**Resonance:** *filling the silence with harmony!*

*Group 4: Jaron Bullock, Drew Beaudry, and Komi Yenkey*

Motivation:

Our team has discovered that we share a common interest in music. While we all may prefer different genres and artists, we are aware of the great effect music can have on human emotion. If you’ve ever felt chills while listening to a certain song, you might be surprised to know that this is caused by the brain releasing dopamine, a feel-good chemical, in anticipation for the peak moment of a song. In fact, music is one of the very few activities that utilizes the entire brain. Research teams have found while utilizing FMRI technology that music can employ large-scale neural networks within the brain, activating emotional, motor, and creative sectors. This motivates us to create an application that may help a user discover music they’ve never heard before, which may be related to music they’ve heard and loved in the past.

The application idea relates directly to one of our team member’s current occupation as a wedding DJ. This position entails constantly analyzing a crowd’s body language to determine their feelings about a current song, and deduce what song might continue a positive experience or instead promote a better experience than the current song. This application may help this deduction process by allowing the DJ to input a current song or artist, and retrieve songs or artists which are similar. This functionality may also prove useful when a guest might request a song which is not in the DJ’s library of audio files. In this situation we may use the application to find songs which are similar to the one that the guest is requesting, and this may greatly enhance the guest’s experience. In addition, this application may help a DJ store metadata, or tags, about a certain song. If a DJ were to pick a song and the reaction was very positive, the DJ would be able to tag this information to the specific song to be able to analyze this data or retrieve it later.

Application Description:

Resonance is a web-based application which utilizes database technology and Last.FM’s API in order to make music recommendations to its users. There are a variety of methods by which users can request music recommendations. Users can input a tag and receive a list of artists which are associated with that tag. Tags are user generated descriptions of artists, which may vary in content from the genre to the year of production to the general mood of the music. Users are also allowed to input artists, and thereafter receive all the tags associated with those artists. This allows users to explore and discover different categories that they might enjoy. It is also possible to input an artist, and receive recommendations based upon that specific artist. This feature utilizes Last.FM’s API, which relies upon an algorithm that analyzes a vast amount of user generated data. Users are also allowed to tag artists with their own customized tags, which are then added to the database. In addition to artists, it is possible for users to interact with specific tracks. Users are allowed to like specific tracks. Future iterations of this application would be able to analyze similarities in user data and make more personalized recommendations from within the application.

Team Task Assignments:

As per the agreed upon role assignment, each team member was given a handful of tasks for which they were responsible. The following describes these tasks, in addition to any tasks that were deemed necessary to undertake as the project progressed.

Jaron was responsible for the tasks related to data and database management, as well as overseeing project planning. As far as the data is concerned, Jaron was responsible for cleaning the data set, as well as inserting it into the MySQL database. He was also responsible for designing the structure of the relations and tables, and then implementing those designs into the database. In addition, he also created all the MySQL queries and stored procedures which were necessary to interact with information stored within the database. Lastly, Jaron was responsible for making sure that checkpoints were addressed in an effective and timely manner. As the project progressed, it became apparent that the interface was more of a challenge than expected, and so Jaron also undertook the task of assisting Komi with implementing the PHP interface.

Drew was responsible for tasks relating to database and server administration, as well as interfacing with Last.FM’s API. This meant that Drew was responsible for creating an Amazon Web Services MySQL Server, setting up access for each team member, and defining security rules to ensure controlled access. He was also responsible for getting an API key, and both creating requests to and parsing responses from the Last.FM API. As a part of this task, he created and hosted an HTTP Endpoint which allowed the PHP interface to make simplistic requests to the API and receive a clean and easier to parse response. Drew also retrieved all artists from the database, requested play count information from the API for each artist, and issued updates to the database’s artist table. Additionally, Drew took on an assisting role in helping the interface connect with the endpoint and assuring that it properly parsed the data.

Komi was responsible for all tasks regarding the application interface. This meant that Komi was responsible for designing and implementing that interface utilizing a combination of HTML and PHP. Another part of this task was enabling the interface to interact with the MySQL database and also an HTTP Endpoint. In addition, Komi was responsible for parsing the data from these sources and displaying it in a user-friendly format.

Description of System Architecture:

The application utilizes data which is hosted on an Amazon Web Services MySQL database. The database utilizes four main tables, and three relational tables in order to work. The main tables consist of artists, tracks, tags and users, while the relational tables consist of uses, likes and compose. Uses connects the artists, tags and users tables. Likes connects the users and tracks tables. Compose connects the artists and tracks tables. Utilizing these tables, we are able to extract and relate any information contained within the database.

For additional data which supplements our dataset, parts of our application will communicate with the Last.FM API. Specifically, we request artist information, which also returns artists which are similar to the artist provided – we request track information, and we request that Last.FM inform us which tracks are similar to a provided track. To do this, we use the OkHTTP3 Java library to create requests, issue them to an http server and receive a response from the server, including headers and a JSON message. We then use carefully created Java objects which match the exact structure of the response, and utilize reflection with the Google code GSON library to map the JSON response into specific objects. In the creation of these object classes, we can specify which data we would like to save from each JSON response, effectively cleaning the incoming data and discarding information which we will not display on our user interface. We allow our PHP interface to communicate by hosting a HTTP endpoint with the Vertx Java library. We establish an endpoint listening on port 8081, and define routing which allows the PHP interface to very simply provide URL parameters, such as a song and/or artist, to the java application, and received a cleaned and easier to parse JSON response.

