THE UNIVERSITY OF SYDNEY

School of Computer Science
Dr. Ying Zhou

COMP5338: Advanced Data Models

Sem. 2/2020

MongoDB Assignment

08.09.2020

1 Introduction

In this assignment, you will demonstrate that you are able to work with MongoDB by writing practical queries to explore real world data and their relations.

The data set you will work with consists of tweet objects downloaded from **Twitter** using `Twitter API`. You are asked to implement a set of target queries to explore different features of the data. A practice data set is released along with the assignment instruction. The expected answer of each query for the practice data set will be given for you to self check the correctness of the implementation. A test data set will be used by markers to mark the implementation. The practice and the test data set are of similar size and format.

2 Data set

The practice data set contains a single JSON file downloaded using keyword search of twitter's standard `search` API. The JSON file contains around 10K `tweet` objects. A tweet object may refer to a *general tweet*, a *retweet* or a *reply* to a tweet. A *general tweet* is "a message posted to Twitter containing text, photos, a GIF, and/or video"[1]. A reply is a response to another person's tweet[2]. "A Retweet is a re-posting of a Tweet."[3] Note that *reply* or *retweet* could happen not only on *general tweet*, but also on other *reply* and/or *retweet*. In other words, there could be *reply* to a *retweet* or a *reply*; and *retweet* of a *reply*. In this assignment, we may refer to tweet that receives a reply or retweet as `parent tweet`. This is not a standard terminologies used by Twitter or general public, rather a borrowed term from tree data structure.

Below is an example tweet object representing a *retweet*:

[1] About different types of Tweets General Tweets
[2] About different types of Tweets General Tweets
[3] Retweet FAQs

```
{
    "id": 1300296290924929024,
    "created_at": "2020-08-31 04:56:08",
    "text": "RT @LewisForMN: Before Biden changed his story and said he
       DIDN'T take a cognitive test, he said he DID take a cognitive
       test. \n\nIf he didn't",
    "user_id": 1104621094122737664,
    "retweet_count": 20,
    "favorite_count": 0,
    "retweet_id": 1300264093270540288,
    "retweet_user_id": 20934272,
    "user_mentions": [
        {
            "id": 20934272,
            "indices": [3,14]
        }
    ]
}
```


The common fields in all tweet objects are:

- `id`: the unique id of the tweet.
- `created_at`: the date and time the tweet is created.
- `text`: the textual content of the tweet
- `user_id`: the id of the user who created the tweet.
- `retweet_count`: retweet count of the tweet or its parent tweet if the tweet is a *retweet*
- `favorite_count`: favorite count of the tweet

The optional fields in a tweet object are:

- `retweet_id`: if the tweet is a *retweet* of another tweet, this field contains the `parent` tweet's id
- `retweet_user_id`: if the tweet is a *retweet* of another tweet, this field contains the `parent tweet's user_id`
- `replyto_id`: if the tweet is a *reply* to another tweet, this field contains the `parent` tweet's id

- `replyto_userid`: if the tweet is a *reply* to another tweet, this field contains the `parent tweet`'s user id.
- `user_mentions`: an array of {id, indices} showing the id of the users mentioned in the tweet text, as well as the location this user is mentioned.
- `hash_tags`: an array of {tag, indices} showing the hash tag appearing in the tweet text and the location it appears

Note that in releasing the twitter data for assignment use, we follow the content and size limitation imposed by content redistribution policy as part of the Twitter API developer agreement and policy. It is important that you do not redistribute the data set used in the assignment to any party outside this course.

3 Query workload

- **[Q1]** Find out the number of *general tweets*, *replies* and *retweets* in the data set. A *general tweet* is a tweet with no `replyto_id`, nor `retweet_id` field; a *reply* is a tweet with the `replyto_id` field; a *retweet* is a tweet with the `retweet_id` field.

  Below are sample answers for the practice data set :

General Tweet	*Reply*	*Retweet*
1035	1148	7817

- **[Q2]** Find out the top 5 hashtags sorted by their occurrence in *general* or *reply* tweets. We do not count retweet, which has the same textual content as the `parent tweet`.

