THE UNIVERSITY OF SYDNEY

School of Computer Science
Dr. Ying Zhou

COMP5338: Advanced Data Models Sem. 2/2021

MongoDB Assignment

03.09.2021

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. You will also demonstrate that you understand how MongoDB executes queries and aggregations, particularly how indexes may help to improve execution performance.

The two data sets you will work with consist of tweet and user objects downloaded from **Twitter** using `Twitter API`. They are of similar structure but slightly larger than those used in the labs. You are asked to implement a set of target workloads to explore different features of the data. You are also asked to analyse the execution statistics of each implementation and to compare different implementations of two workloads. Two practice data sets are released along with the assignment instruction. The expected results of each query for the practice data set are given in the assignment instruction to help you self-check the correctness of the implementation. A test data set will be used by markers to mark the implementation. The practice and test data sets are of similar size and format.

2 Data set

Each data set consists of two single JSON files downloaded using keyword search of Twitter's standard `search` API. The `tweets-xxx.json` file contains around 10K `tweet` objects. A tweet object may refer to a *general tweet*, a *retweet* or a *reply* to a tweet. 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 a tweet that receives a reply or retweet as a `parent tweet`. This is not standard terminology used by Twitter or the general public; rather it is a term borrowed from the tree data structure lexicon. The `users-xxx.json` file contains corresponding `user` objects. The `user` object is referenced by `user_id` in the the `tweet` object.

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

```
{
    "_id" : ObjectId("61302528a4bbc543d84e79e1"),
    "id" : NumberLong(1433006910895054859),
    "created_at" : "2021-09-01 10:00:46",
    "text" : "RT @OnTheRoad34: Yosemite National Park https://t.co/nw6
       LIlc9pr",
    "user_id" : 115473990,
    "retweet_id" : NumberLong(1432931738926391297),
    "retweet_user_id" : NumberLong(1420751959921807363),
    "user_mentions" : [
        {
            "id" : NumberLong(1420751959921807363),
            "indices" : [3, 15]
        }
    ]
}
```

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.

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

Below is the corresponding user object (all user objects have the same set of fields):

```
{
    "_id" : ObjectId("6130253b1c1541d0961ed70d"),
    "id" : 115473990,
    "name" : "Lars Lind",
    "screen_name" : "thevikingman",
    "location" : "",
    "description" : "",
    "created_at" : "2010-02-18 19:33:30",
    "favourites_count" : 107067,
    "friends_count" : 4872,
    "followers_count" : 1189
}
```

Note that in releasing the Twitter data for assignment use, we follow the content and size limitations imposed by the 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 this assignment to any party outside this course.

3 Query workload

- **[Q1]** Find the number of *general tweets* with at least one reply and one retweet in the data set. Note that a *general tweet* is a tweet with neither a `replyto_id` field, nor a `retweet_id` field; a *reply* is a tweet with the `replyto_id` field; a *retweet* is a tweet with the `retweet_id` field.

- **[Q2]** Find the *reply tweet* that has the most *retweets* in the data set.

- **[Q3]** Find the top 5 hashtags appearing as the FIRST hashtag in a *general* or *reply* tweet, ignoring the case of the hashtag. Note that the order does not matter if a few hashtags have the same occurrence number.

- **[Q4]** For a given `hash_tag`, there are many tweets including that `hash_tag`. Some of those tweets mention one or many users. Among all users mentioned in those tweets, find the top 5 users with the most `followers_count`. For each user, you should print out the `id`, `name` and `location`. Not all users have a profile in the users data set; you can ignore those that do not have a profile. If there are less than 5 users with profile, print just those users with a profile.

- [**Q5**] Find the number of *general tweets* published by users with neither location nor description information.
- [**Q6**] Find the general tweet that receives most retweets in the first hour after it is published. Print out the tweet Id and the number of retweets it received within the first hour.

4 Implementation Requirements

The given json files should be initially imported into two collections called `tweets` and `users` respectively, belonging to a database called `a1`. These collections must not be updated in any subsequent queries.

The recommended practice is to make a copy of these collections using names of your own choice. You can make necessary updates, such as type change when duplicating the `tweets` or `users` collection. You may also remove fields that will not be used in any workload. All such modifications should be carried out as MongoDB queries or aggregation commands. NO client side scripting is allowed. You should also create indexes to improve query performance. There should be at most two newly created collections, one for each original collection.

Implement each target workload as a **SINGLE** MongoDB `find` or `aggregate` command. If you implementation contains many queries/aggregations, only the first one will be marked. If the first query/aggregation does not produce any useful intermediate result, you may not get any point for that implementation.

In addition, you should provide an alternative implementation for `Q4` and `Q6`. Each alternative implementation should be expressed as a **SINGLE** MongoDB `find` or `aggregate` command. It should have expressions that are substantively different to those in the original one. Changing only a query condition from `"$exists:true"` to `"{$ne: null}"` is not considered as a valid alternative implementation for this assignment. For aggregation, an alternative implementation may use different stages or arrange stages in a different order.

Implementation and alternative implementation of all target workloads 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 --eval "args=<hash_tag>"`. The argument part is the input for `Q4`. A sample script `a1sample.js` is released along with the assignment instruction. This script assumes the practice data has been imported into the collections `tweets` and `users` in a database called `a1`. Your script should make the same assumption and can reuse the first few statements in the sample script to connect to the database server and to specify the database.