We have chosen the Model-View-Controller (MVC) architecture pattern to develop our web application. This allows us to separate our code and to manage the way the information is presented to the user and how the information also is accepted from the user. The major components allowing more efficient code reuse.

Model: Our mode is written in PHP. This model file holds all of our functions that we have created to perform many tasks such as check a valid user, get one record and to get all the records.

View: In our view folder, we use PHP, and bootstrap to make a dynamic web application that will make it easy to display the result set from a query in a more presentable way back to the user.

Controller: The index file is written in PHP and is the main controller for our web application. Every interaction made through the GUI by the user will be passed through the index file that will then redirect the user request to the right page to return the appropriate result back to the user.

The client side is written in PHP with a combination of HTML that allows users to interact with our database. First, a user needs to authenticate himself through a login process. The validation will be done through the database against the users' table before any access will be given. After a successful login, the user will have access to the menu page. The menu interface is composed of four inputs box.

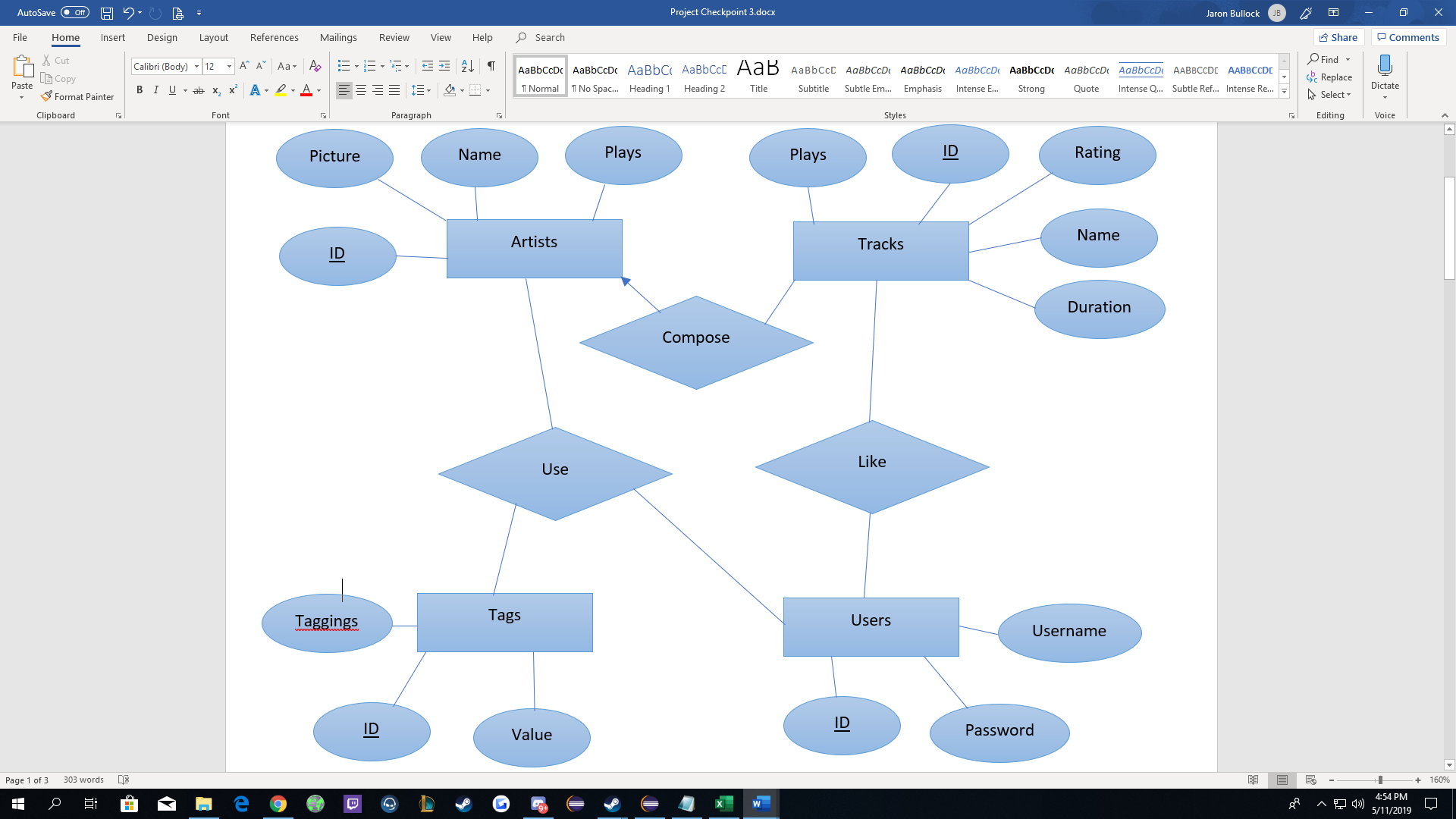
The first input box allows the user to search for artist tags by entering an artist name. The second input allows the user to retrieve the artist name by entering their tags. The third input box allows the user to query the API for a similar artist by entering an artist name. The last input also allows the user to tag an artist by entering the artist name and the tag name. The interface also gives options to users to perform two additional searches. The user will have the ability to provide an artist name and track name, and discover songs provided by the API which are similar to the one provided. Additionally, the user can query directly the database for popular artists. This returns a short list of artists sorted by play count according to the API.

