CISC 4610 Fall 2020 Homework 3: MongoDB

Due 11/12/2020 (assignment designed by Prof. Michael Mandel)

Introduction

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

Setup

MongoDB is a popular document database management system that comes with a free desktop version. To get started, you need to install MongoDB and the pymongo driver for Python. The following instructions assume that you have already installed Python 3.8 and pip, which we used for Assignments 1 and 2.

1. Install MongoDB Community Edition for your platform following the instructions on this page: https://docs.mongodb.com/manual/administration/install-community/. Follow the instructions on the same page to start the database (mongod process). In Windows, this can be set up as a service during installation, so it runs automatically on startup.
2. Download and install the pymongo module for Python by opening a shell (command prompt in Windows) and running “pip install pymongo”.

Introduction to the data

The data and this description of it are the same as for assignments 1 and 2. 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 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 "runMongo.py". If you have all dependencies installed, you should be able to run the script as it is to populate the database and perform a basic aggregation query on it.

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

```
   Query 0. Count the total number of JSON documents in the database
{'documents': 100}

   Query 1. 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 score to lowest

***************** Aggregate pipeline ****************
[{'$unwind': {'path': '$response.labelAnnotations'}},
 {'$project': {'mid': '$response.labelAnnotations.mid',
               'score': '$response.labelAnnotations.score',
               'url': 1}},
 {'$match': {'mid': '/m/015kr'}},
 {'$sort': {'score': -1}},
 {'$project': {'_id': 0}}]
********************** Results **********************
{'mid': '/m/015kr',
 'score': 0.9771296381950378,
 'url': 'https://farm4.staticflickr.com/3394/5828289828_9f9bb8a45a.jpg'}
{'mid': '/m/015kr',
 'score': 0.976899266242981,
 'url': 'https://farm5.staticflickr.com/4238/34310239173_43c51169db.jpg'}
…
```

MongoDB review

We covered an overview of what you need to know to get started with MongoDB in Lecture 6 and I showed you a demonstration of MongoDB Compass at the end of Lecture 7. This section reviews some of that information.

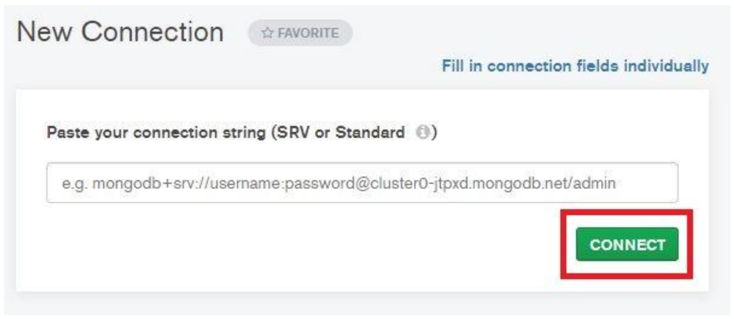
MongoDB stores *documents* in *collections* within *databases*. A document is a valid JSON object and is comparable to a row in a relational database. A collection is a group of related documents that do not necessarily share identical schemas. A database is a group of related collections. For the purposes of this assignment, we will be using a single collection within a single database. Each document will be the analysis result of one image from the Google Cloud Vision API.

There are three different frameworks that you can use for querying MongoDB collections: Single-purpose aggregation operations, the aggregation pipeline, and the MapReduce framework. You will develop an aggregation pipeline for each query except query 2, which uses the single purpose "distinct" operation.

