Project 1A Specification

Computer Science 143 - Spring 2019

TuneSearch

Project 1 Part A

Due Friday, 04/19/2019 by 11:59pm

Partners

The CS143 project may be completed in teams of one or two students. The choice is up to each student. Partnership rules consist of the following:

- An identical amount of work is expected and the same grading scale is used regardless of the size of your team.
- If you work in a team of two, choose your partner carefully. Partners are permitted to "divorce" at any time during the course, and "single" student may choose to find a partner as the project progresses. However students from divorced teams may not form new teams or join other teams.
- Partners in a team will receive exactly the same grade for each project part submitted jointly.

If you have two students in your team, please make sure to submit your work jointly - do *not* turn your project in twice, once for each partner.

Scope

The primary purpose of Part A is the following:

1. To provide you with the data you need to complete the project.
2. To create the PostgreSQL relation/table schemas you will need to complete the project.
3. To load the data.
4. To make sure the VirtualBox image is working correctly to ensure your further success.

We provide you with a whole bunch of basic information to get everyone up-to-speed on our computing systems and the languages and tools we will be using. Those of you who have used a database server before (remember, we do not expect any familiarity), will find this part nearly trivial. Those of you who haven't will find it mostly straightforward. With all that said, *please don't start at the last minute* -- as with all programming and systems work, everything takes a bit of time, and unforeseen snafus do crop up.

System Setup

We will be using VirtualBox to run the Linux operating system in a virtual machine. VirtualBox allows a single machine to share resources and run multiple operating systems simultaneously. Read our VirtualBox setup instruction and follow the instructions to install VirtualBox and our virtual-machine image on your own machine.

The provided virtual machine image is based on Ubuntu 18.04.2, PostgreSQL 10.1, Nginx, Python 3 and Flask. You will need to use the provided VirtualBox guest OS to develop and test all projects for this class.

Your VirtualBox guest is essentially a Linux machine. Your guest is accessible from your host through secure shell (SSH) at `localhost` port **1422** with username `cs143` and password `cs143`. The web server and framework (nginx + uwsgi + Flask) is accessible from your host at `localhost` port **1480**.

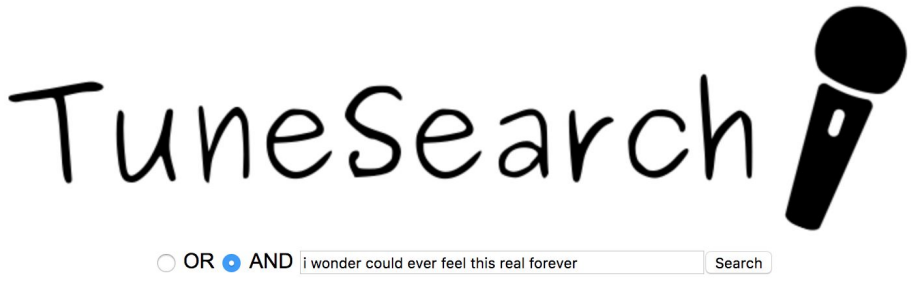
If you have any problems with the image, let the TAs know ASAP on Piazza.