To make a connection from our web interface to our AWS MySQL database, we have used PDO connect object within PHP script. To avoid sharing the database login information we have written all the codes in PHP and ran that on the server side. We have used a PHP script to be able to connect to a HTTP endpoint which allows us to query the API indirectly and receive cleaner responses. The result we get from the API is in a JSON file and then a PHP script is used to decode the file and then return the result back to the user in an appropriate readable format.

Description of the Dataset:

The dataset utilized was a Last.FM dataset from 2011. It contained six different text files of which only three were used. The three utilized text files were named “Artists.dat”, “Tags.dat”, “User\_taggedartists.dat”. “Artists.dat” contained 17,632 artists, including their id, name, Last.FM URL, and picture URL. “Tags.dat” contained 11,946 tags, including their id and value. “User\_taggedartists.dat” contained 186,479 tag assignments, including the user’s id, artist’s id, tag’s id, and the day, month, and year of the tagging. These files were used as the main dataset for most of the tables. However, this data alone proved to not be enough, and so one more additional data file was added. This file was also a Last.FM dataset and contained 1,000,000 different songs, including their id, title, release album, artist, and year. While this supplemented our track data, there were no files connecting the tracks to the artists, and so the relational data was generated using java to parse and match up the data as well as run MySQL lines and procedures.

ER Diagram:



***Disclaimer: This ER diagram has been changed since submission in Checkpoint 2. This one accurately reflects the format of the database but does not necessarily reflect the names used. Names in the actual database were updated slightly for more clarity.***

Relational Model:

**Artists** (ID: Big Int, Picture: String, Name: String, Plays: Big Int)

Primary Key: ID

*FD: ID -> Picture, ID -> Name, ID -> Plays*

**Tags** (ID: Big Int, Taggings: Big Int, Value: String)

Primary Key: ID

Other keys: Value

*FD: ID -> Taggings, ID -> Value, Value -> ID*

**Tracks** (ID: Big Int, Plays: Big Int, Rating: Integer, Name: String, Duration: String)

Primary Key: ID

*FD: ID -> Plays, ID -> Rating, ID -> Name, ID -> Duration*

**Users** (ID: Big Int, Username: String, Password: String)

Primary Key: ID

Other Keys: Username

*FD: ID -> Username, Username -> ID, ID -> Password, Username -> Password*

**Use** (Users ID: Big Int, Tags ID: Big Int, Artists ID: Big Int)

Primary Key: Users ID and Tags ID

**Like** (Users ID: Big Int, Tracks ID: Big Int)

Primary Key: Users ID and Tracks ID

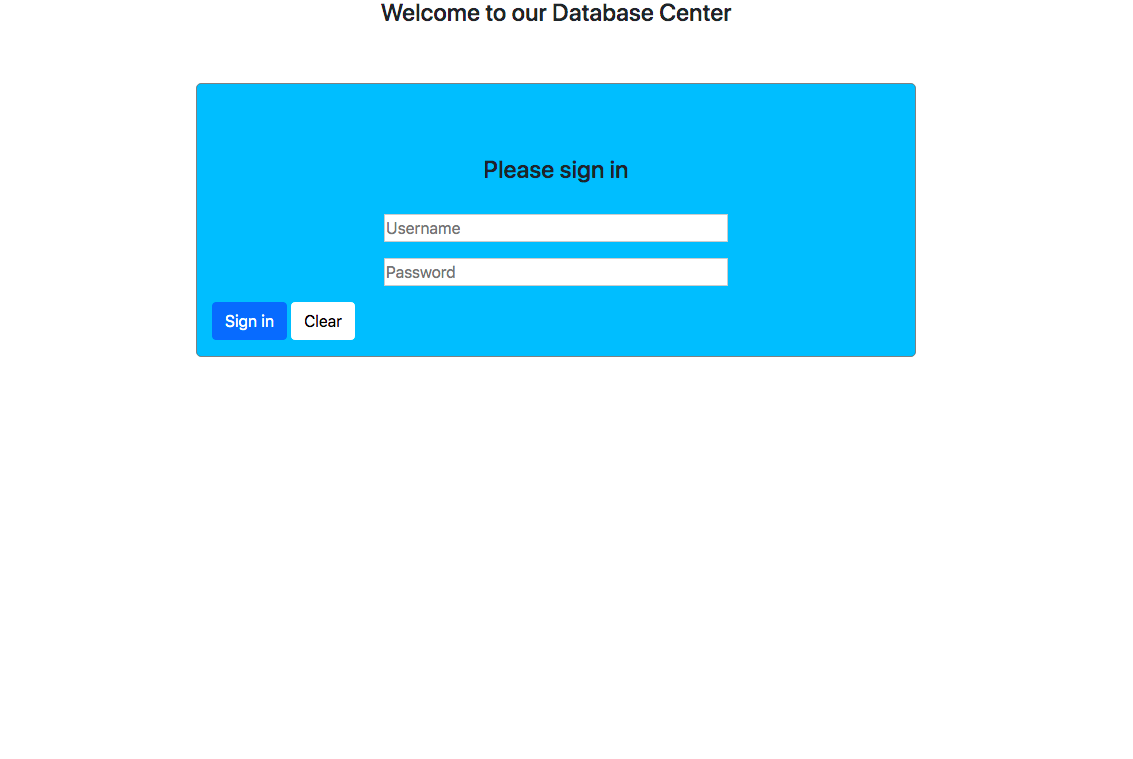
Compose (Artists ID: Big Int, Tracks ID: Big Int)

