EECS 484 Fall 2019: Project 3
Import and Query MongoDB Database

Due on Nov. 21st, 2019 by 6:00PM

(An up-to-date version of this document can be found at https://drive.google.com/file/d/1DnTROp033AjGx4N65yMZCgh5TbZl72my/view?usp=sharing)

Overview

In this project, we will be using the same dataset as in Project 2 FakeBook and explore the capability of MongoDB (NOSQL). You will first export FakeBook Database to JSON format using JDBC, and then implement nine MongoDB queries in JavaScript. This spec will give you an introduction to MongoDB syntax. There are two parts to the project: Part A and Part B. Part A of the project does not use MongoDB. You will extracting data from tables in a Fakebook database and exporting a JSON file that contains information about Users. In Part B of the project, you will be importing the exported JSON file (or a sample.json that we give you) into MongoDB to create a mongo collection called "Users" and then you will be writing 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.

1. The Environment

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 A. If you get a connection error during Part B, join the VPN and re-try. The autograder machine is now behind the VPN and to submit to it, you will need to be on the UM network.

For Part B of the project, 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. Alternatively, you can use the shared mongod server that we have set up on eecs484.eecs.umich.edu.

To connect to your private server, you will generally type mongo with the database name in a Terminal window (below, replace <uniqname> by 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 like to call your database):

```
$ mongo <uniqname>
```

To connect to the shared mongo server, you will type mongo with arguments that specify the hostname, your uniqname, and your password and the database name.

```
$ mongo <uniquename> -u <uniquename> -p <password> --host
eecs484.eecs.umich.edu
```

In the above, replace <uniquename> with your uniqname and <password> with your mongo password (it is initially set to uniqname as well). Notice that the first <uniqname> is the name of the database. The second <uniqname> is the user ID.

https://docs.mongodb.com/manual/tutorial/getting-started/#getting-started contains a very basic mongo tutorial to get you oriented and is a good starting point to get the basics and becoming comfortable with mongo database and interacting with it. You can try out the commands in the interactive window in the tutorial above or in the mongo shell that is running on your local machine.

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`, `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 8 JavaScript files `query1.js` to `query8.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 `<uniquename>` and `<password>` with your MongoDB account information (NOT UM account) in `Makefile`. **The default MongoDB password is your uniquename**. 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.

To prepare a zip file for submission, execute `make submit` via the terminal. You will be uploading the resulting zip file to the autograder as detailed later in this document.

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
- YEAR_OF_BIRTH
- MONTH_OF_BIRTH
- DATE_OF_BIRTH
- 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 larger user_id than the current user

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

```
{
   "first_name":"Frodo",
   "friends":[
      184,
      214,
      240,
      242,
      244,
      255
   ],
   "hometown":{
      "state":"Gondor",
      "country":"Middle Earth",
      "city":"Edhellond"
   },
   "current":{
      "state":"Gondor",
      "country":"Middle Earth",
      "city":"Edhellond"
   },
   "DOB":28,
   "MOB":12,
   "last_name":"Baggins",
   "gender":"male",
   "YOB":547,
   "user_id":163
}
```


Note: It is possible that a user might have no information regarding his\her hometown or current city in our database. In this case **put an empty JSONObject**({})as value under key “hometown” or “current”. See the file `sample.json` for an example valid output.

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` (usually, it will be in a different order).

3) Makefile

To compile, execute `make` in the terminal. Or you can use the following command:

```
javac -Xlint:-unchecked -cp
"ojdbc6.jar:json-20151123.jar:json_simple-1.1.jar:" Main.java
GetData.java
```

To run, execute `make run` in the terminal.Or you can use the following command:

```
java -cp "ojdbc6.jar:json-20151123.jar:json_simple-1.1.jar:" Main
```

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. But, output.json and sample.json should contain the same elements in the JSON array, though possibly in a different order.

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 MongoDB queries. The AutoGrader testing on Part B **does not** rely on correct results from Part A.

If you 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 3 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 srever. Do what 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:

```
$ mongo <uniquename> -u <uniquename> -p <password> --host
eecs484.eecs.umich.edu
```

The first <uniqname> is the database name.