  The sample answers for the practice data set are:

Yosemite	yosemite	YoSemite	NationalPark	California
47	47	16	10	9

  Note that the order does not matter if a few hashtags have the same occurrence number. The 5^{th} tag in the sample result could be 'nature',which also occurs 9 times.

- **[Q3]** Find out the tweet taking the longest time to receive a reply; print out the id and the duration between the tweet's creation time and the creation time of its first reply in `second`.

  The sample answers for the practice data set are:

```
id: 1298008149551587328; first response in: 75952s
```

- [**Q4**] Find out the number of *genera*l and *reply* tweets that do not have all their retweets included in the data set. Note that the total number of retweets a tweet is stored in the field `retweet_count`.

  Sample answer for the practice data set is: `139`

- [**Q5**] Find out the number of tweets that do not have its `parent tweet` object in the data set.

  The sample answer for the practice data set is: 6924

- [**Q6**] Find out the number of *general tweets* that do not have a reply nor a retweet in the data set.

  The sample answer for the practice data set is: 824

4 Implementation Requirements

The given json file should be initially imported to a collection called `tweets`, belonging to a database called `a1`. This collection should not be updated in any subsequent queries.

The recommended practice is to make a copy of the `tweets` collection using a name of your own choice. You can make necessary updates, such as type change when duplicating `tweets` collection. You can create other auxiliary collections if necessary. All subsequent read and write queries should happen in the newly created collection(s).

Implementation of all target queries should be packaged as a single `mongo` shell script, which is a `JavaScript` file. The script should be executed by `mongo` shell in the form of `mongo script.js`. A sample script `a1sample.js` is released along with the assignment instruction. This script assumes the practice data has been imported to a collection called `tweets` in a database called `a1`. Your script should make the same assumption and can reuse the first few statements in the sample to connect to the database server and to specify the database.

Implement all target queries as MongoDB query or aggregation commands. Ideally, you should implement each target query as a single command. You can modify the document structure by adding fields that will help implementing the target queries. All such modifications, including computations should be carried out as MongoDB query or aggregation command as well. You can also create index to improve the performance.

At the end of the script, include command(s) to remove all collections you have created and leave the database back to the initial clean state with only a `tweets` collection. An example can be seen from the last line in `a1sample.js`, which drops the collection `tweets_v2` created at the beginning of the script.

Running your script should produce the result of each query in correct order with results for different queries clearly labelled and delimited. Below is a sample output:

```
Q1=======
{
    "general" : 1035,
    "reply" : 1148,
    "retweet" : 7817,
}
Q2======
{
    "Yosemite" : 47,
    "yosemite" : 47,
    "YoSemite" : 16),
    "NationalPark" : 10),
    "California" : 9)
}
Q3=======
{
    "id" : 1298008149551587328,
    "duration" : 75952
}
Q4=======
    139
Q5=======
    6924
Q6=======
    824
```

You don't need to follow the exact format. In fact, the above output is not in the original format produced by running a script on `mongo` shell.

5 Deliverable and Submission Guidelines

The final deliverable of this assignment consists of two elements:

- A `mongo` shell script including all queries/aggregations as described in section 4. The shell script is due for submission on Canvas by **week 5 Monday September 21, 2020 23:59pm**.
- A **5-10 minutes** zoom demo with your tutor in week 5. In the demo the **tutor** will run your submitted script on a test data set of similar size and of the same format. It is essential that your script does not modify the data in the original `tweets` collection

and your script should also remove new collections created by you. Your tutor will ask you a few questions on implementation details. The demo will happen outside lab time and your tutor will publish the schedule later.

6 Mark Distribution

The target query is worth 1.5 points each. The marking is based on whether or not your query produces the correct results on the test data set during the demo. The test data set is collected using the same Twitter API with different search term(s). There is no intentionally added corner case(s) in the test data set.
There is 1 point allocated for script conforming to the required format. Your script needs to produce at least the correct result following the implementation requirements to receive the format point.

7 Resources

- Query and Projection operators
- Visual Studio Code is an alternative client tool that helps better type setting MongoDB command. A brief installation and usage guide has been included in week1's MongoDB setup guide. Please check the document for updated content Canvas lab resources.