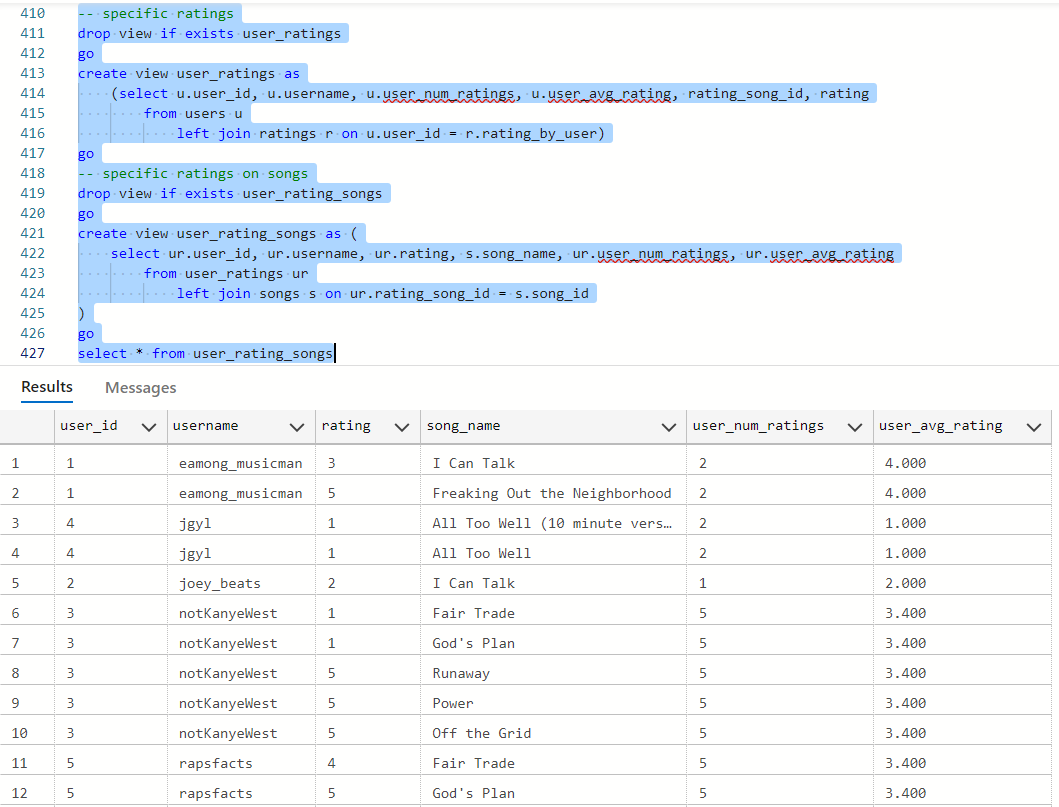
1. Major Data Questions
   1. Music Recommendations. Depending on the user, they may want recommendations by song, album, artist or genre. But whatever their preference, *The People’s Music* is the perfect platform to explore music once users give their input.



