CISC 4610 Fall 2020 Homework 1: SQL

Due 09/24/2020 (assignment designed by Prof. Michael Mandel)

Introduction

For this assignment, you will be interacting with a set of JSON documents in SQLite. The JSON documents are the output of the Google Cloud Vision API applied to images returned from a Flickr API query for interesting images related to the text "New York".

Starter code is provided in Python to load the JSON into the database and query it. You will complete this code with several SQL queries (both DDL and DML) and some Python code analogous to examples included in the starter code. You will submit your code, the output of your queries, and the database.

Setup

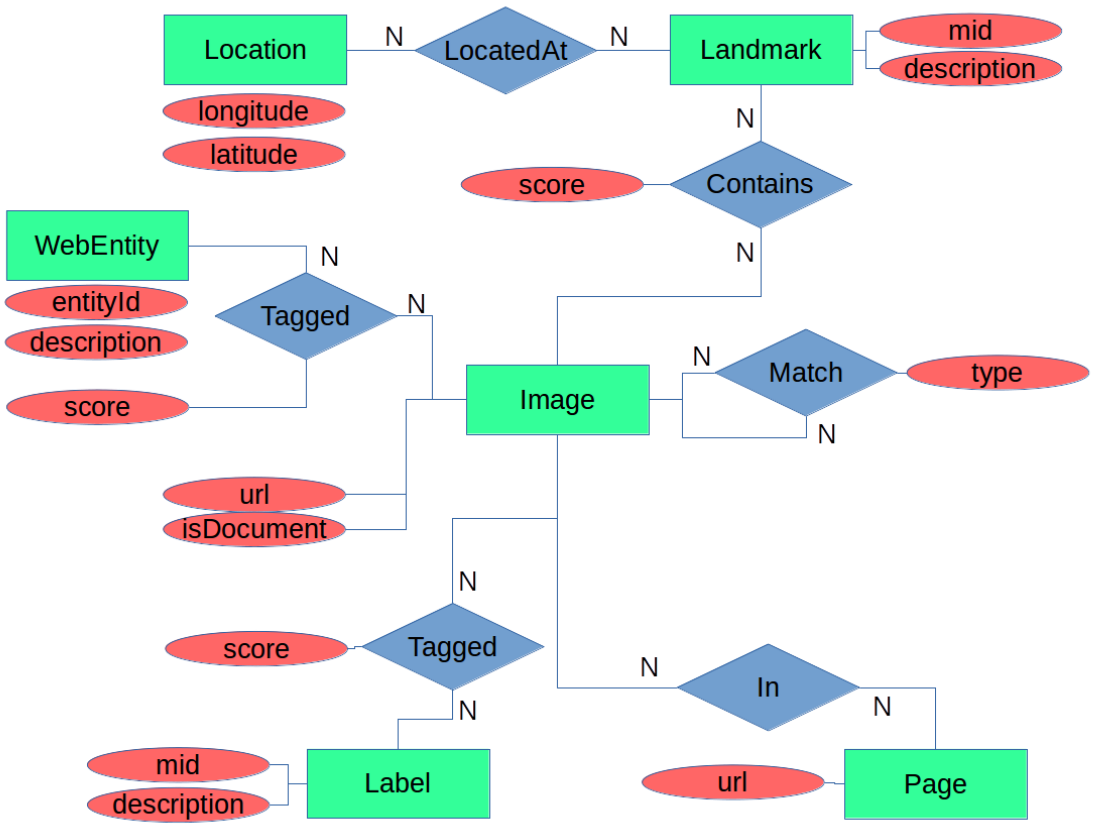
SQLite is a lightweight, SQL-compliant database that stores all its data in a single file. To get started, you need to make sure Python and a driver for SQLite are installed on your system (the starter code uses sqlite3, please stick with that).

1. Download and install Python 3.8 either directly from https://www.python.org/downloads/ or through the Software manager of your operating system. It is available directly in the Microsoft Store on Windows 10, for instance. For Linux and Mac, it may already be installed. If you already have an earlier version of Python 3 installed, that should work as well.
2. Download and install the sqlite3 module for Python. I recommend you use pip for that, which should only require you to run “pip install sqlite3” or “python -m pip install sqlite3” from a command line. If pip is not installed yet (on Linux it should be, on Windows it should be after installing Python), follow these instructions to set it up: https://pip.pypa.io/en/stable/installing/.
3. (optional) You might find it helpful to install an SQLite browser to interact with your database through a GUI. I recommend this one: https://sqlitebrowser.org/dl/ (note: you still have to create the tables and insert the data in code, not in the GUI).

Introduction to the data

The “data” folder contains the images returned by the Flickr API as well as the JSON documents returned by the Google Cloud Vision API. You do not need to do anything with the images for this assignment, it is all about processing the JSON data.

