Group Members: Lori Liu, Charles Richards, Raymond Hu, Johnny Cuevas

Group Number: 4

Group Name: Potify

Domain: Music

**Introduction**

Potify is a web application (if time permits) or a command line application. The application’s front-end languages include HTML, CSS, and/or Javascript, and the back-end languages include SQL.

To use Potify, a user will have to log in or create a new account. To create a new account, the user will have to provide their information such as email and name. The account has its own username and password determined by the user. Once the user has an account, they can log in using their unique username and encrypted password.

A user and their account have the options to upload music into the Potify database, create collections of music, and listen to music. Users can upload music through specifying a folder/directory containing mp4 files to upload or individual mp4 files to upload. Each file that is uploaded must contain the title and artist of the song. The remaining information that cannot be stored in the mp4 file is stored in an excel sheet that the application will create. Listening to a song shows a ‘thumbnail’ of the current song playing.

**EER Diagram**

When reading the EER, the attributes are in camel case, but when reading the relational

model, the attributes have an underscore since the letters are all uppercase.

The application consists of 5 entities: User, Collection, Song, Artist, and

Album. Each of the entities has one or more attributes.

The User entity can be created from an email, password and unique username. The entity in addition can add their first and last name to complete their account, which are both part of a greater “name” entity. Upon creation, a creation date will be created for them in the format MM/DD/YYYY. This account will also track their last access date in the format MM/DD/YYYY as well. The user also has the ability to create collections (playlists that comprise songs). Finally, the user has an attribute “playCount” for each individual song that the user plays which does exactly what it sounds like - it records the amount of times that the user has played the song from the time that they created their account to the present time.

An artist has a name and a unique ID - artistNum. They can create many songs and a song needs an artist to be created. The Song entity can be identified by its genres (which is a multi-valued attribute), the title of the song, a release date (in the format of MM/DD/YYYY) which is associated with an artist, a unique song id, and the length of the song. An additional attribute, play count, is unique for individual users and it just tells the user how many times they listened to a particular song. A song can either be a single (not contained in an album) or in a single album.

An album can be created by one or more artists and an artist can have as many albums. Each album contains one or more songs and each song has its own track number. Each album has a release date, a unique id, genres, name, the number of songs it contains and the play time. An album cannot contain duplicate songs.

The Collection entity, created by a user, has a non-unique name as well as a unique ID - collectionNum. The user can change the collection name after it has been created and can delete the collection. It comprises of as many songs as the user wants to put in. The collection cannot contain an album otherwise there would be duplicate tracks, but the user can add all of the album’s songs into a collection manually. The album in addition contains the number of songs in the collection and the total duration of the collection. The collection cannot exist without its original user. Much like an album, the Collection cannot contain duplicate songs. The songs in the collection can be played in the order the songs were added into the collection or shuffled - songs are randomly played.

**Reduction to Tables**

Collection-Song, Song-Genre, Album-Artist, Album-Genre and Artist-Song now represent the relationship between the respective many to many entities or multivalued attributes. These special relations use the primary keys of each participant in the relationship as the total primary key.

Album-Artist links together the artist and their album via their primary keys. We chose to do it like this because an artist can have multiple albums and an album can have multiple artists.

Album-Genre links the genre and album together. The genre is a multivalued attribute on the album which is why we had to create a separate relation.

Artist-Song connects the song and artist together since a song can have many different artists and an artist can have many different songs.

Collection-Song connects the collection with the song. The collection can have multiple songs and the songs can be in multiple collections.

Song-Genre is a unique attribute for the Song entity. Since it is a multi-valued attribute, it has additional attributes - song\_num and genre\_list. Song\_num is the identifier that correctly assigns a list of genres to a particular song.

Song-Album connects the songs to the album they are featured in. This relation also has a special attribute called track\_num which stores the order of the album. This was done because sometimes an artist arranges an album in a specific way to add to the music.

User-Song connects the user to the song that they played or are playing. This relationship has an attribute called playCount that is unique for each individual song that the user has played and basically tells them how many times the song was played since the moment their account was created.

**Phase 3**

Regarding the EER Diagram, we added the attribute “played” to the relationship between collection and song. Since one of the requirements of phase 3 is that when a song is “played” by a user the song will be recorded as “played”, the best way we could implement this was having a boolean attribute to determine whether or not a song was played in a collection.

Regarding the Reduction-to-tables, the same exact changes to the EER Diagram was also carried out here.

Initially we had planned to code in Java, but instead we chose to change the input to cmd line while coding with python.

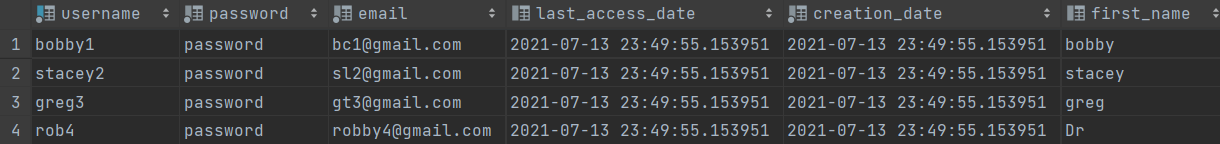
Sample SQL Statements:

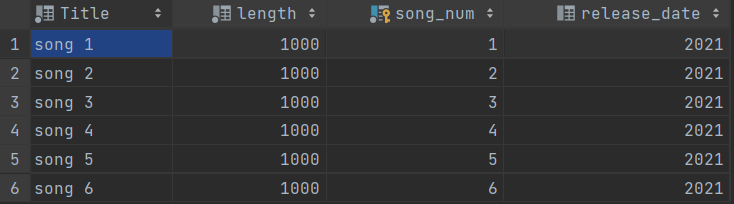
SELECT COUNT(\*) FROM “collection” WHERE name=%s AND username=%s, (collect, user)

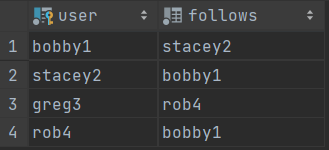
INSERT INTO “collection” VALUES (%s, %s, %s, %s, %s), (collect, new\_num, “  
0”, “0”, user)

DELETE FROM “friends” WHERE “user”=%s AND follows=%s, (current\_user, found\_user[0])

Sample Queries:







Data was loaded into the database via connections to reddwarf.cs.rit.edu, sending SQL statements in the form of strings through the connection so that we can see the changes in DataGrip.