The Big Picture: What we are Going to Do

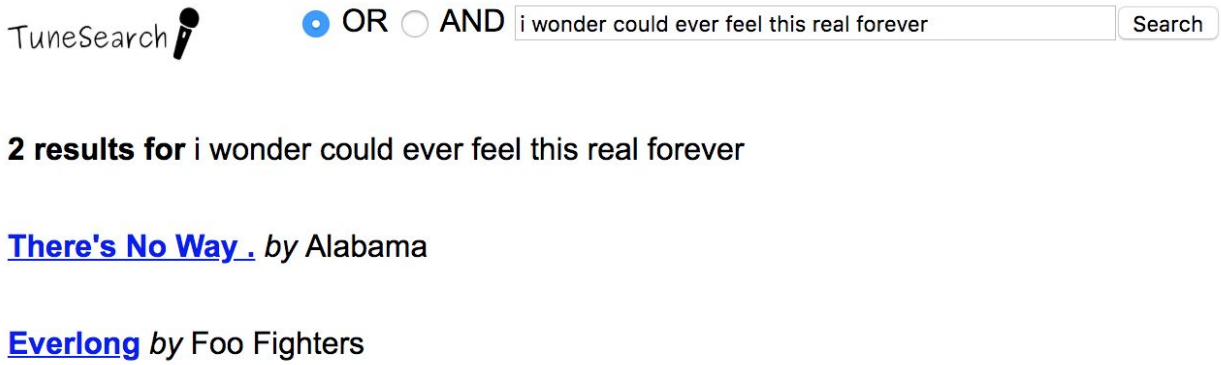
In this project, we are creating a very basic search engine a la the 1990s, called TuneSearch. We will assume that a TuneSearch user enters his/her query as a space-delimited string of tokens/words. The user can then use one of two radio buttons labeled AND or OR to determine how we will search for the terms. If OR is selected, any song that contains any of the words will be returned as search results. If AND is selected, only songs that contain ALL of the words will be returned. Nowadays, search engines like Google assume AND queries, but believe it or not, several search engines used to require the user to specify how to create the boolean predicate.

An example of searching from some lyrics:

An example of an AND query:



The results:



But it is not just enough to search for words in a song and present them to the user. We also need to *rank* them in a way that makes sense. This is very complicated for modern search engines, so we are going to keep it simple. One way is to just rank by how many times the words in the query appear in each song result. A slightly more sophisticated method that we will use involves using TF-IDF, which is a metric that explains how *important* a particular word is in a song. Words with high TF-IDF values are those that will lead to more specific results. TF stands for term frequency, and is simply the number of times a word appears in a song. IDF stands for Inverse Document Frequency. Its reciprocal is the number of songs where the particular term is used. TF-IDF will prioritize songs that use a rare word frequently. While a set of songs may contain the word *raspberry*, it is rare enough that the song *Raspberry Beret* by Prince should be ranked first, because the word *raspberry* appears many times compared to others and not very often in other songs. A word such as *and* has a very low TF-IDF because it appears in practically all songs. We will discuss TF-IDF more in the specification for Project 1B. **For project 1A, it is important to know that there is a TF-IDF score for each song-token pair.**

The Data

The dataset consists of lyrics from several thousand songs collected from the popular website LyricsFreak in CSV format. This data was heavily cleaned by your professor. In your VirtualBox image, there is a directory named `data`. This directory contains three files:

1. `artist.csv` contains data about each artist in the dataset. There is an ID number, and their name in quotation marks. The quotation marks are necessary because some names may have commas in them.
2. `song.csv` contains data about each song in the dataset. There is an ID for both the song and the artist, the name of the song in quotation marks, and the suffix of the LyricsFreak URL where the data came from. For example, the song *Ants Marching* by Dave Matthews Band has the URL `/d/dave+matthews+band/ants+marching_20036621.html`, which, as a full URL is

http://www.lyricsfreak.com/d/dave+matthews+band/ants+marching_20036621.html

3. `token.csv` contains data about the individual words (referred to as *tokens*) used in each song. There is an ID for the song, a token, and the number of times that word was used in the song. This table basically contains TF, or at least part of it.

The Database

Your VirtualBox instance contains a Postgres database server on it. You can access the command-line interface with the command `psql`. You will be authorized as user `cs143`, though note that this username did not need to be the same as the Linux username. We have set the password to `cs143`.

If you are familiar with MySQL, you will immediately notice some differences. Let's start with looking at what databases are stored in this server by using the `\l` command (lowercase L). Every database user (not necessarily Linux user) has their own database that they can do whatever they want with thus there are databases for `cs143` and postgres. `postgres` is a system user that is created when the server is installed. We will only be using user `cs143`. There is a `TEST` database you are free to use for whatever you like. You can use `cs143` or `TEST` for development purposes if you wish. There are databases for homework, and `searchengine`. You will use database `searchengine` for project 1. `template0` and `template1` are system databases -- don't modify or delete them. `template1` is a backup of `template0`.