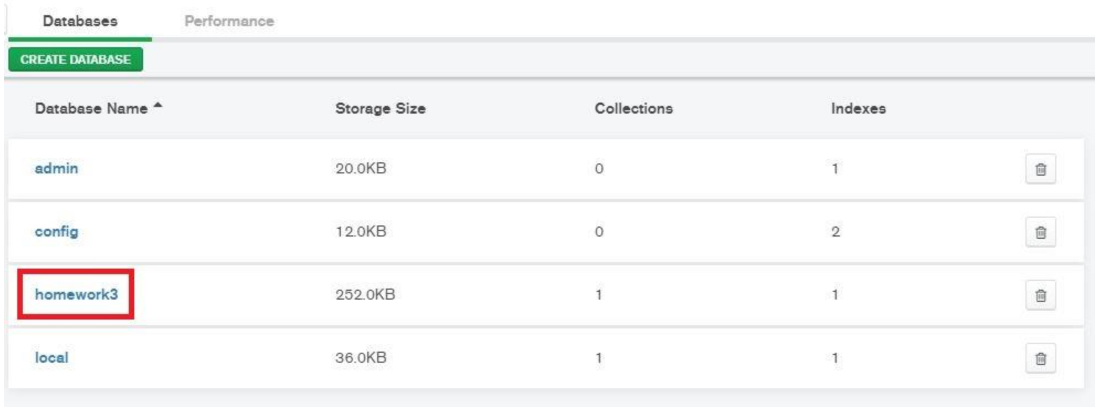
Queries using the aggregation pipeline are constructed using a sequence of *stages*, where each stage modifies or filters the set of documents as they "flow" through the pipeline. See the documentation here https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/ for a full list of stages. Note that in MongoDB syntax, stage and operator names are prefixed with the dollar sign character. Useful stages include "$project", "$match", "$group", "$sort", "$limit", "$unwind", "$set" (see link above for details).

Creating an aggregation pipeline

As discussed at the end of lecture 7, I suggest that you use the MongoDB Compass application to develop your aggregation pipelines. The following screenshots illustrate how to do this:

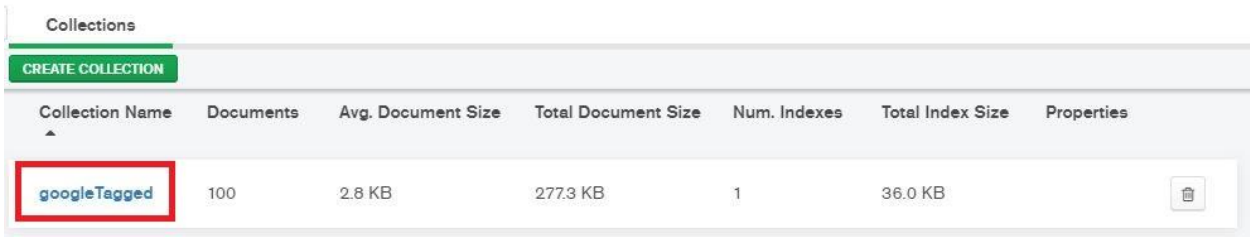

New Connection
☆ FAVORITE
Fill in connection fields individually
Paste your connection string (SRV or Standard)
e.g. mongodb+srv://username:password@cluster0-jtpxd.mongodb.net/admin
CONNECT

1 Connect to the local database server



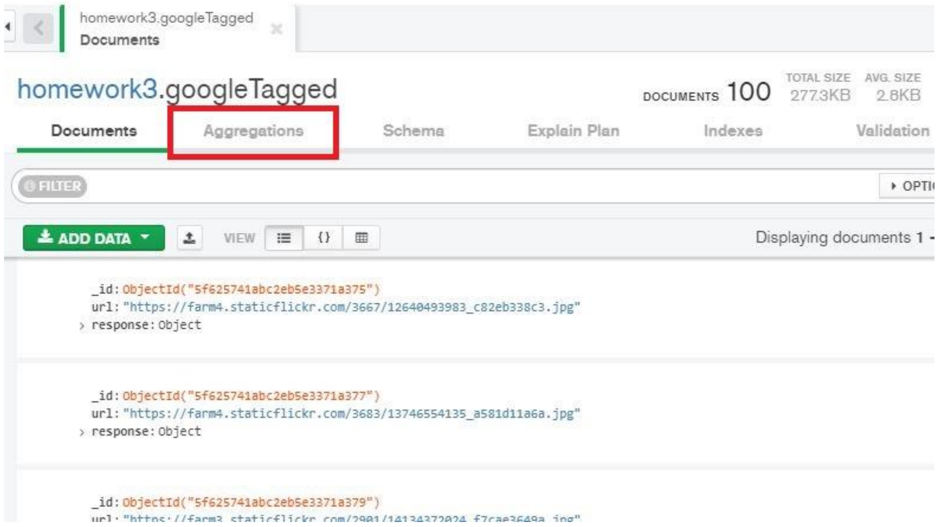
Database Name	Storage Size	Collections	Indexes
admin	20.0KB	0	1
config	12.0KB	0	2
homework3	252.0KB	1	1
local	36.0KB	1	1