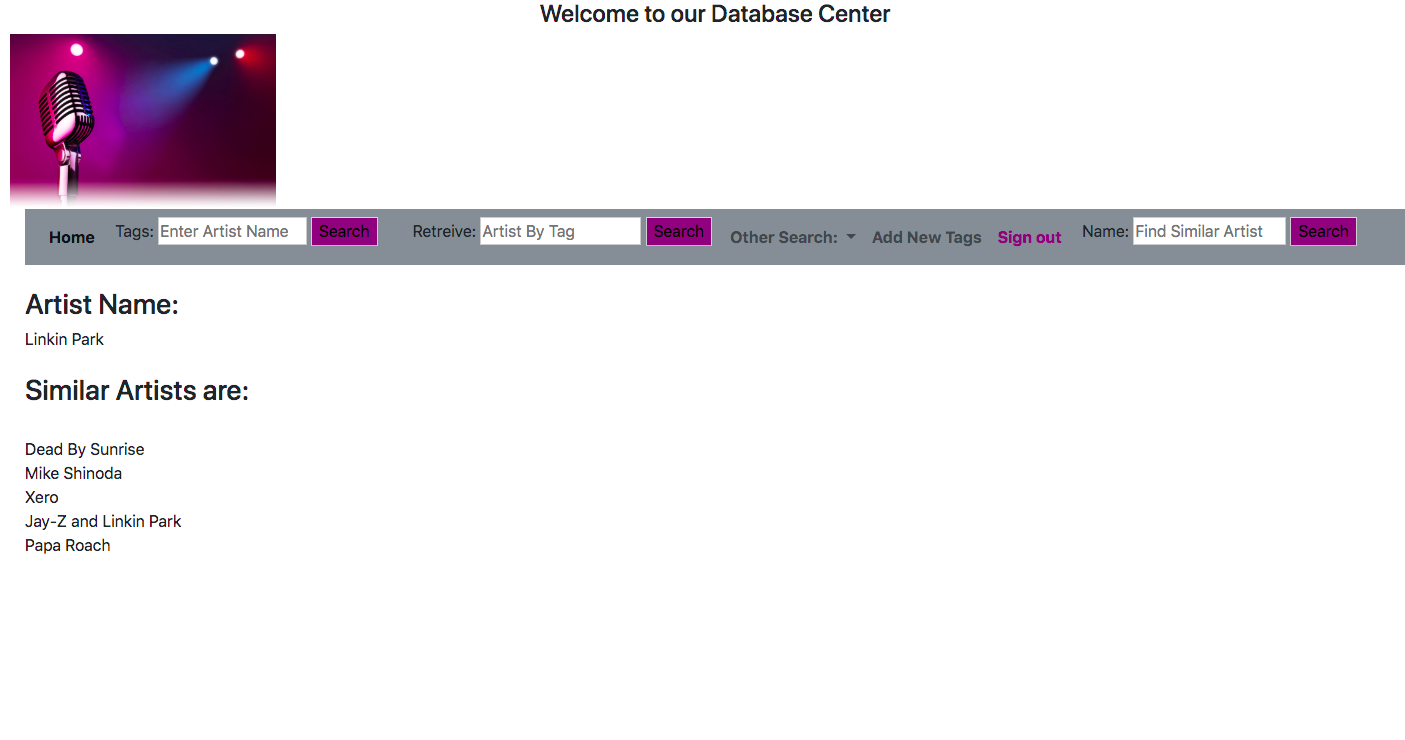
Primary Key: Artists ID and Tracks ID

***Disclaimer: Relational model was written to match the ER diagram, and once again does not necessarily have matching names to the actual database. The format and style of values is still the same, it is only the names that do not necessarily match.***

Implementation – Description of the Prototype:

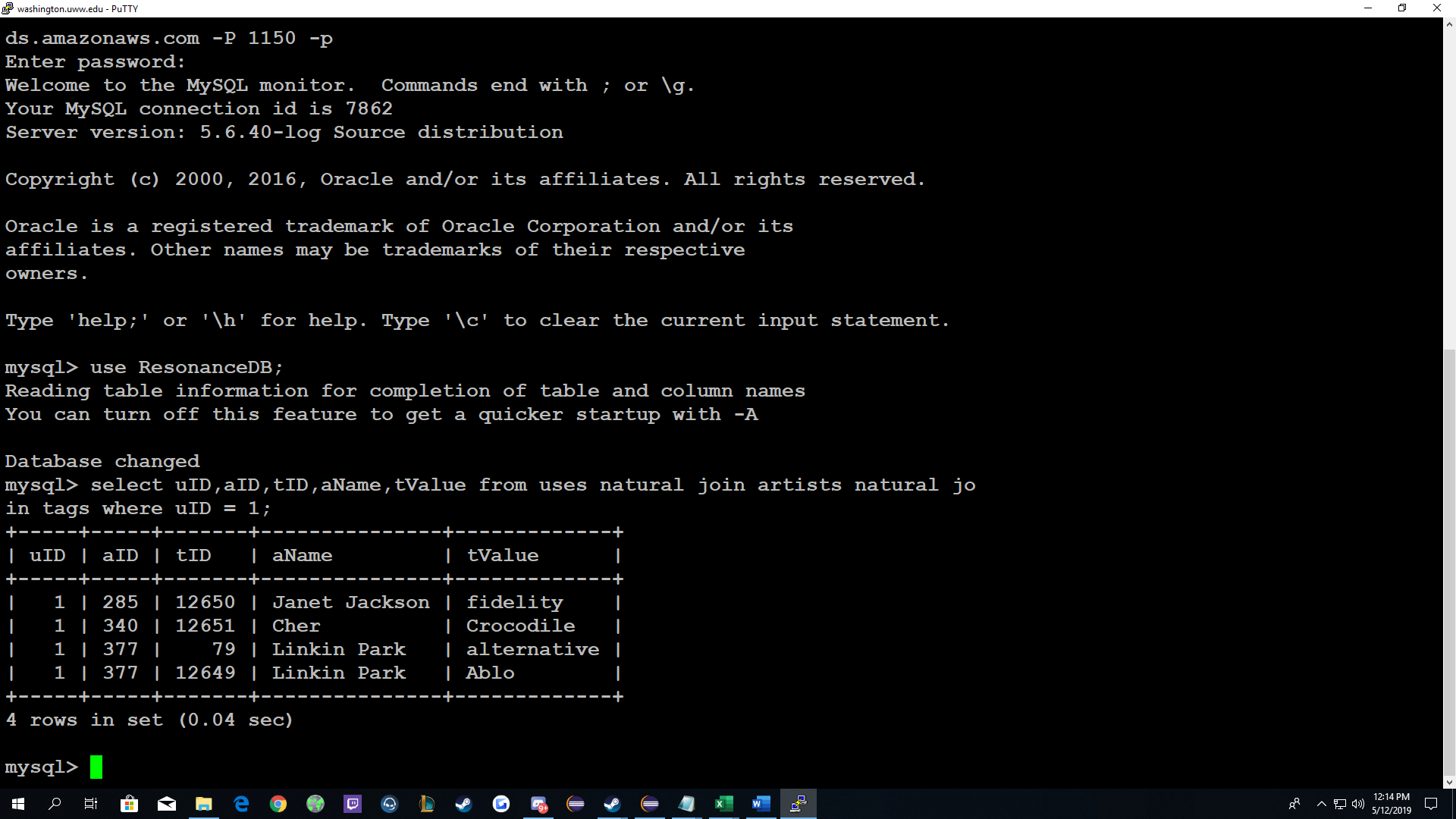
Our prototype uses a PHP/HTML interface (pictured below) in order to interact with the AWS MySQL database and HTTP Endpoint. Users are prompted to login, and then can search the data on the following page. The interface pulls information from two locations, the AWS database using SQL queries designed for different purposes, and the HTTP Endpoint which hosts data pulled from the API. Data is pulled from the API and uploaded to the Endpoint using a java-based program.



