EECS 484 WN 20 Project #3: MongoDB

Due on Mar. 26th, 2020 by 11:55PM

Overview

In this project, we will use the same dataset as in Project 2 FakeBook and explore the capability of MongoDB (NOSQL). This spec will give you an introduction to MongoDB syntax.

There are two parts to the project. Part A of the project does not use MongoDB. You will be extracting data from tables in the Fakebook database and exporting a JSON file that contains information about Users. In Part B of the project, you will be importing the JSON file `output.json` (or a `sample.json` that we give you) into MongoDB to create a mongo collection called "users" and then you need to write 8 queries on the users collection. Thus, you can start on Part A right away even without knowing anything about MongoDB. Part B requires interacting with MongoDB.

This project is to be done in teams of 2 students or individually. You may work with the same partner as project 1 and/or 2, or you may switch partners.

The autograder is located at autograder.io and you may follow the same instructions to form teams.

Do not make any submissions before joining your team! Once you click on “I’m working alone'', the autograder will not let you change team members. If you do need to make a correction, the teaching staff has the ability to modify teams.

The University of Michigan College of Engineering Honor Code strictly applies to this assignment, and we will be thoroughly checking to ensure that all submissions adhere to the Honor Code guidelines. Students whose submissions are found to be in violation of the Honor Code will be reported directly to the Honor Council. You may not share answers with other students actively enrolled in the course, nor may you consult with students who took the course in previous semesters. You are, however, allowed to discuss general approaches and class concepts with other students, and you are also permitted (and encouraged!) to post questions on Piazza.

1. The Environment

For **Part A**: Because some of our servers, including the Oracle server, are only accessible from the University network, you need to either be on-campus or connected to UM VPN.

For **Part B**: It is a good idea to install MongoDB on your personal computer. Refer to the MongoDB installation document for instructions. Once you have installed it, you should be able to execute 'mongod' (with a mongo with a d) to start a private mongo server. To connect to your private server, you will generally type 'mongo' with the database name in a Terminal window (below, replace <uniqname> with your uniqname. It is being used as the database in the command below, rather than a user ID. So, it could, in principle, be any string you'd like to call your database):

```
$ mongo <uniqname>
```

Notice that the starter code `Makefile` does not work in local environment unless properly modified. And we may be unable to provide support in case you run into issues with the local Mongo environment.

Alternatively, you can use the shared mongodb server that we have set up on the host `eecs484.eecs.umich.edu`. To connect to this server, you will need to be on CAEN.

Type the following command everytime you open a new terminal:

```
$ module load mongodb
```

Then, update the `Makefile` in the starter code with your uniqname and password. A few helpful commands are listed below:

```
$ make loginmongo          # mongo interactive mode
$ make setupsampledb       # loads users collection using sample.json
$ make setupmydb           # loads users collection using output.json
$ make mongotest           # runs test.js against all query*.js
$ make dropdb              # clears users collection in your database
```

https://docs.mongodb.com/manual/tutorial/getting-started/#getting-started contains a very basic mongodb tutorial to get you oriented and is a good starting point to get the basics and become comfortable with mongo database and interacting with it.

2. Files Provided to You

On Canvas you will find "`starter code.zip`" containing files provided to you for this project.

To complete **Part A: Export Oracle database to JSON,** start with the 2 Java files: `Main.java` and `GetData.java`. We have also provided 3 jar packages: `ojdbc6.jar`, `json_simple-1.1.jar`, and `json-20151123.jar`. Put all the above files in the same folder with `Makefile`.

To complete **Part B: MongoDB Queries**, set up your MongoDB database using the provided `Makefile`. Implement MongoDB query in the 8 JavaScript files `query[N].js`. The file `test.js` can be used to check partial correctness of your query results.

To setup or clear MongoDB database for Part B, please substitute <uniqname> and <password> with your MongoDB account information (NOT UM account) in Makefile. **The default MongoDB password is your uniqname**. Please login and change your password following Section 4 Part B instructions. If you are using a private mongod database, then change the Makefile accordingly to omit the hostname, userid, and password information from all mongo commands, or add additional commands in Makefile so that you can test with either private server or the shared server. All the grading is eventually done on the shared server on eecs484.eecs.umich.edu.

