Project #3 (MongoDB)

CSC 261/461 (Database Systems), Spring 2017,
University of Rochester
Due Date: 04/12/2017 (11:59 pm)
This IS a group project

Introduction

Last time, we had provided you a fairly large volume of data downloaded (over a decade ago!) from the eBay website stored as XML files. For this project, we are giving you the same data but in JSON format. Like XML, JSON format is primarily used to store semi-structured data.

Your task is to examine the `.json` files, load these files as a `json array` in a MondoDB collection (`database name: ebay; collection name: items`) and later perform a few operations and run a few queries on the data. In this project, we will see how elegantly MondoDB handles 'schemaless' data. Some of the queries you will run for this project are taken from Project 2 Part 1. This is to demonstrate that we can achieve the same results through MongoDB even without storing four different relations as we did for Project 2.

The difficulties of the queries vary but the learning curve would be steep. So, start early. Also, note that this is the first time NoSQL database systems are in the curriculum. So, help from the UTAs might be scarce.

Task A: Working on Bluehive and running MongoDB

CS servers do not host MongoDB services. We will solely use Bluehive server (hosted by CIRC) for this project. You are welcome to do download MondoDB on your own machine, but all the testing would be performed on Bluehive. So, it is your responsibility to make sure your queries work on Bluehive.

We will maintain a separate tutorial (`http://www.cs.rochester.edu/courses/261/spring2017/projects/proj3/mongodb-tutorial.html`) to help you working on Bluehive.

Task B: Examine and load the JSON data files

The JSON files are located in http://www.cs.rochester.edu/courses/261/spring2017/projects/proj3/json.tar. You can use the following command to download the file:

```
wget http://www.cs.rochester.edu/courses/261/spring2017/projects/proj3/json.tar
```

Untar the directory and change directory to 'json' This database is consist of 40 json files: `items-0.json, items-1.json, ..., items-39.json` and a file `load.sh`

There is a total of about 20,000 auctions. Note that the **_id** attribute for any item is unique and involved in only one auction.

Here is a sample of the data:

```
{
    "_id": "1043817906",
    "Name": "NEVADA ghost town bottle digging book, 1961",
    "Category": [
        "Collectibles",
        "Housewares & Kitchenware",
        "Bottles: Antique (Pre-1900)",
        "Medicines, Cures"
    ],
    "Currently": 6.00,
```

```
    "First_Bid": 6.00,
    "Number_of_Bids": 1,
    "Bids": {
        "Bid": {
            "Bidder": {
            "_id": "mimiyaya",
            "_Rating": 10,
            "Location": "Atlanta, Ga",
            "Country": "USA"
        },
        "Time": "Dec-07-01 04:56:27",
        "Amount": 6.00
        }
    },
    "Location": "Iowa",
    "Country": "USA",
    "Started": "Dec-04-01 16:56:27",
    "Ends": "Dec-14-01 16:56:27",
    "Seller": {
        "_id": "tjsdsm",
        "_Rating": 439
    },
    "Description": "GREAT BOOK from the early heyday of bottle digging. "
},
```

Task C: Load your data into MongoDB

You need to load all the JSON files into MongoDB. Each entry in these files will be stored as a document.

For this, run the following commands:

(All the lines starting with # are comments)

Replace XX in 270XX with your 2-digit class ID throughout the document

```
# Check the current directory. You should be in json directory
pwd

# Make a directory mongo.
# We will provide this directory to MongoDB to store all its information.
mkdir mongo

# Give execution permission to load.sh
chmod u+x load.sh

## MongoDB server should run on the compute node server. Starting an interactive
    session
interactive -p standard -t 2:00:00

# loading MongoDB module
module load mongodb/3.3.11

# Start Mongo daemon (mongo is the folder name where you want to store the file)
# Replace XX with your 2 digit classID
# Each student must use different port throughout this project
mongod --dbpath mongo --port 270XX > mongod.log &
```

```
# Run load.sh with the port number you used
./load.sh 270XX

# You should see messages like:
# imported 500 documents ... multiple times
# STOP: if you get any error. Something is wrong. Check again.
```

- You will create `mongo` folder and run `load.sh` file only once
- If you ever need to start from scratch. Delete the database ebay, and run load.sh file again.
- You should always work on a compute node. So, from now onwards, every time you log in, you should go to the JSON directory, and run this three commands (same as you have done before.)

  ```
  interactive -p standard -t 2:00:00
  module load mongodb/3.3.11
  mongod --dbpath mongo --port 270XX > mongod.log &
  ```

- To start mongo client, use the command:

  ```
  # Providing the port number and database name
  mongo --port 270XX ebay
  ```

- Alternatively, if you want to run mongo client and execute commands from a javascript file, use the command:

  ```
  # Providing the port number and database name.
  # Assume commands are stored in 0.js
  mongo --port 270XX ebay < 0.js
  ```

Task D: Test your MongoDB database

The final step is to take your newly loaded database for a test drive by running a few queries over it. As with database creation, first test your queries interactively using the MongoDB command-line client, then store these queries in a file to execute them directly. First, try some simple queries then more complex queries involving aggregation. Make sure the results look correct. When you are confident that everything is correct, write queries for the following specific tasks:

1. Find the **number of seller** in the database.
2. Find the **number of bidders** in the database.
3. Find the **number of items from Canada** (Country)
4. Return the **id and rating** of all the **sellers** whose **rating is higher than 1000** (sorted by rating, id (both ascending)).
5. Find the **number of auctions** belonging to **exactly four categories**
6. Compute the average of all **First_Bid** fields
7. **Find the ID(s) of auction(s)** with the **highest current price**. (tough)

8. **Insert** a new item/auction with _id "88888888" and zero (0) bids. This auction/item must have a name and two categories. (Name and Categories values can be arbitrary)

9. **(UPDATE)** Add a new bid of $1000 to 1678348584. Add 1 to the number of Bids. Other fields in the bid are optional.

10. **Delete** all the auction documents where seller rating is less than zero (0).

We may/may not provide answers to the queries this time. But we highly encourage to discuss the answers with your teammates and also on Piazza.

Put each of the 10 commands in its own command file: 1.js, 2.js, ..., 10.js. Make sure that the naming of your files corresponds to the ordering above, as we will check the correctness using an automated script. We will deduct points for not adhering to the naming conventions.

The content of a sample .js file (0.js) which displays the first document from the collection `items` may look like:

```
db.items.findOne()
```

You can run the file as:

```
mongo --port 270XX ebay < 0.js
```

Submission instructions

To submit the project, first gather all ten (10) `.js` files in the folder `proj3`.

Now, create an archive file (with .tar extension) for submission.

```
cd ..
tar -cvf proj3.tar proj3
ls
```

(Note: `cd ..` takes you to the parent directory and `ls` command should display all the files int the directory including the tar file.)

You will submit your work on betaweb server. That means you need to copy proj3.tar file into betaweb. You can do so using the scp command.

```
scp proj3.tar betaweb.csug.rochester.edu:~/
```

This command will copy the file in your home directory.

Now, Log in to betaweb (probably use a different terminal window) and submit the tar file using the command:

```
/home/csc261/submit proj3.tar
```

You should get a feedback whether the submission is successful or not.

You may resubmit as many times as you like; however, only the latest submission (along with the timestamp) will be saved, and we will use your latest submission for grading your work. Submissions via email will not be accepted! No late submissions allowed.