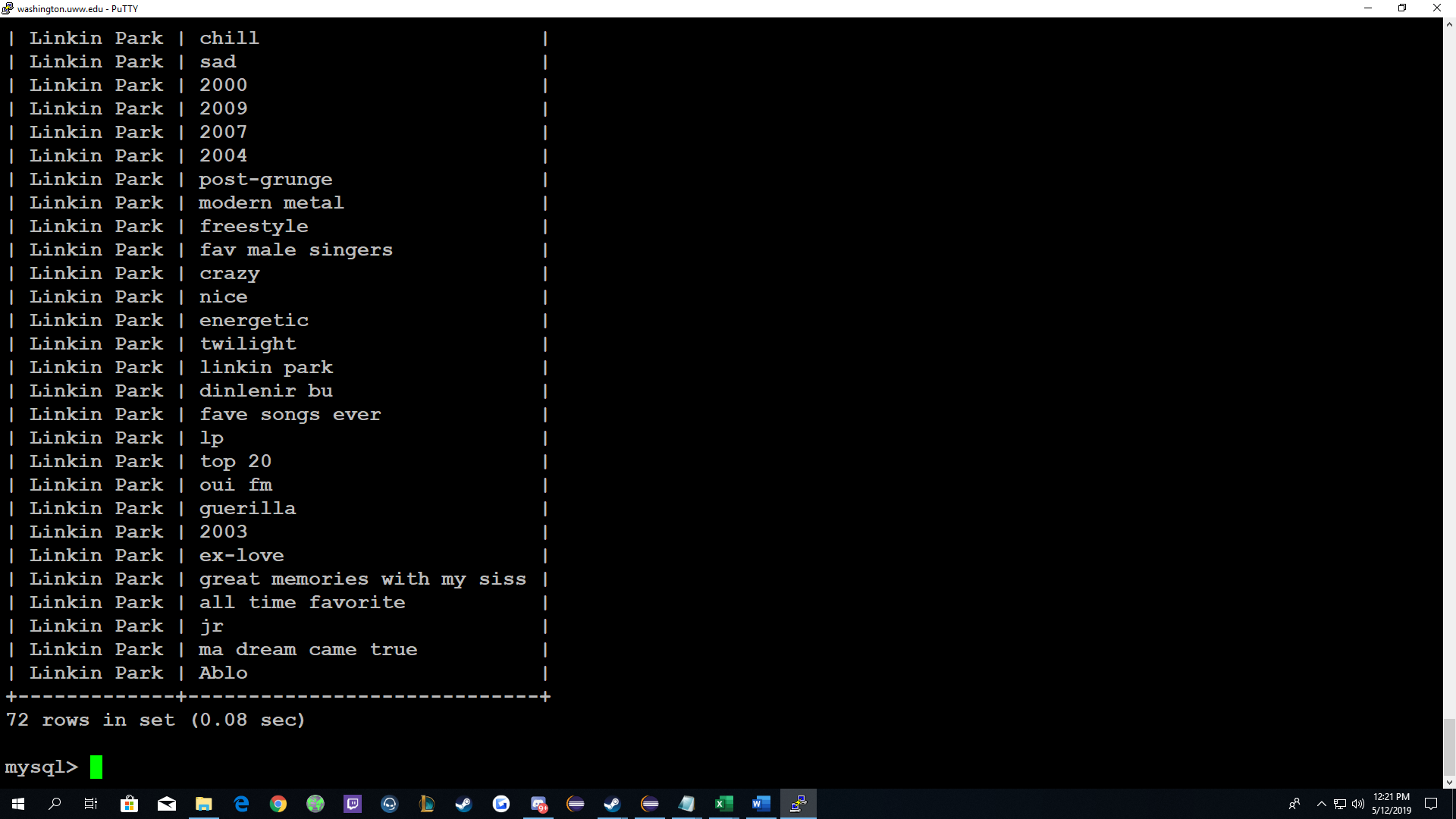
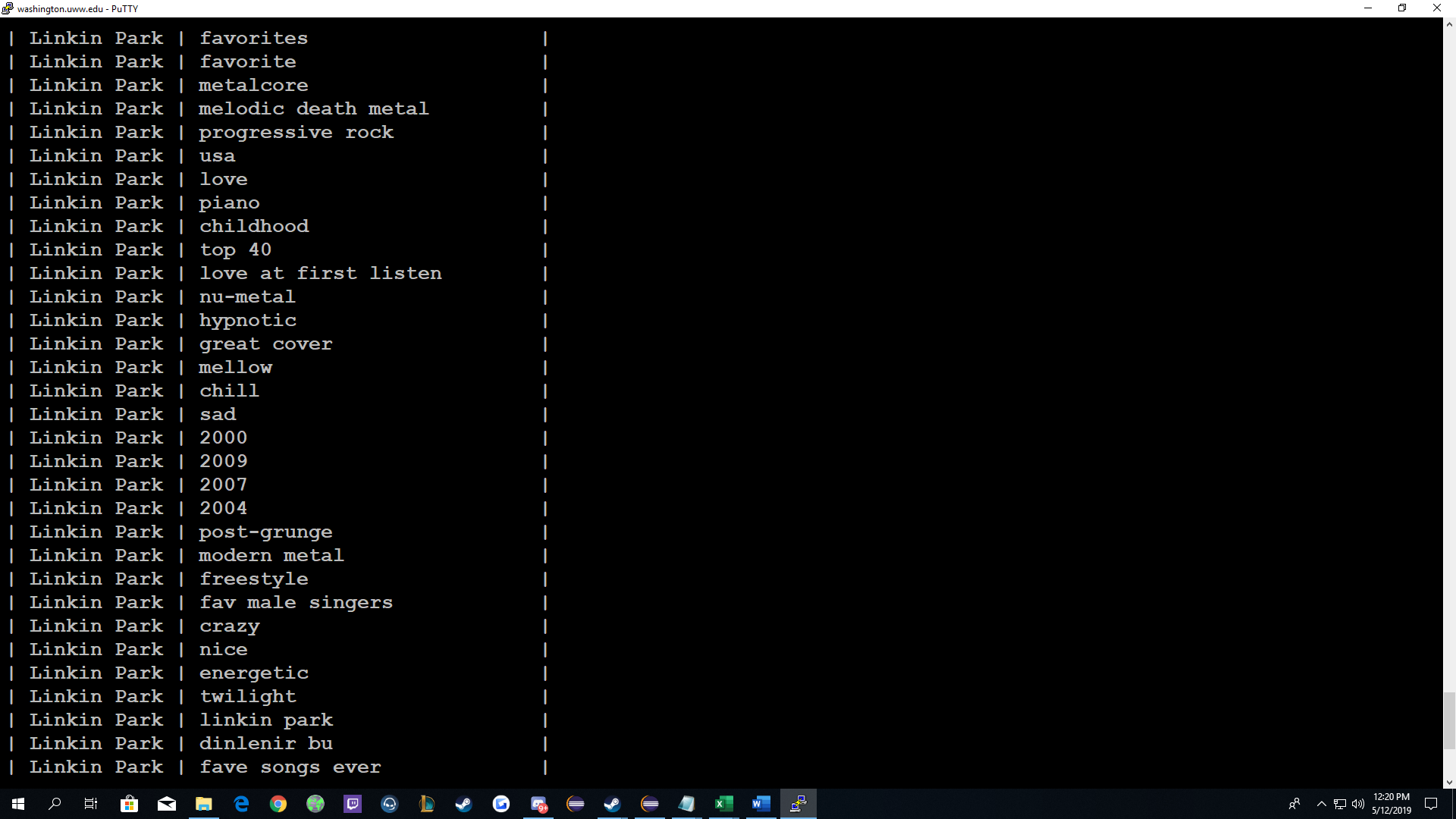