2 Open the homework database



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
googleTagged	100	2.8 KB	277.3 KB	1	36.0 KB	

3 Open the collection

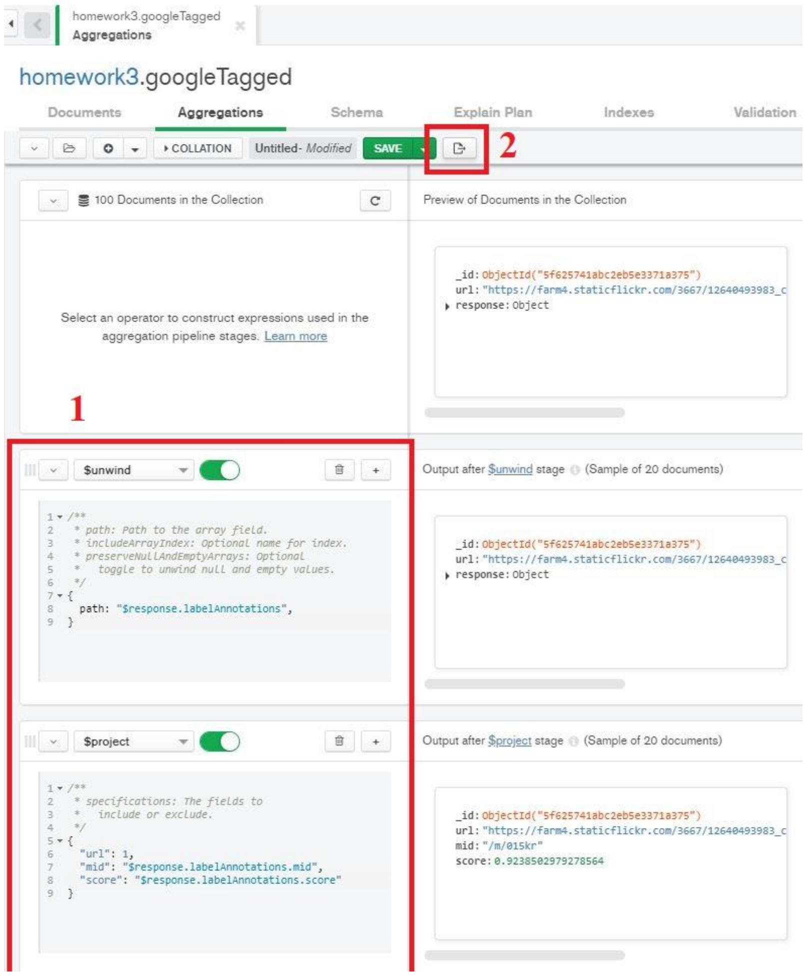


```
_id: ObjectId("5f625741abc2eb5e3371a375")
url: "https://farm4.staticflickr.com/3667/12640493983_c82eb338c3.jpg"
› response: Object

_id: ObjectId("5f625741abc2eb5e3371a377")
url: "https://farm4.staticflickr.com/3683/13746554135_a581d11a6a.jpg"
› response: Object

_id: ObjectId("5f625741abc2eb5e3371a379")
url: "https://farm3.staticflickr.com/2901/14134372024_f7cae3649a.jpg"
```

4 Switch to the Aggregations tab


homework3.googleTagged
Aggregations
homework3.googleTagged
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
COLLATION
Untitled- Modified
SAVE
2
100 Documents in the Collection
Preview of Documents in the Collection
1
$unwind
Output after $unwind stage (Sample of 20 documents)
$project
Output after $project stage (Sample of 20 documents)

Select an operator to construct expressions used in the aggregation pipeline stages. Learn more

```
_id: ObjectId("5f625741abc2eb5e3371a375")
url: "https://farm4.staticflickr.com/3667/12640493983_c
▸ response: Object
```