At the end of the script, include command(s) to remove all collections you have created and return the database to the initial clean state with only the `tweets` and `users`

collections. An example can be seen from the last line in `a1sample.js`, which drops the collections `tweets_v2` and `users_v2`, which were created at the beginning of the script.

Running your script should produce the result of each query in the correct order with results for different queries clearly labelled and delimited. See sample results section for a suggested format. The sample does not include the results of alternative implementations, which should be presented after the original Q6 result.

5 Report

Your report must contain the following sections:

- Introduction
- Performance analysis of query implementations
- Conclusion

The introduction section should include a brief overview of what you will analyse in your report.

For each target query, analyse the performance of your implementation (this includes both the first and alternative implementations for Q4 and Q6). If your implementation uses the find command, analyse the various candidate plans evaluated by MongoDB and the winning plan. You should highlight the index usage to justify any index created. If your implementation uses an aggregation command, provide a detailed analysis of each stage, highlighting the index usage and memory usage in certain stages. You can also include other useful information such as documents examined in different stages. You may include relevant screenshots from MongoDB's explain method output. You need to consider the execution statistics in your analysis.

Your conclusion should include a summary of your key findings.

Your report must be in Academic English. Simply copying and pasting screenshots without providing an analysis, or writing mainly in bullet points is not acceptable.

6 Deliverable and Submission Guidelines

The final deliverable of this assignment consists of the following:

- A `mongo` shell script including all queries/aggregations as described in section 4. The deadline for submitting the shell script in Canvas is **23 September 2021 (Thursday, week 7) 23:59 Sydney time (13:59 UTC)**.
- A **pdf file** containing your report. The deadline for submitting this file is the same as the deadline for submitting the shell script.

- A **5-10 minutes** zoom demo with your tutor in week 7 or week 8. In the demo, the **tutor** will run your submitted script on a test data set of similar size and of the same format as the data sets you have been given. It is essential that your script does not modify the data in the original `tweets` and `users` collections 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.

7 Mark Distribution

- Implementation of target query: **2 points** each
- Alternative implementation of **Q4** and **Q6**: **1 point** each
- Performance analysis of implementation and alternative implementation: **0.5 point** each
- Script formatting and compliance with output requirements: **1 point**
- Overall report quality: **1 point**

8 Sample Results

There are four json files representing two data sets:

- `tweets_hurricane.json` and `users_hurricane.json` are downloaded from Twitter using search term "hurricane". We refer to them as the "hurricane" data set.
- `tweets_yosemite.json` and `users_yosemite.json` are downloaded from Twitter using search term "yosemite". We refer to them as the "yosemite" data set.

Sample results for the “hurricane” data set

```
Q1=======
{ "Number of tweets" : 10 }
Q2======
{ "id" : NumberLong("1432553467466317828"), "retweet_count" : 8 }
Q3=======
{ "tag" : "ida", "count" : 25 }
{ "tag" : "hurricaneida", "count" : 18 }
{ "tag" : "hurricane_ida", "count" : 14 }
{ "tag" : "vivopklplayerauction", "count" : 10 }
{ "tag" : "hurricane", "count" : 10 }
Q4=======
input tag: Ida
{
    "id" : 19071682,
    "name" : "Breaking Weather by AccuWeather",
    "location" : "State College, PA",
    "followers_count" : 1044045
}
{
    "id" : 15791186,
    "name" : "KHOU 11 News Houston",
    "location" : "Houston, TX",
    "followers_count" : 730057
}
{
    "id" : 22032260,
    "name" : "Platts Oil",
    "location" : "London",
    "followers_count" : 133084
}
{
    "id" : 14427407,
        "name" : "#WVTM13",
        "location" : "Birmingham, AL",
        "followers_count" : 71972
}
{
        "id" : 259595987,
        "name" : "Simon Brewer",
        "location" : "Ohio",
        "followers_count" : 27351
}
Q5=======
    tweet_count: 114
Q6=======
    { "id" : NumberLong("1432568863334539264"), "retweet_count" : 37 }
```

Sample results for the “yosemite” data set (non-printable characters are not included)

```
Q1 ==========
{ "Number of tweets" : 23 }
Q2 ============
{ "id" : NumberLong("1431646921270210560"), "retweet_count" : 148 }
Q3 ======
{ "tag" : "yosemite", "count" : 99 }
{ "tag" : "travel", "count" : 8 }
{ "tag" : "android", "count" : 7 }
{ "tag" : "yosemitenationalpark", "count" : 6 }
{ "tag" : "photography", "count" : 5 }
Q4 ========
input tag: Yosemite
{
        "id" : 18726942,
        "name" : "Yosemite National Park",
        "location" : "Yosemite National Park, CA",
        "followers_count" : 747386
}
{
        "id" : 16261627,
        "name" : "KCBS 106.9 FM/740 AM",
        "location" : "San Francisco",
        "followers_count" : 48953
}
{
        "id" : 360930955,
        "name" : "Day Hiking Trails",
        "location" : "",
        "followers_count" : 25307
}
{
        "id" : 1928841116,
        "name" : "Jeff Kuhn, MBA",
        "location" : "Los Angeles / Global",
        "followers_count" : 24890
}
{
        "id" : 72160503,
        "name" : "wentworth falls",
        "location" : "Sydney",
        "followers_count" : 4029
}
Q5 =======
{ "tweet_counts" : 52 }
Q6 =========
{ "id" : NumberLong("1432853709843800064"), "retweet_count" : 97 }
```