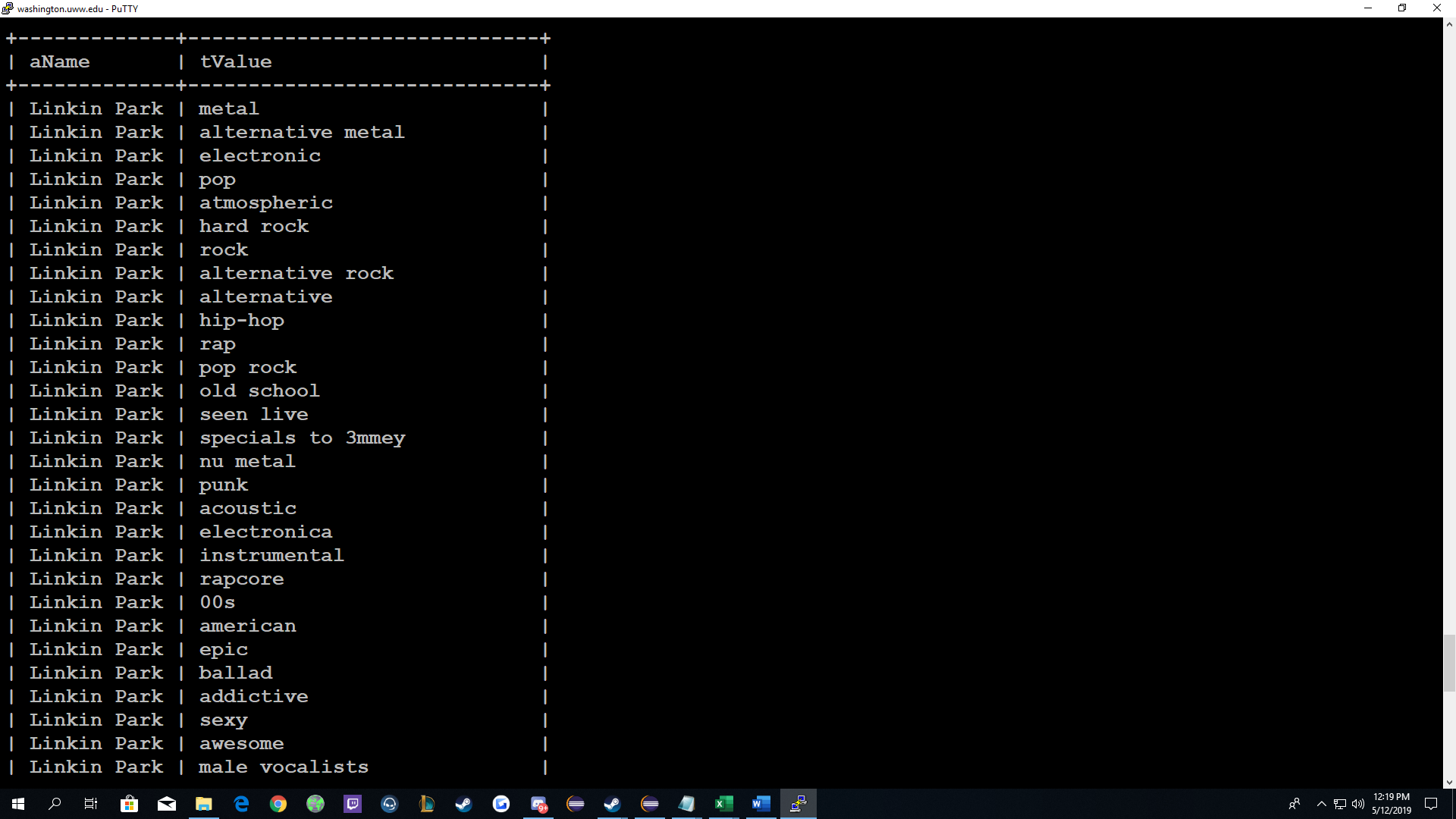