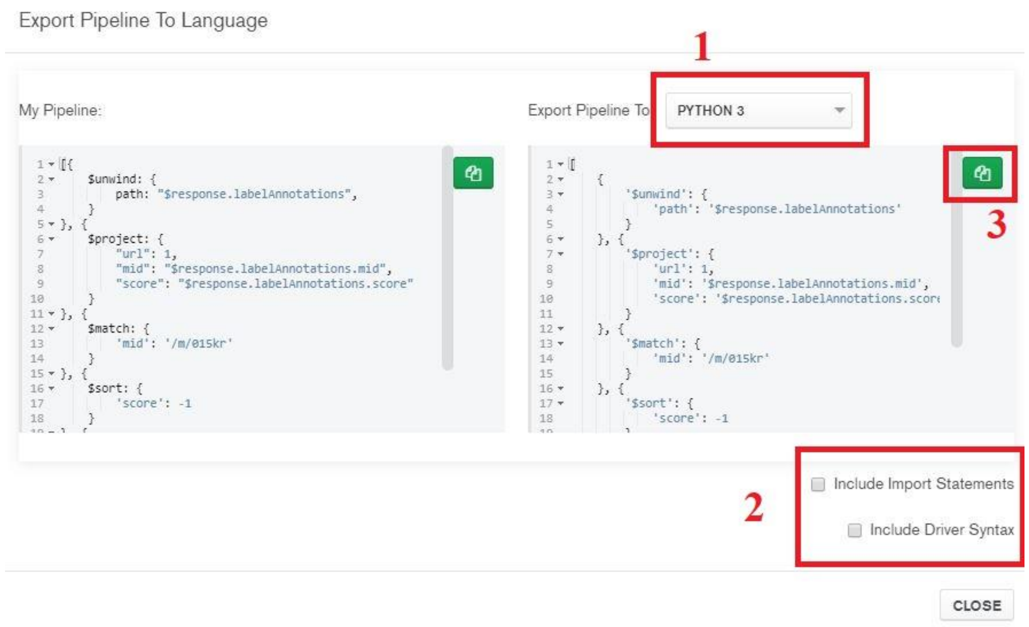
```
/**
 * path: Path to the array field.
 * includeArrayIndex: Optional name for index.
 * preserveNullAndEmptyArrays: Optional
 *   toggle to unwind null and empty values.
 */
{
  path: "$response.labelAnnotations",
}
```

```
_id: ObjectId("5f625741abc2eb5e3371a375")
url: "https://farm4.staticflickr.com/3667/12640493983_c
▸ response: Object
```

```
/**
 * specifications: The fields to
 *   include or exclude.
 */
{
  "url": 1,
  "mid": "$response.labelAnnotations.mid",
  "score": "$response.labelAnnotations.score"
}
```

```
_id: ObjectId("5f625741abc2eb5e3371a375")
url: "https://farm4.staticflickr.com/3667/12640493983_c
mid: "/m/015kr"
score: 0.9238502979278564
```

5 Create your pipeline ("Add stage" and configure), then click the Export button



My Pipeline:

```
[{
    $unwind: {
        path: "$response.labelAnnotations",
    }
}, {
    $project: {
        "url": 1,
        "mid": "$response.labelAnnotations.mid",
        "score": "$response.labelAnnotations.score"
    }
}, {
    $match: {
        'mid': '/m/015kr'
    }
}, {
    $sort: {
        'score': -1
    }
```

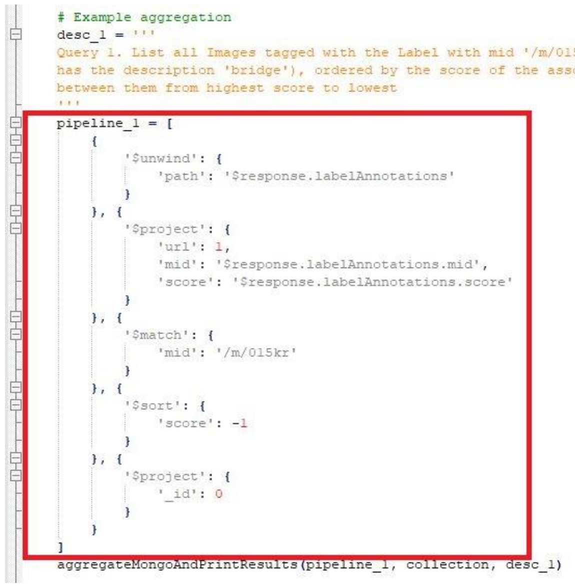
Export Pipeline To: PYTHON 3

```
[
    {
        '$unwind': {
            'path': '$response.labelAnnotations'
        }
    }, {
        '$project': {
            'url': 1,
            'mid': '$response.labelAnnotations.mid',
            'score': '$response.labelAnnotations.scor
        }
    }, {
        '$match': {
            'mid': '/m/015kr'
        }
    }, {
        '$sort': {
            'score': -1
```