3. Part A: Export Oracle database to JSON

1) Introduction to JSON

JSON (JavaScript Object Notation) is a key-value representation, in which values can also be JSON objects. Different from std::map in C++, the values does not have to be consistent in terms of data types. Here is an example of JSON object (initialized in JavaScript):

var student1 = {"Name": "John Doe", "Age": 21, "Major": ["CS", "Math"]}

Name, Age and Major are the keys. Their corresponding values are string, integer and array of strings. An example of calling certain field of value is shown below:

student1["Name"]; // gives John Doe

With multiple JSON objects, we can create a JSON array in JavaScript:

```
var students = [
        {"Name": "John Doe", "Age": 21, "Major": ["CS", "Math"]},
        {"Name": "Richard Roe", "Age": 22, "Major": ["CS"]},
        {"Name": "Joe Public", "Age": 21, "Major": ["CE"]} ];
students [0] [ "Name"];                    // gives John Doe
```

2) Export to JSON

From Project 2 FakeBook Database, you need to use JDBC from a Java program to query USERS, FRIENDS, CITIES and other relevant tables in the Oracle database to export comprehensive information on each user. The results should be a JSONArray users_info, containing 800 JSONObjects for 800 users. It is suggested that you use multiple queries to achieve all the information. Each JSONObject should include:

- user_id
- first_name
- last_name
- gender
- YOB
- MOB
- DOB
- hometown (JSONObject) that contains:
    - city
    - state
    - country
- current (JSONObject) that contains:
    - city
    - state
    - country
- friends (JSONArray) that contains: all of the user_ids of users who are friends with the current user, and has a **larger** user_id than the current user

Below is an example of one element of this JSON array.


```
{
   "MOB":2,
   "current":{
      "country":"Middle Earth",
      "city":"Pelargir",
      "state":"Gondor"
   },
   "hometown":{
      "country":"Middle Earth",
      "city":"Minas Tirith",
      "state":"Gondor"
   },
   "gender":"female",
   "user_id":582,
   "DOB":13,
   "last_name":"JOHNSON",
   "first_name":"Ornella",
   "YOB":41,
   "friends":[
      597,
      598,
      631,
      632,
      645,
      669,
      687,
      714,
      738,
      739,
      742,
      746,
      751,
      768,
      780,
      789
   ]
```


Note: It is possible that a user might have no hometown/current city or list of friends. In this case **put an empty JSONObject**({}) as the value under key "hometown"/"current"/"friends". See `sample.json` for the correct output. More descriptions can be found in **subsection 4)**.

Here are the relevant files to get you started for this part of the project, which you can find in the Starter code that is provided to you.

1) Main.java

This file provides the main function for running Part A. You can use it to run your program, but you don't need to turn it in.

Please only modify the `oracleUserName` and `password` static variables, replacing them with your own **Oracle** username and password.

```
13  public class Main {
14
15      static String dataType = "PUBLIC";
16      static String oracleUserName = "username"; //replace with your Oracle account name
17      static String password = "password"; //replace with your Oracle password
```

2) GetData.java

This file contains the function you need to implement for Part A. Query USERS, FRIENDS, CITIES tables to export data from Oracle Database to JSON format. An output file named `output.json` should be generated in the folder where your Java files are. Your `output.json` is expected to contain the same data as in the provided `sample.json` file, but it can be in entirely different order from `sample.json`.

3) Makefile

To compile, execute `make compile` in the terminal.

To run, execute `make run` in the terminal.

4) sample.json

This file contains the JSON data from running our official implementation of GetData. Please DO NOT `diff output.json sample.json` because JSON arrays are likely to come in entirely different order between any two runs. However, `output.json` and `sample.json` should contain the same elements in the JSON array. There are command line json processors that allow you to diff the contents properly.

Part A and Part B in this project do not have dependencies on each other. You may setup your database for Part B using `sample.json` to test your MongoDB queries. The autograder testing on Part B **does not** rely on correct results from Part A.

If you'd like, you can submit the Java file from Part A to be graded without completing part B. See submission instructions at the end of part B.

4. Part B: MongoDB Queries

1) Introduction to MongoDB

MongoDB is a document-oriented database program. It's comparable to SQL Oracle Databases in many aspects. Each document in MongoDB is one JSON object, with key-value pairs of data, just like a tuple in SQL has fields of data; each collection in MongoDB is one JSON array of multiple JSON objects, just like a table/relation in SQL has multiple tuples. Refer to the following table for concepts of document and collection in MongoDB, as well as queries to select certain columns and rows.

