CISC 4610 Fall 2020 Homework 2: Neo4j

Due 10/15/2020 (assignment designed by Prof. Michael Mandel)

Introduction

For this assignment, you will be interacting with a set of JSON documents in Neo4j. 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 Cypher queries. You will submit your code and the output of your queries.

Setup

Neo4j is a popular graph database management system that comes with a free desktop version. To get started, you need to install Neo4j and the neo4j driver for Python. The following instructions assume that you have already installed Python 3.8 and pip, which we used for Homework 1.

1. Download Neo4j Desktop from http://neo4j.com/download/ and unpack/install it.
2. Download and install the neo4j module for Python by opening a shell (command prompt in Windows) and running “pip install neo4j”.
3. Run Neo4j Desktop. When prompted, set the password for user “neo4j” to “cisc4610”. Add a new database called “neo4j” with the password set to “cisc4610” and start it.

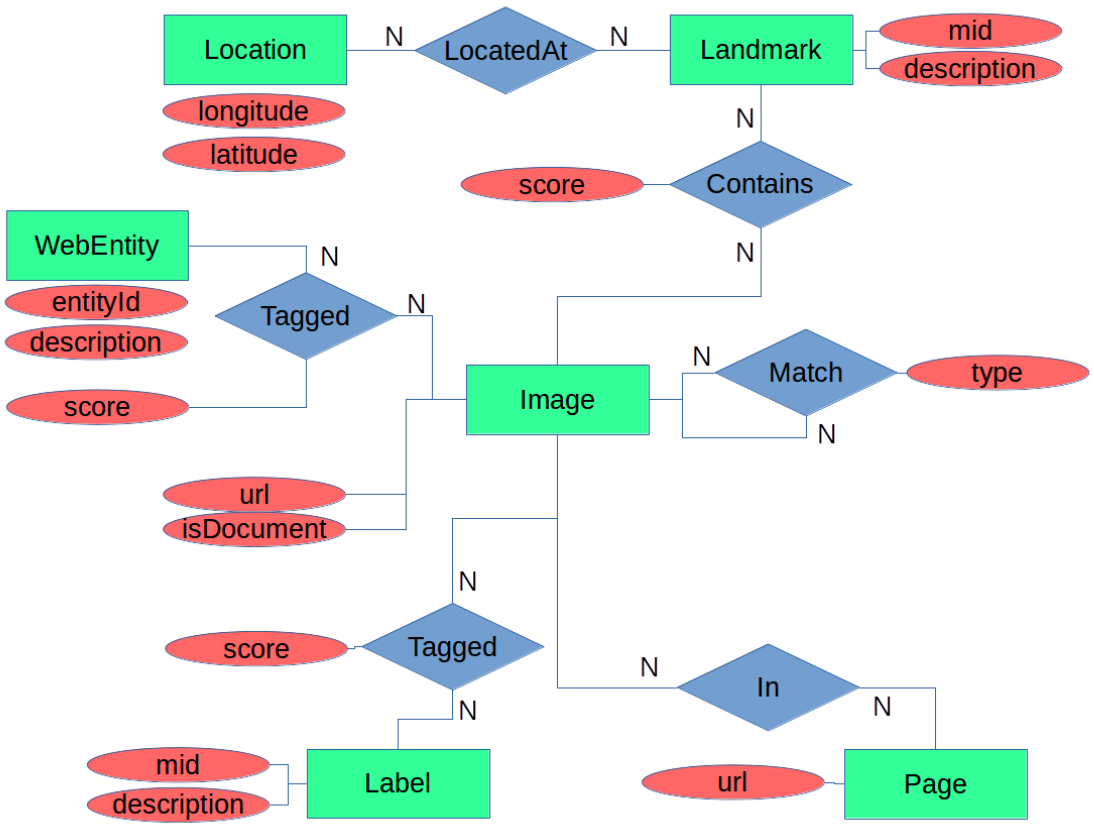
Getting started

After the setup steps, the starter code should run through successfully (run “python runNeo4j.py” from the directory where you unpacked the assignment zip file) and create the first 100 nodes. You can view your graph using the Neo4j Browser (click “Open” on the database in Neo4j Desktop after starting it). This link provides some information on how to use it: http://neo4j.com/developer/guide-neo4j-browser/. By running the following Cypher queries in the Neo4j Browser, you can get further introduction: “:play intro”, “:play concepts”, and “:play movie graph”.

