Bailey Anderson

German, Linstead

Database Management

12 December 2018

BetterBeatMatch™

With the ever growing popularity of electronic music, DJing is quickly becoming one of the most widespread activities. Whether it be at a simple house party or a huge music festival, DJs today rely on mixing software to spin their tracks together. These programs store tracks in a library, where the tracks are organized alphabetically by artist name. While this might not seem like it would matter, it makes finding the right track a very difficult task. When looking for the perfect song to transition to, a DJ must take several factors into consideration. First, the DJ must pick a song in the same style, or subgenre, to keep the crowd going. Next, they need to think about the key and beats per minute, or BPM, of the song. Key and BPM are important because if the transition song has too different a key or a BPM than the song playing, then the transition will sound like garbage and the dancing crowd will lose all momentum and energy. On top of this, DJs might have to sort through thousands of songs, and only have mere minutes to transition to the next song before the current song ends. With these glaring problems in mind, DJs need a better way to sort through their libraries so that they can keep the party going. Enter BetterBeatMatch™.

BetterBeatMatch™ is the name of my music database. It is written in Java and implements MySQL to store two tables of track information read off of two CSV files. It allows a user to connect to their own database in mySQL which they create. Once the program is ran, it asks the user to enter the database name, username, and password for MySQL. After this is done, the user should select option 8 to create the tables, and option 9 to populate them. createSongs() is used to make the Songs table, and createArtists() is used to make the Artists table. Then, readSongData() and readArtistData() are used to read the CSVs into the tables and store them in MySQL. The first table, Artists, contains information about the artists, where ArtistID is the primary key.:

ArtistID	**Artist**	**Genre**
1	Excision	Bass
2	Habstrakt	House
3	Dela Moontribe	Drum and Bass
4	Space Jesus	Bass

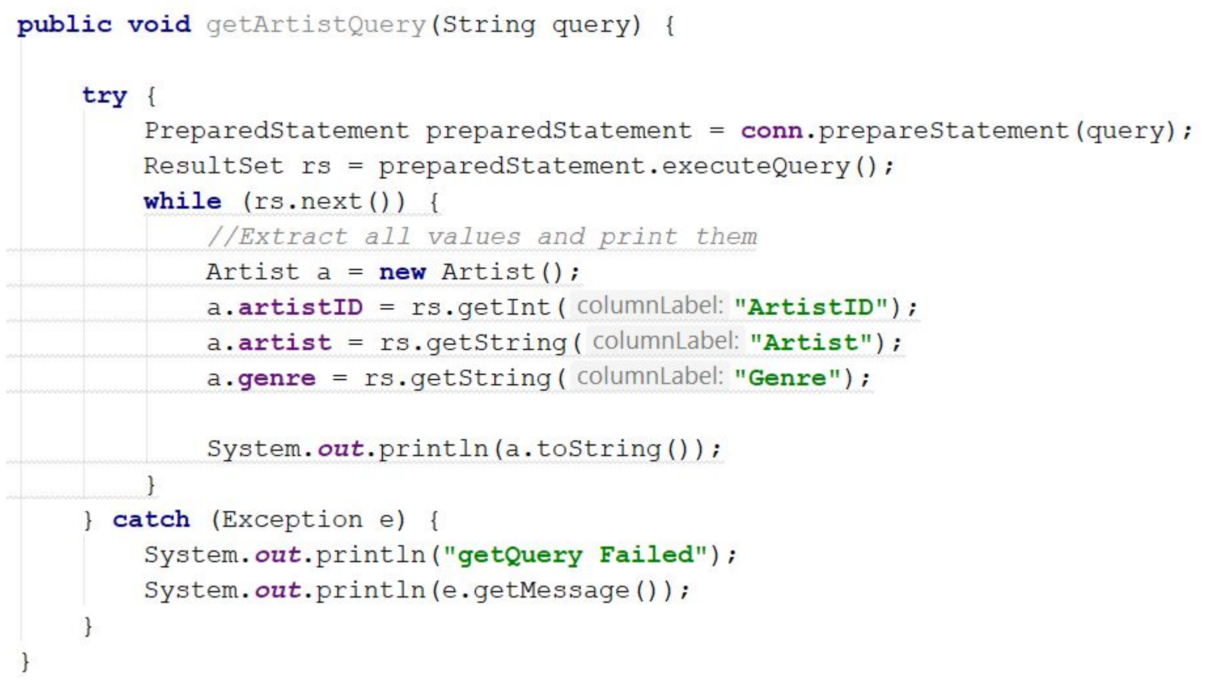
The second table, Songs, contains information about the songs of each artist where SongID is the primary key and is connected to Artists through the ArtistID field:

SongID	**Subgenre**	**Key**	**BPM**	**Duration**	**Comments**	**ArtistID**
1	Dubstep	A	70	200		1
2	Dubstep	G	88	300		1
3	Bass House	C	70	350		2
4	Neuro Funk	B	128	256		3

After reading the track data from the CSVs, the program is ready to be used. It implements CRUD in the following ways. Create - the methods createSongs() and createArtists()

create the two datatables used in the program. Read - the program features several search and print functions that use SELECT statements to display information to the user. The method get getFullQuery() uses a SELECT statement to print all song information. First, it uses a query to bind the Songs and Artists tables together. Then, it creates two objects, Song and Artist, which contain member variables of each attribute of their respective tables. It then uses a loop to assign the values in the tables to the member variables of the objects using ResultSet(). Finally it calls each of the object's toString() methods to print out the completed table to the user.