SQL	**MongoDB**
Tuple	Document. JSON object
Relation/Table	Collection. Initialized using a JSON array
SELECT * FROM users;	db.users.find();
SELECT * FROM users WHERE name = 'John' and age = 50;	db.users.find({name: 'John', age: 50});
SELECT user_id, addr FROM users WHERE name = 'John';	db.users.find({name: 'John'}, {user_id: 1, addr: 1, _id: 0});

https://docs.mongodb.com/manual/reference/sql-comparison/ is a document comparing MongoDB with SQL.

2) Log in to MongoDB

To perform the MongoDB queries, you will need to login to Mongo Shell. There are 2 options here, depending on whether you are running the mongo shell on your personal computer or on CAEN and whether you are using a private mongod server or the shared server. Do whatever is convenient and works best for you.

Option 1: Login from your private machine to the private mongo server:

```
$ mongo <uniqname>
```

No hostname, userid, or password is required. <uniqname> is the name of the database that mongo will use for commands that follow.

Option 2: Login from your private machine to the shared mongo server:

```
$ module load mongodb
$ make mongologin
```

Notice you need to change the uniqname and password in `Makefile`.

The default MongoDB password is your uniqname. You can update password with the following command in Mongo Shell:

```
> db.updateUser("<uniqname>", {pwd : "<newpassword>"})
```

The new password takes effect when you logout (Ctrl+D).

3) Import JSON to MongoDB

Open a terminal in the folder where you have `sample.json` (and/or `output.json`) and `Makefile`. Remember to load module each time you open a new terminal to perform any MongoDB operations. Modify `Makefile` with your updated **MongoDB account** information and run `make setupsampledb` in the terminal to load database from `sample.json`, or `make setupmydb` to load database from your `output.json`.

Refer to the `Makefile` for the actual command. Please do not modify the `–collection users` field.

To load data into your private database on the private mongo server:

If you are using a private mongodb installation on your local machine for the project, omit the `--host …--password <password>` portion from the above commands.

On success, you should have imported 800 user documents. **Notice mongoimport command is accumulative, meaning you will have another 800 user objects imported next time you use mongoimport.** To clear up data in the database, use make dropdatabase to drop. See the Makefile contents on what this command does, in case you are using the private server. There's no need to repetitively load and drop database for each query.

4) Locally testing your queries using test.js

In Part B, you will implement 8 queries in the given JavaScript files. The file `test.js` contains one **simple** test on each of the query. You may use it to check **partial correctness** of your implementation. Notice an output saying "Local test passed! Partially correct." does not assure your query1 will get full score on the autograder. Use `make mongotest` to feed the test file into MongoDB, or run the following command on a CAEN terminal (use your mongodb account and password):

```
$ mongo <uniqname> -u <uniqname> -p <password> --host
eecs484.eecs.umich.edu < test.js
```

In the above commands, the first <uniqname> is the name of the database. It can be any string of your choice. The mongodb on the eecs484.eecs.umich.edu uses your uniqname as the name of your database.

Alternative: Again, if you are using a private installation of mongdb on your personal machine, and you have started mongod server as instructed earlier, you can omit username, hostname, and password arguments and connect to your local mongdb server more simply as follows:

```
mongo <uniqname> < test.js
```

(substitute <uniqname> with your private database name)

4) The Eight Queries you need to write

Query 1: Townsmen

In this query, we want to find all users whose hometown city is the specified 'city'. The result is to be returned as a JavaScript array of user_ids, in which the order of user_ids does not matter.

You may find `cursor.forEach()` helpful:
https://docs.mongodb.com/v3.0/reference/method/cursor.forEach/

Query 2: flat_users

In Part A, we have created a `friends` array using JDBC for every user. Each user (JSON object) has `friends` (JSON array) that contains all the user_ids representing friends of the current user who has a larger user_id. In this query, we want to restore the friendship information into friend pairs.

Create a collection called `flat_users`. Documents in the collection follow the schema:
{"user_id": xxx, "friends": xxx}
For example, if we have the following user in the users collection:
{"user_id": 100, "first_name": "John" , ... "friends": [120, 200, 300]}
The query would produce 3 documents (JSON objects) and **store them in the collection** `flat_users`:
{"user_id": 100, "friends": 120},
{"user_id": 100, "friends": 200},
{"user_id": 100, "friends": 300},

You do not need to return anything for this query.
Hint: You may find this link on MongoDB `$unwind` helpful:
https://docs.mongodb.org/manual/reference/operator/aggregation/unwind/
You may use `$project` and `$out` to create the collection, or you may insert tuples into `flat_users` iteratively.