The file “exampleJson.txt” contains a JSON document that has all the fields that may be present in the individual JSON documents. Note that this is not a valid JSON document itself because in lists it only contains a single entry followed by “...”. Although individual JSON documents may not contain all these fields, if they do, they will be in the structure shown in “exampleJson.txt”. The entity-relationship diagram for the schema to be used in importing the data into SQLite is shown in the following figure:



The annotations come from the Google Cloud Vision API and are described here: https://cloud.google.com/vision/docs/reference/rest/v1/images/annotate#AnnotateImageResponse

However, only the following subset of annotations is included in the data:

- “landmarkAnnotations” -- identify geographical landmarks in photographs. For the purposes of discussing entities in the database schema, this will add “Landmarks”, each of which can have zero or more “Locations”.
- “labelAnnotations” -- predict descriptive labels for images. This will add “Label” entities to the schema.
- “webDetection” -- predict the presence of web entities (things with names) in images along with pages that contain the image and other images that are similar. This will add “WebEntity”, “Page”, and “Image” entities to the schema.

Introduction to the code

The Python code is contained in the file "runSqlite.py". If you have all dependencies installed, you should be able to run the script as it is to complete very basic versions of each of the tasks that you will complete:

1. Create the database schema
2. Populate the database
3. Query the database

If it works, it should print out (among other things):

Query 0

```
SELECT COUNT(*)

FROM image;

100
```

Tasks

To complete the assignment, perform the following tasks. For all code, follow the formatting style of the provided examples.

Write code to create the database tables

Implement the missing TODO entries in the "createSchema()" function. The schema should follow the above entity-relationship diagram. The meaning of most of the fields should be clear, but there are two that need some explanation:

- "isDocument" in the "image" table should be 1 if an image is the subject of the analysis in the JSON file and 0 if an image only appears in the "webDetection" fields.
- "type" in the "image_matches_image" table should be either "full" if the image appears in "webDetection.fullMatchingImages" and "partial" if it appears in "webDetection.partialMatchingImages"

Choose appropriate data types for all attributes and use lower case names with underscore-separation, as we did in class and as is done in the provided examples. Define primary keys and foreign keys as needed, following the provided examples. Note that you do not have to create indices for this task, although in a production database this would be highly recommended.

Write code to import the data

Implement the missing TODO entries in the "insertImage()" function. "insertImage()" is called with the JSON from the analysis of a single image by Google Cloud Vision API. The "json" module imports it so that JSON dictionaries become Python dictionaries, and JSON lists become python lists.

Use the helper function "getOrCreateRow()" in your implementation (you do not need to make any changes to this function). This function returns the ID of a tuple with the attributes provided in a dictionary. If such a tuple exists, it is returned. If it does not exist, it is created. This will ensure that you do not duplicate entries like "Labels" and "WebEntities".

If your code works properly, you should get the following counts of rows in each entity table (Query 2 below checks for this):

Entity	Count
image	560
label	201
page	556
landmark	18
location	26
web_entity	451

Write code to query the database

Implement the missing TODO entries in the "querySqlite()" function. Use the supplied "querySqliteAndPrintResults()" function to query the database. The queries should be as follows:

1. Count the total number of JSON documents in the database.
2. Count the total number of Images, Labels, Landmarks, Locations, Pages, and Web Entities in the database, listing the count for each separately.
3. List the URLs of all Images tagged with the Label with mid "/m/015kr" (which has the description "bridge"), ordered by the score of the association between them from highest to lowest.
4. List the 10 most frequent Web Entities that are associated with the same Images as the Label with an id of "/m/015kr" (which has the description "bridge"). List them in descending order of the number of times they appear, followed by their entity_id alphabetically.
5. Find all Images associated with any Landmarks that are not "New York" or "New York City", ordered alphabetically by landmark description and then by image URL.
6. List the 10 Labels that have been applied to the most Images along with the number of Images each has been applied to.
7. List the 10 Pages that are linked to the most Images through the webEntities.pagesWithMatchingImages JSON property along with the number of Images linked to each one. Sort them by count (descending) and then by page URL.
8. List the 10 pairs of Images that appear on the most Pages together through the webEntities.pagesWithMatchingImages JSON property. List them in descending order of the number of pages that they appear on together, then by the URL of the first image. Make sure that each pair is only listed once regardless of which is first and which is second.

Submission

Submit the following three files in a zip file named "<lastname>_1.zip" by 09/24/2020, 2:15pm:

1. Your completed version of "runSqlite.py".
2. A text file containing the output of your code.
3. Your populated database "sqlite.db".

On Collaboration and Academic Integrity

All code must be produced by yourself. You may, of course, discuss the assignment with other students but keep those discussions limited to concepts and ideas, do not write the actual code together. You may also look for help online but if you use code snippets found online, mark the source. Violations of these policies are subject to penalty.