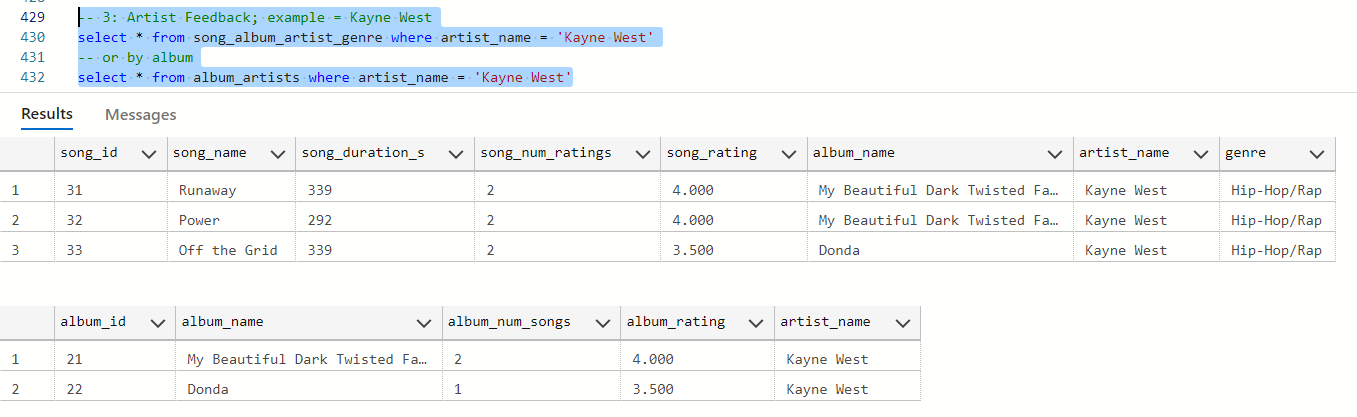


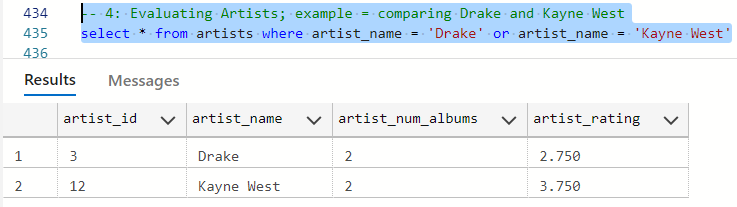
* 1. Diverse Mix of Opinions. Some users may want to follow the ratings of others. Whether they would like to follow their friends are music industry icons, they can individual rating by each user in the database.



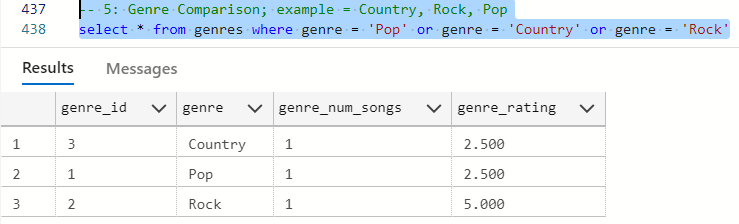
* 1. Feedback for Artists. *The People’s Music* is not limited to public ratings. Artists can also view the ratings of all of their music to compare their songs and albums to know what’s hot and what’s not.



* 1. 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.

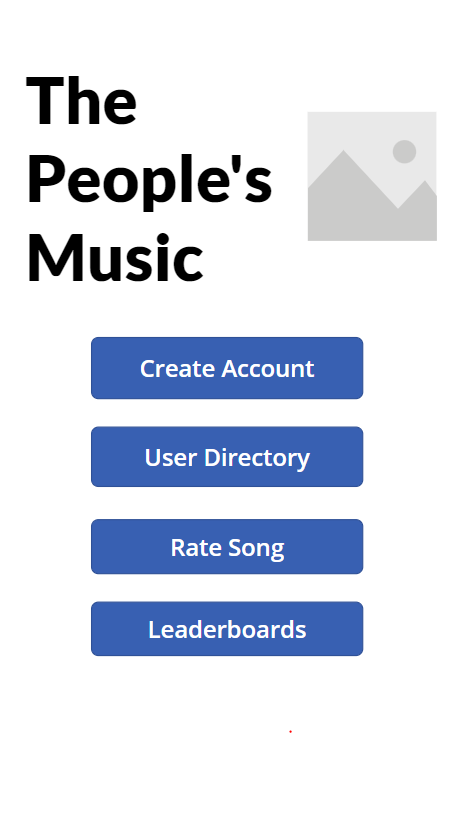


* 1. Genre Comparison. Never before has there been a database to determine which genres are best. Now, with *The People’s Music*, music listeners can settle their debates based on the world’s feedback.



