CS4417 Assignment 3 - MongoDB

The goal of this assignment is for you to gain familiarity with MongoDB, one of the most widely-used tools for the management and querying of big, unstructured data.

MongoDB uses the JSON (and related BSON) format for data, as we saw in class.
The MongoDB shell provides an interactive JavaScript interface to MongoDB.
We will provide JavaScript templates for each question so that you can see how the syntax works.

The mongodb manual is here: https://docs.mongodb.com/manual/mongo/

Installing MongoDB

There are various ways of installing MongoDB on different systems. You can use any approach you want. We are providing detailed instructions using `docker` which should work in a platform-independent fashion. You will need at least a few GB of free hard drive space to install.

Installing using Docker

Windows alert: If after installing docker desktop and restarting the PC you get a message that states, "WSL 2 installation is incomplete," then going to https://docs.microsoft.com/en-us/windows/wsl/install-win10#step-4---download-the-linux-kernel-update-package and installing `wsl_update_x64` and then restarting the PC should fix this problem.

First, install https://www.docker.com/ Docker Desktop or Engine. We'll use the Docker command line https://docs.docker.com/engine/reference/commandline/docker/ to install mongodb. From a command prompt on your machine:

Make sure docker has fully started before issuing the following commands.

Download a docker image containing mongodb.

```
docker pull mongo:4.4
```

Create a new container from this image and run it.

```
docker run --name my-mongo -d mongo:4.4
```

Running `docker ps` should show something like

```
CONTAINER ID   IMAGE       COMMAND                  CREATED
STATUS          PORTS       NAMES
4a442375f0ff   mongo:4.4   "docker-entrypoint.s…"   32 seconds ago   Up
32 seconds   27017/tcp   my-mongo
```

You can now start a bash shell from within your docker container like so:

```
docker exec -it my-mongo bash
```

If you run into trouble with docker and the `docker` commands stop working, you can re-start it using "Restart Docker" in the Docker Desktop application. You can also use it to see all of the containers you've created and delete any that you don't need. You can make a new container using the `docker run` command above.

Importing the Tweet Data

First, download the tweets data from https://www.csd.uwo.ca/~dlizotte/tweets/. (Right-click and "Save as..." or "Download Linked File As...", or if you have `curl` you can use `curl https://www.csd.uwo.ca/~dlizotte/tweets.json -o tweets.json` to get it but curl on my Windows machine was very slow.)

If you are using docker, copy the `tweets.json` file to your container. From the command prompt of your operating system (for this command, *not* the prompt in your docker container) in the folder where you saved the `tweets.json` file, run the following command.

```
docker cp tweets.json my-mongo:/
```

From this point onward, we assume you are running a shell, either in the docker container as described above or your machine's command prompt, if you have installed mongodb natively.

You can now import our example twitter data, contained in the `tweets.json` file, by typing

```
mongoimport --db tweetdb --collection tweets --file tweets.json
```

MongoDB organizes data into *databases* and *collections*. In this case, you will use the *tweetdb* database and the *tweets* collection of documents.

MongoDB uses its own shell that allows users to run queries. To start the shell, run the following command:

```
mongo
```

Once you have started the shell, to access the *tweetdb* database, type

```
use tweetdb
```

Having done so, you can now access the *tweets* collection. For example, to print an example tweet, type

```
db.tweets.findOne()
```

Have a look at the resulting JSON object. *You may want to review the JSON slides in the Structure of Unstructured Data topic on OWL.*

MongoDB *queries* are written in JavaScript. They are in some ways analogous to SQL. The following tutorial has many examples:

https://docs.mongodb.com/v4.4/tutorial/query-documents/

The following reference page gives the different operators that can be used in queries:

https://docs.mongodb.com/v4.4/reference/operator/query/

For this assignment, we recommend you compose your queries in a local file on your computer, and copy-and-paste it into the mongo shell to try them out.

If you find the output of your queries hard to read, you can append `.pretty()` to them to produce indented output.

Submit a file called `queries.js` that contains a query for each of the following questions.

1) [5pts] Retrieve all tweets that are replying to the user with screen name "globeandmail"

2) [5pts] Retrieve all tweets made by the user with screen name "MLHealthUnit"

Submit a file called `aggregations.js` that contains a query for each of the following questions, using the MongoDB aggregation framework.

Aggregations in MongoDB are summaries of a collection. They are similar in concept to the operations performed in a MapReduce. MongoDB aggregations are more restrictive than MapReduce, but their implementation is very efficient. See the following documents for details.

https://docs.mongodb.com/v4.4/core/aggregation-introduction/

https://docs.mongodb.com/manual/reference/operator/aggregation/

3) [10pts] Produce a list of users, together with the total number of times they tweeted, sorted in decreasing order.

4) [10pts] Produce a list of place names, together with the total number of tweets from that place name, sorted in decreasing order.

5) [15pts] Produce a list of users, together with the total number of replies to that user, sorted in decreasing order.

6) [15pts] Produce a list of users, together with the total number of hashtags used by that user, sorted in decreasing order.

Submit a file called `mapreduce.js` that provides a mapper, reducer, and mongodb query to answer the question below.

Because the aggregation model cannot handle all types of summaries, MongoDB also provides a mechanism for MapReduce computations. The syntax is somewhat simpler than python and hadoop; here is an example:

```
function myMapper() {
    //The mapper function is called with each document, which has the
special name 'this'
    //Emit a key-value pair:
    emit(this.user.screen_name, 1);
}
```

```
function myReducer(key, values) {
    //The reducer is called once for each key, and is passed an array
    //containing all values corresponding to that key.
    //Produce the desired result
    return Array.sum( values );
}

db.tweets.mapReduce(myMapper, myReducer, { query: {}, out: "mroutput" })
db.mroutput.aggregate({$sort: {value: -1}})
```

Note that the output of the MapReduce has been placed in a new collection called `mroutput`, which is then queried to get the top answers. (This collection can be given any name.)

7) [40pts] Produce a new collection that contains each hashtag used in the collection of tweets, along with the number of times that hashtag was used.

Hint:

To do something with each object in an array in javascript, we can write:

```
for(obj of arr) {
    //Do something with obj, or obj.field, or whatever...
}
```