Introduction to the data

The data and this description of it are the same as for homework 1. 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 “example.json” 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 "runNeo4j.py". If you have all dependencies installed, you should be able to run the script as it is to complete very basic versions of the tasks that you will complete:

1. Populate the database
2. Query the database

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

Query 0

```
MATCH (n:Image) RETURN count(n) as cnt
```

100

Tasks

To complete the assignment, perform the following tasks (each task is marked by a "TODO" in the code). For all code, follow the formatting style of the provided examples.

Write code to import the data

Complete the Cypher query stored in the string "insertQuery" (starting on line 36). "insertQuery" is run on each JSON document from the analysis of a single image by Google Cloud Vision API.

Sub-structure of the JSON document will be represented by different node types (nodes with different labels) in the database and these are created at the same time using a single, combined query in Neo4j's Cypher language composed of several "MERGE" clauses and several "FOREACH" loops.

The "MERGE" clause finds a node or relationship matching a given specification or creates it if it does not already exist. The syntax of the "MERGE" clause is:

```
MERGE (varName:LabelName {mustHaveProperty: value1})
  ON CREATE SET varName.optionalProperty = value2
  ON MATCH SET varName.optionalProperty = value3
```

In this case, the "MERGE" uses the name "varName" to refer to a node with a label of "LabelName" and with a property called "mustHaveProperty" that has a value of "value1". If no such node exists, it is created, and the "ON CREATE SET" clause is executed, setting the additional property of "optionalProperty" to "value2". If the node already exists, the "ON MATCH SET" clause is executed, setting "optionalProperty" to "value3". The "ON CREATE SET" and "ON MATCH SET" clauses are both optional. Multiple property-value pairs can be specified by separating them with commas.

The "MERGE" clause can also be used to find or create relationships:

```
MERGE (n1 {property1: value1})
MERGE (n2 {property2: value2})
MERGE (n1)-[relVal:REL_LABEL {property3: value3}]->(n2)
  ON CREATE SET relVar.optionalProperty = value4
  ON MATCH SET relVar.optionalProperty = value5
```

This will create or find two nodes having the specified property-value pairs. The nodes are called “n1” and “n2” for the purposes of this clause. The “MERGE” then finds or creates a relationship between those two nodes with a property “property3” having value “value3”, which it refers to as “relVar”. Again, if it is found the “ON MATCH SET” clause is executed and if it is created, the “ON CREATE SET” clause is executed. Notice that several “MERGE” clauses can follow one another to build up more complicated structures and relationships.

The “FOREACH” clause loops over the elements of an array, setting a variable to be each one in turn. In the case of a JSON array, it will loop over objects in the array allowing you to write a “MERGE” clause for each one. The syntax of the “FOREACH” statement is

```
FOREACH (jsonArrayElement in json.fields.array |
  MERGE (nodeName:LabelName {property: jsonArrayElement.value}))
```

Note the use of the extra parentheses around the whole statement except for the “FOREACH” keyword and the pipe character “|” separating the loop setup from the loop body. The JSON structure is navigated using field names separated by periods without any quotation marks or other punctuation, so the statement “json.fields.array” will index into the “array” object within the “fields” object within the “json” object. This statement will loop over the array “json.fields.array” setting a variable called “jsonArrayElement” to each value in turn and will then find or create a node called “nodeName” with label “LabelName” and property “property” equal to the property “value” from “jsonArrayElement”.

The official documentation of these clauses in the Cypher manual can be found here:

http://neo4j.com/docs/developer-manual/current/cypher/clauses/merge/

http://neo4j.com/docs/developer-manual/current/cypher/clauses/foreach/

After importing the JSON files into Neo4j, the total number of each label should be:

Entity	Count
Image	560
Label	201
Landmark	18
Location	26
Page	556
WebEntity	451

Write code to query the database

Implement the missing queries (marked by “TODO” comments) in the “queryNeo4j()” function using the “MATCH” clause (https://neo4j.com/docs/cypher-manual/current/clauses/match/). Most of them require you to count the number of matched nodes. Use the “count” function for this, which is described here: https://neo4j.com/docs/cypher-manual/current/functions/aggregating/#functions-count Use the supplied “queryNeo4jAndPrintResults()” function to query the database. The queries should be as follows (same as for homework 1):

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 node label separately.

3. List 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 two files in a zip file named “<lastname>_2.zip” by 10/15/2020, 2:15pm:

1. Your completed version of “runNeo4j.py”.
2. A text file containing the output of your code.

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.