Evaluation – Description of Testing:

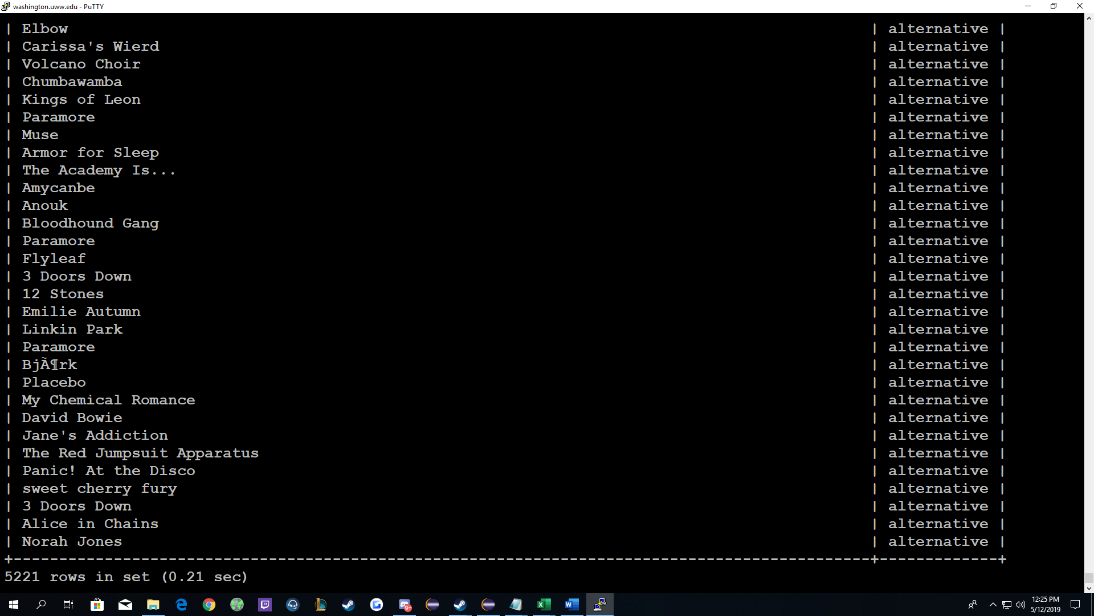
Testing our application consisted of running a variety of queries for our different methods of searching. We selected a few possible queries of each type to assure that each function was working correctly. We ran several queries to tag artists under user 1, which successfully tagged “Janet Jackson” with “fidelity”, “Cher” with “Crocodile”, “Linkin Park” with “alternative” and “Linkin Park” with “Ablo” and uploaded these values to the database.



We were also able run several queries to return the tags associated with different artists.



As well as all of the artists associated with a tag.



Conclusion:

Overall this project had great amount of excellent learning opportunities. It was a largely open ended project which allowed us to explore practical uses of the languages and techniques we’ve learned in our years of schooling. In Lab two, we were able to learn enough about DB2 to decide against using it for the sake of simplifying the connection process. There was also an opportunity to deal with large amounts of data from the 1,000,000 song dataset, and experience cleaning and parsing that data. From creating the backend, we learned about using some popular libraries like GSON to encode JSON data and decode it with reflection, and Vertx to provide HTTP endpoint routing. We also learned about using a not so popular library – OkHTTP3 to communicate with the Last.FM API via HTTP GET requests. We learned a lot about setting up and connecting to AWS Relational Database Servers, and using popular tools like MySQL Shell and MySQL Workbench, as well as some not so popular MySQL utilities like the “mysqldbexport” utility to export both data and metadata remotely from our AWS RDS server. We collectively learned more than we would have liked to about PHP, and our team member who implemented the user interface learned a great deal about practical uses of PHP from the frustrations of roadblocks while learning the difficult new topics of parsing JSON received from the HTTP endpoint, as well as connecting to and correctly communicating with an AWS MySQL server using PHP. We collectively feel that this was an excellent experience, and once which was important to have.

References – *(an incomplete listing of some of the most important documentation)*:

Connecting to AWS RDS via PHP

<https://stackoverflow.com/questions/29039296/connect-to-mysql-on-amazon-aws-rds>

Basic SQL Cheat Sheet

<https://www.w3schools.com/sql/>

Our Data Set

<http://files.grouplens.org/datasets/>  
Setting up AWS

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateDBInstance.html>

<https://aws.amazon.com/getting-started/tutorials/create-mysql-db/>

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateInstance.html>

Vertx Java Library Documentation

<https://vertx.io/docs/vertx-core/java/>

<https://vertx.io/docs/vertx-web/java/#_routing_by_exact_path>

Managing MySQL User Permissions

<https://www.a2hosting.com/kb/developer-corner/mysql/managing-mysql-databases-and-users-from-the-command-line>

MySQL Shell Documentation

<https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-commands.html>

Last FM API Instructions

<https://www.last.fm/api>

<https://www.last.fm/api/show/artist.getInfo>

<https://www.last.fm/api/show/track.getInfo>

<https://www.last.fm/api/show/track.getSimilar>

Information about IP Ranges – Setting up AWS RDS Security Rules/Firewall

<https://arstechnica.com/civis/viewtopic.php?t=751834>