getArtistQuery, half of getFullQuery:



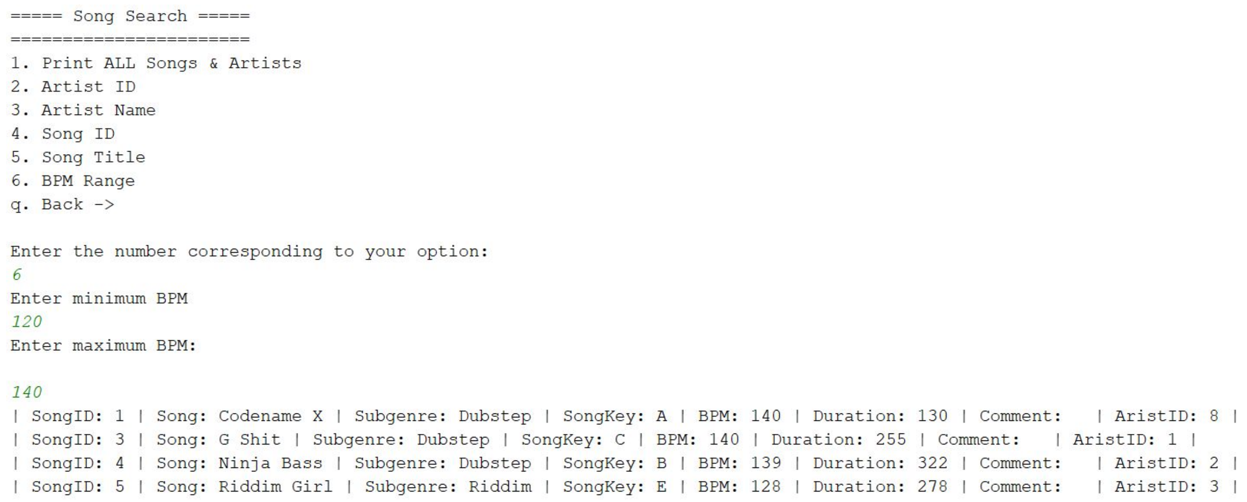
```
public void getArtistQuery(String query) {

    try {
        PreparedStatement preparedStatement = conn.prepareStatement(query);
        ResultSet rs = preparedStatement.executeQuery();
        while (rs.next()) {
            //Extract all values and print them
            Artist a = new Artist();
            a.artistID = rs.getInt( columnLabel: "ArtistID");
            a.artist = rs.getString( columnLabel: "Artist");
            a.genre = rs.getString( columnLabel: "Genre");

            System.out.println(a.toString());
        }
    } catch (Exception e) {
        System.out.println("getQuery Failed");
        System.out.println(e.getMessage());
    }
}
```

Update - the method addSong() allows users to insert a song into the tables. As of the current release, the user is unable to add an artist to the database, and adding songs with a new artist will not add that artist to the artist table. Delete - the deleteSong() method allows users to delete songs from the Songs table upon entering SongID. One of the features I would have liked to implement is the ability to delete songs by entering other attributes, such as BPM or subgenre, or delete whole artists at a time. BetterBeatMatch™ can export data to a CSV by using the writeData() method, which lets the user save both the songs table in a file called song.csv and artists in artist.csv.

An example of the user menu and program output after selecting to search for a song based on BPM range:



```
===== Song Search =====
=======================
1. Print ALL Songs & Artists
2. Artist ID
3. Artist Name
4. Song ID
5. Song Title
6. BPM Range
q. Back ->

Enter the number corresponding to your option:
6
Enter minimum BPM
120
Enter maximum BPM:

140
| SongID: 1 | Song: Codename X | Subgenre: Dubstep | SongKey: A | BPM: 140 | Duration: 130 | Comment:   | AristID: 8 |
| SongID: 3 | Song: G Shit | Subgenre: Dubstep | SongKey: C | BPM: 140 | Duration: 255 | Comment:   | AristID: 1 |
| SongID: 4 | Song: Ninja Bass | Subgenre: Dubstep | SongKey: B | BPM: 139 | Duration: 322 | Comment:   | AristID: 2 |
| SongID: 5 | Song: Riddim Girl | Subgenre: Riddim | SongKey: E | BPM: 128 | Duration: 278 | Comment:   | AristID: 3 |
```

One of the cool features of BetterBeatMatch™ is the QuickSort™ function. It allows the DJ fast access to the optimal choices for their next track. First, the user enters the SongID of the track they are currently playing. Then the QuickSort() function is called, prints songs with a BPM range of BPM - 10 <= BPM <= BPM + 10, and in the same subgenre and key as the one they entered. This will show the DJ the easiest songs to transition to with only a single command, perfect for novices.

The next step for this project would be to implement it as a feature of commercial mixing programs, such as Mixxx. Mixxx is the program that I use to mix, and while it has many cool features such as BPM matching, it lacks a good way to sort music besides the user organizing them into different folders on their own, which lacks the ability for on the fly custom sorting. To implement this, the program would need to read the mp3 files from Mixxx and put them into tables for the program to use. I have talked to a few of my friends who DJ and they said that this feature would be extremely useful for them. All in all, this project taught me a lot about SQL in java and gave me some interesting ideas about how using databases is useful in creating products in the real world.