Below are a few more commands that will be of use:

Command	**Postgres / psql**	**MySQL Equivalent**
List databases	`\l`	`SHOW DATABASES;`
List relations	`\d`	`SHOW TABLES;`
Show schema for relation *r*	`\d <r>`	`DESCRIBE <r>;`
Switch databases	`\c <dbname>`	`USE <dbname>;`

All SQL queries must be suffixed with a semicolon, but commands do not need to be.

Task 0: Load the Skeleton

Most of the Python code and web pages are already written for you. You can download these files at this link. Navigate to your `vm-shared` directory on your system and unzip the file so that your directory structure looks like the following:

```
vm-shared
|
|-- logs
|--SearchEngine
```

Depending on your operating system, or your zip client, it may create a directory named after the ZIP file. If this happens, move everything within that directory so that it is directly under `vm-shared`. This content is accessible on your VM at directory `www` in the home directory.

Task 1: Create the Schema

Your first task is to create the database schema for this problem. You will design four tables with appropriate names, specify types for each of the attributes, and specify the proper primary and foreign keys. The fourth table will contain the TF-IDF score, even though we are not computing it yet. What attributes should it contain? What should the keys be?
This type of problem would usually be solved with a system like ElasticSearch backed by an indexing engine like Lucene, but here we will use PostgreSQL and text processing code your professor has written for you.

Create a directory in your shared folder named `sql`. Create a file `schema.sql` that creates the relations in your schema. A SQL file is simply a collection of SQL queries that are executed in order. For your sanity, you are required to drop relations if they already exist. If you do not do this, Postgres will throw an error at each query saying the relation already exists.

You will be graded on using the correct data types without being wasteful, picking the correct keys, and of course, not yielding any errors or warnings.

Task 2: Load the Data

This task in some ways will allow you to check your work for any major problems from Task 1.

Now you will load the data into the schema you just created. There are a variety of ways to do this including command-line utilities, or writing a Python script with the `psycopg2` package. PostgreSQL uses the `\copy` (and/or `COPY` construct) command to read delimited files into Postgres pretty easily. Please take a look here for more information. You will want to pay attention to the file format, delimiter, quoting and header. This is similar to MySQL's `LOAD LOCAL INFILE` construct.

In your `sql` directory, create a file called `load.sql` that contains, one on each line, a `\copy` command that loads the three CSV files into Postgres.

Once you are finished, you can check the number of rows in each relation to make sure they match:

Relation Description	Expected Tuples
Artist info	643
Song info	57,650
Token info	5,375,064
TF-IDF scores	0

Late Submission Policy

Part 1A must be submitted by the deadline. See the syllabus for more information about the late policy.

Submission Instruction

Preparing Your Submission

Please create a folder named as your UID, put all your files into the folder, then compress this folder into a single zip file called `P1A.zip`. If you are working with a partner, **one** partner will use their UID as the name for the folder. We will figure out who the partner is from `TEAM.txt`.That is, the zip file should have the following structure.

```
P1A.zip
 |
 +- Folder named with Your UID, like "904200000" (without quotes)
     |
     +- schema.sql
     |
     +- load.sql
     |
     +- TEAM.txt
     |
     +- README.txt
```

Please note that the file names are case sensitive, so you should use the exact same cases for the file names. (For teamwork, use the submitter's UID to name the folder)

- `schema.sql`: A SQL file containing the schema to create the necessary relations.
- `load.sql`: A SQL file containing the `\copy` commands to load the data into Postgres. **Please make the paths absolute so we can check them automatically.**

- `TEAM.txt`: A **plain text** file that contains the UID(s) of every member of your team. If you worked by yourself, just include your UID. If you worked with a partner, write both UIDs on one line separated by a comma (,).
- README.txt: A **plain text** file that contains anything thing else you feel may be useful to us. If you had any problems getting something to work they way we want it to, include it in this file. While you may receive a point deduction, it is better than 0. This is also the file to include any citations.

Projects are submitted to CCLE, and only to CCLE.