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

1. Application Screens
   1. Main Menu



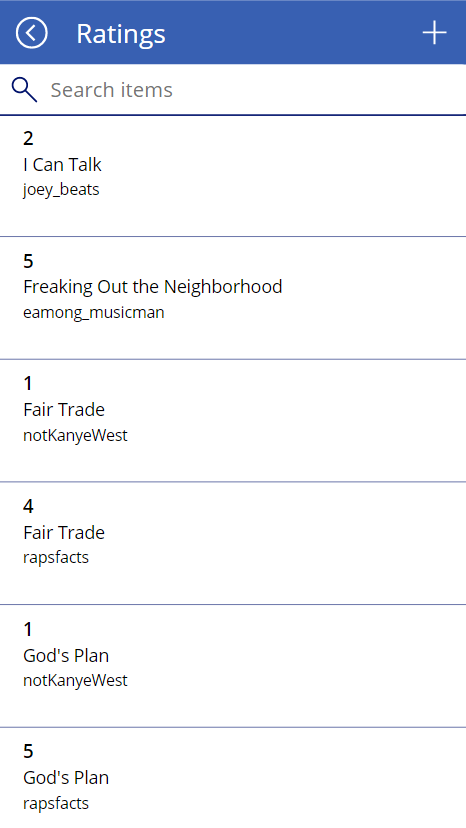
* 1. Create Account



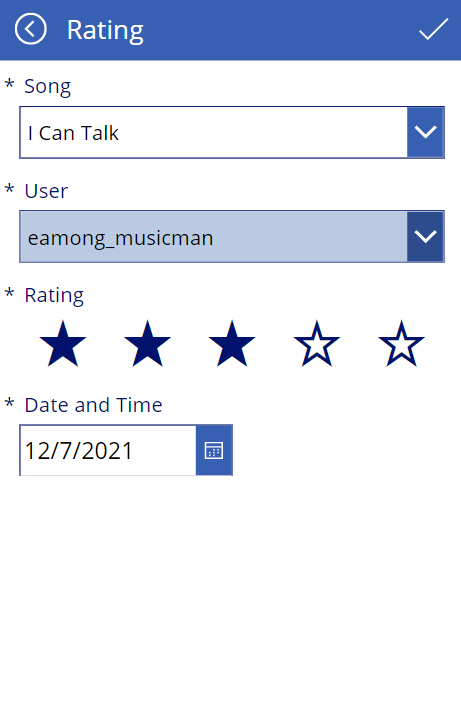
* 1. Users



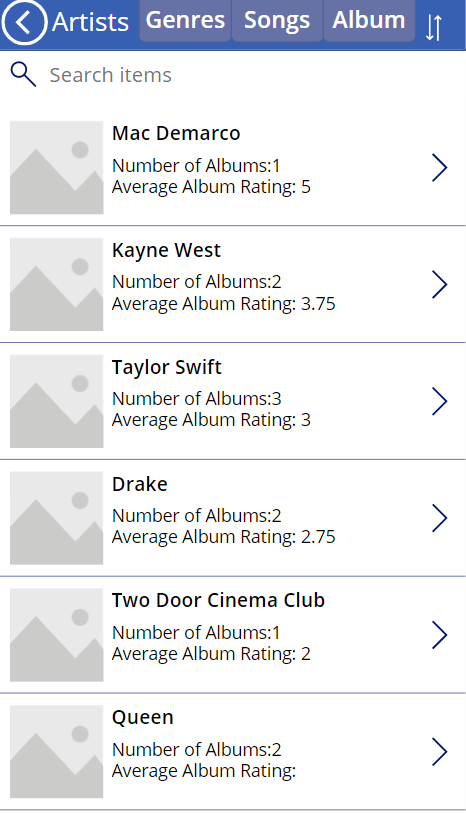
* 1. Ratings Page



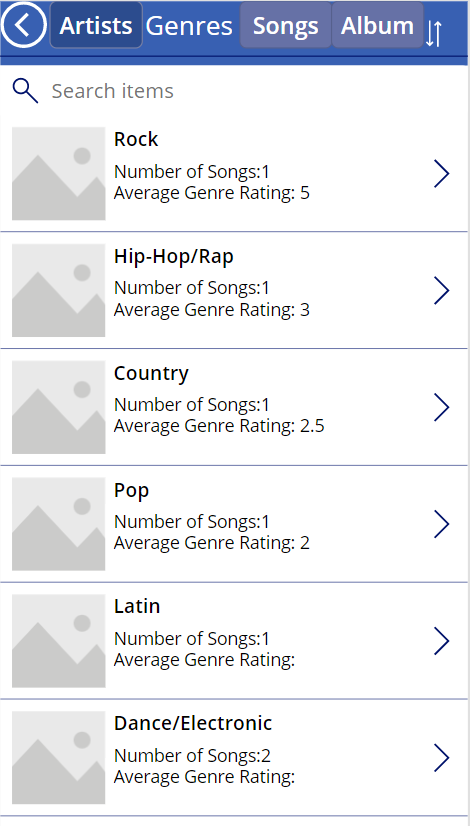
* 1. New Ratings Page



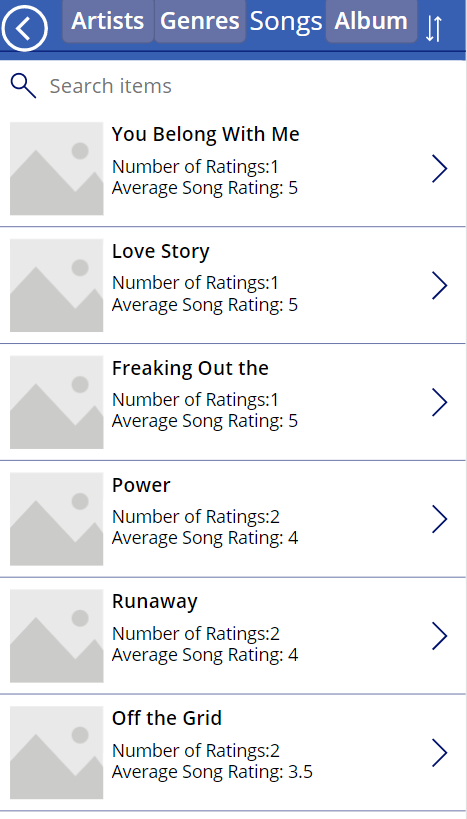