Query 3: Current Cities collection

In this query, we want to create a collection named `cities`. Each document in the collection should contain two fields: `_id` field holding the city name, and `users` field holding an array of user_ids who currently live in that city.

For example, if users 10, 20 and 30 live in Bucklebury, the following document will be in the collection `cities`:

{" _id": "Bucklebury", "users": [10, 20, 30]}

You do not need to return anything for this query.

Query 4: Suggest friends

Find all user_id pairs (A, B) that meet the following requirements:

i. user A is male and user B is female
ii. their Year_Of_Birth difference is less than year_diff, an argument passed in to the query
iii. user A and user B are not friends
iv. user A and user B are from the same hometown city

Your query should return a JSON array of "pairs"; each pair is an array with two user_ids. Essentially it's an array of arrays.

Hint: You may find `cursor.forEach()` useful.

You may use `array.indexOf()` in JavaScript to check for the non-friend constraints.

Query 5: Find the oldest friend

Find the oldest friend for each user who has friends. For simplicity, use only year of birth to determine age. In case of a tie, return the friend with the smallest user_id.

Notice in the `users` collection, each user has only information on friends whose user_id is greater than their user_id. You will need to consider all existing friendships. The idea of Query2 and 3 may be useful.

Your query should return a JSON object: key is the user_id and the value is his/her oldest friend's user_id. The order does not matter. The schema should look like the following:

{ user_id1: user_idx,
user_id2: user_idy, ...}

The number of key-value pairs should be the same as the number of users who have friends.

Query 6: Find average friend count

Find the average number of friends a user has in the `users` collection and return a decimal number. The average friend count on users should also consider those who have 0 friends. In order to make this easier, we're treating the number of friends that a user has as equal to the number of friends in their friend list (we ARE NOT counting users with lower ids, since they aren't in the friend list). DO NOT round the result to an integer.

Query 7: Find count of user born in each months using MapReduce

MapReduce is a powerful parallel data processing paradigm. We have set up the MapReduce calling point in the test.js and you need to implement the mapper, reducer and finalizer.
In this query, we are asking you to use MapReduce to find the number of users born in each month.
Hint: You need to emit a JSON object from your mapper and return a JSON object of **the exact same form** (same keys, same type of values) from your reducer. Since the output of a reducer can be fed into another reducer (a reducer can take input from both mappers and reducers).
Note that after running test.js, running `db.born_each_month.find()` in Mongo Shell allows you to bring up the collection with users born in each month. For example, if there are 200 users born in September, the document below would be in the collection:
{" _id": 9, "value": 200}
You may find the following document helpful: https://docs.mongodb.com/v3.2/core/map-reduce/

Query 8: Find city-average friend count using MapReduce

In this query, we are asking you to use MapReduce to find the average friend count per user where the users belong to the same city. Instead of getting only one number for all users' average friend count, we will have an average friend count for each hometown city.
Hint: You need to emit a JSON object from your mapper and return a JSON object of **the exact same form** from your reducer. **The average calculation should be performed in the finalizer.**
Note that after running test.js, running `db.friend_city_population.find()` in Mongo Shell allows you to bring up the collection with per city average friend count. For example, if Bucklebury has average friend count 15.23, the document below would be in the collection:
{" _id": "Bucklebury", "value": 15.23}

5. Submission and Grading

The autograder is available at https://autograder.io/web/project/609. Before you submit to the autograder, it is best to do some local testing using the provided test.js and Makefile since you have limited submissions per calendar day on the autograder.

To submit on AG, join a team first and submit the following files directly:

1. `GetData.java`

2. `query[1-8].js`

All test cases are graded separately, so you can submit just the files you want to get graded.

Late day policy:

Project 3 is due on Mar. 26th, 2020 at 11:55 pm EST. If you do not turn in your project by this date, or you are unhappy with your work, you may re-submit until Mar. 31st, 11:55 pm (5 days after the due date). You have 4 free late days in the entire semester. Each late day after that incurs a 10% deduction to your project 3 score.