Option 3: Login from a CAEN Linux machine to the shared mongo server:

The following instructions apply to CAEN environment. **The default MongoDB password is your uniquename.**

```
$ module load mongodb
```

This command allows you to connect to our mongodb server hosted on `eecs484.eecs.umich.edu` via mongo shell in you CAEN computer. If the above does not work, try the following:

$ module load mongodb/3.6.8

Then, you should be able to execute the mongo shell, connecting to the shared server as follows:

```
$ mongo <uniquename> -u <uniquename> -p <password> --host
eecs484.eecs.umich.edu
```

You can update password with the following command:

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

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

3) Import JSON to MongoDB

Open the 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 put `make setupsampledb` in the terminal to load database from `sample.json`, or `make setupmydb` to load database from your `output.json`.

Alternatively, put one of the following commands in the terminal:

```
$ mongoimport --host eecs484.eecs.umich.edu --username <uniquename>
--password <password> --collection users --db <uniquename> --file
sample.json --jsonArray

OR (do the above or below -- either one)

$ mongoimport --host eecs484.eecs.umich.edu --username <uniquename>
--password <password> --collection users --db <uniquename> --file
output.json --jsonArray
```

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 "test1 correct!" does not assure your query1 will score full on the AutoGrader. Use `make mongoquerytest` to feed the test file into MongoDB, or put the following command into 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 <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 <uniquename> < test.js
```

(substitute <uniquename> 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. Each user (JSON object) has `friends` (JSON array) that contains all user_ids, who are friends to the current user and has a larger user_id. In this query, we want to restore the friendship information to 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 keep 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: Hometown Cities collection

Similar to query 1, 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 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 should not 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 a male and user B is a 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 city

Your query should return a JSON array of pairs; each pair is an array with two user_ids.
Hint: You may user `cursor.forEach()` useful.
You may use `array.indexOf()` in JavaScript to check for the non-friend constraints.

```
16 function suggest_friends(year_diff, dbname) {
17     db = db.getSiblingDB(dbname);
18     var pairs = [];
19     // TODO: implement suggest friends
20     // Return an array of arrays.
21     return pairs;
22 }
23
```

Query 5: Find the oldest friend

Find the oldest friend for each user who has friends. For simplicity, user only year of birth to determine age. In case of a tie, return the friend with smallest user_id.
Notice in the `users` collection, each user has only information on friends whose user_id is greater than his/hers. 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 friends user_id. The order does not matter. The schema should look like the following:
{ user_id1: user_idx, user_id2: user_idxx, …}
The number of keys should equal the number of users who have friends.

```
 9 function oldest_friend(dbname){
10   db = db.getSiblingDB(dbname);
11   var results = {};
12   // TODO: implement oldest friends
13   // return an javascript object described above
14   return results
15 }
16
```

Query 6: Find average friend count

Find the average number of friends a user owns in the `users` collection and return a decimal number. The average friend count on users should take into consideration 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 aren't counting users with lower ids, since they aren't in the friend list). DO NOT round the result to an integer.

```
 6 function find_average_friendcount(dbname){
 7   db = db.getSiblingDB(dbname)
 8   // TODO: return a decimal number of average friend count
 9 }
10
```

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

MapReduce is a powerful yet simple 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 from your reducer.

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 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 get 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.

```
6  var city_average_friendcount_mapper = function() {
7    // implement the Map function of average friend count
8  };
9
10 var city_average_friendcount_reducer = function(key, values) {
11   // implement the reduce function of average friend count
12 };
13
14 var city_average_friendcount_finalizer = function(key, reduceVal) {
15   // We've implemented a simple forwarding finalize function. This implementation
16   // is naive: it just forwards the reduceVal to the output collection.
17   // Feel free to change it if needed. However, please keep this unchanged:
18   // the var ret should be the average friend count per user of each city.
19   var ret = reduceVal;
20   return ret;
21 }
22
```

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://grader484.eecs.umich.edu. Before you submit to the autograder, it is best to do some local testing using the provided test.js and Makefile since you only limited submissions per calendar day on the autograder.

To submit on AG, join a team first and then run `make submit` to generate a `project4`.tar.gz**. (Don't worry about the project4 in the name -- that is OK).** The .zip file will bundle the following files for submission:

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

You will get an email when your submission is graded, usually in a few minutes.

Grading policies:

Late day policy:

Project 3 is due on Nov. 21st, 2019 at 6:00 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 Nov. 26th, 6:00 pm (5 days after the due date). Each late day (or part thereof) on which you submit incurs a 10% deduction to your project 3 score.