* 1. Artists Leaderboard



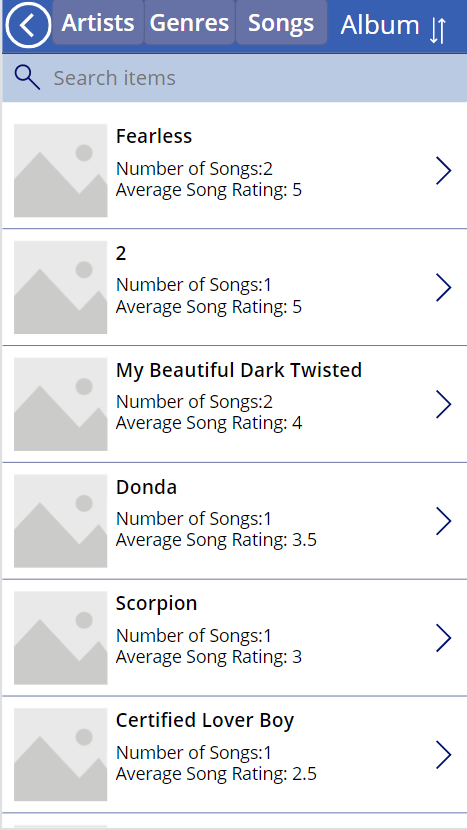
* 1. Genres Leaderboard



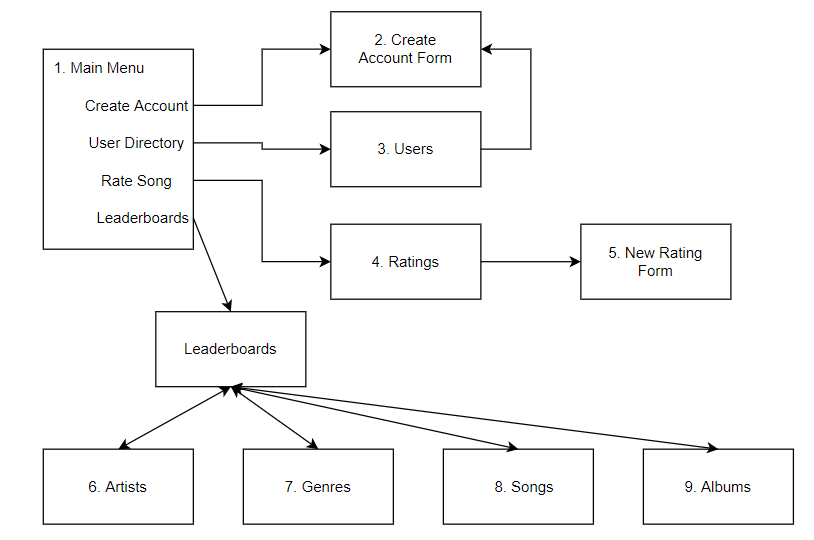
* 1. Songs Leaderboard



* 1. Albums Leaderboard



1. Screen Diagram



1. 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.*
2. 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.

1. References

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