☐ Include Import Statements

☐ Include Driver Syntax

6 Make sure language is set to "Python 3" and checkmarks are unchecked, then copy the code.



```
# Example aggregation
desc_1 = '''
Query 1. 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 score to lowest
'''
pipeline_1 = [
    {
        '$unwind': {
            'path': '$response.labelAnnotations'
        }
    }, {
        '$project': {
            'url': 1,
            'mid': '$response.labelAnnotations.mid',
            'score': '$response.labelAnnotations.score'
        }
    }, {
        '$match': {
            'mid': '/m/015kr'
        }
    }, {
        '$sort': {
            'score': -1
        }
    }, {
        '$project': {
            '_id': 0
        }
    }
]
aggregateMongoAndPrintResults(pipeline_1, collection, desc_1)
```

7 Paste the code in runMongo.py, making sure it is indented as needed

Tasks

Note that because MongoDB does not use an explicit schema, there is no schema creation step, as there was in homework 1. Note also that inserting into the database is just a matter of inserting each of the JSON documents into a single collection and **the starter code already does that** in "populateMongo()".

To complete the assignment, solve the queries marked with "TODO" in the code.

Write code to query the database

Implement the missing queries (marked by “TODO” comments) in the “queryMongo()” function. For all except query 2 (the first one in the list below), use aggregation pipelines, either writing them directly or using the recommended procedure outlined above. The queries should be as follows (not the same as assignments 1 and 2):

2. Count the total number of distinct Labels (by mid), Landmarks (by mid), Locations (by latitude and longitude), Pages (by url), and Web Entities (by entityId) in the database (note that images are counted in a separate query below). Use the single purpose *distinct* function and Python’s built-in *len* function for this query. See the TODO comment for this query for further details.
3. List the 10 Labels that have been applied to the most Images along with the number of Images each has been applied to in descending order.
4. 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), followed by page URL.
5. List the descriptions of all Landmarks in alphabetical order, with each description appearing only once.
6. List all Images that have at most 5 Labels *and* at most 5 Web Entities associated with them. List them in descending order of the number of Labels, followed by descending order of the number of Entities. List only the Image URLs and the numbers of Labels and Entities.
7. List all Images that have at least 1 Landmark associated with them. List them in descending order of the number of Landmarks, followed by their URL alphabetically. List only the Image URLs and the numbers of Landmarks.
8. List the 10 Images with the greatest number of Image matches of either type (partial or full). List them in descending order of the number of matches, followed by their URL alphabetically. List only the Image URLs and the numbers of matches.
9. 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 entityId alphabetically.
10. Find all Images associated with any Landmarks that are not 'New York' or 'New York City' with an association score of at least 0.6, ordered alphabetically by Landmark description and then by image URL.
11. Count the total number of unique Images in the database. This should include both those that have been directly submitted to the Google Cloud Vision API as well as those that are referred to in the returned analyses.
12. Some Images have Landmarks and Web Entities with the same description associated with them. That is, some Landmarks are also Web Entities or vice versa. List the distinct descriptions of all Landmarks like that, i.e., Landmarks associated with at least one Image which is also associated with a Web Entity of the same description as the Landmark.
13. 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>_3.zip” by 11/12/2020, 2:15pm:

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

On Collaboration and Academic Integrity

All code must be produced by yourself (which includes the MongoDB Compass procedure described above). 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 and *do not configure the stages in MongoDB Compass together*. You may also look for help online but if you use code snippets etc. found online, mark the source. Violations of these policies are subject to penalty.