Project 4: NoSQL – MongoDB

Overview

In the previous project, we used a relational database to store and query a JSON dataset. This is an excellent choice when we need to pose many ad-hoc queries on the dataset, because SQL makes complex queries easy to write and quick to run. But if all we want is to store and retrieve JSON objects, converting back and forth between JSON objects and relations may be an unnecessary overhead with little benefit. In this project, we repeat the same tasks of Project 3 with MongoDB, a popular NoSQL database designed for JSON data, and experience the pros and cons of using a non-relational database.

Development Environment

Project 4 needs a new container with MongoDB. Download and run our new "mongo-apache" container using the following command after replacing `{your_shared_dir}` with your shared directory location:

```
$ docker run -it -v {your_shared_dir}:/home/cs143/shared -p 8889:80 --name mongo-apac
```

The container image has MongoDB, Apache2, PHP, Python, and Node installed and the underlying operating system is Linux. The above command maps the port 8889 on the host to the port 80 of the container, so that the apache server is accessible at http://localhost:8889/. Open a browser and make sure that you can access the apache server without any problem. As before, the default username inside the container is "cs143" with password "password".

You will use the same JSON dataset that you used in Project 3. Please download the data into the `/home/cs143/data/` directory of the container using the `wget` command:

```
$ mkdir -p /home/cs143/data
$ cd ~/data/
$ wget http://oak.cs.ucla.edu/classes/cs143/project3/nobel-laureates.json
```

(In Unix, the tilde character ~ represents the user's home directory, which is `/home/cs143/` in our case.)

After you stop the container, you can restart it using the `docker start` command any time:

```
$ docker start -i mongo-apache
```

Part A: Learn MongoDB

In this part, you get yourself familiar with the basic MongoDB commands by inserting simple "JSON documents" into MongoDB and issuing queries. Start the "MongoDB shell" with the `mongo` command in the container:

```
$ mongo
MongoDB shell version v4.4.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mong
Implicit session: session { "id" : UUID("1a61baf6-76cf-4b10-84de-9103b8cdd8a7") }
MongoDB server version: 4.4.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2021-02-15T17:51:17.583+00:00: Access control is not enabled for the database
        2021-02-15T17:51:17.583+00:00: You are running this process as the root user,
---
>
```

Once in the MongoDB shell, you can run any MongoDB commands. For example, you can insert two JSON documents into the `testcol` collection of the `testdb` database by issuing the following commands:

```
> use testdb;
> db.testcol.insertMany([{ "id": 1, "name": "John" }, { "id": 2, "name": "Elaine" }])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("602775d6e0777393d9d648f2"),
                ObjectId("602775d6e0777393d9d648f3")
        ]
}
```

The first command switches the current database to `testdb`. The second command inserts the two JSON objects as documents in the `testcol` collection of the current database. Once stored, the documents can be retrieved using the `find()` function:

```
> db.testcol.find({ "name": "John" });
{ "_id" : ObjectId("602775d6e0777393d9d648f2"), "id" : 1, "name" : "John" }
```

The above command retrieves all JSON documents in the `testcol` whose `name` attribute has the value `"John"`.

Now that you know how to interact with the MongoDB shell, read the MongoDB tutorial on the Tutorialspoint to learn the basic MongoDB commands. Make sure to read through the Projection Section at least to learn the key commands necessary for this project.

Part B: Load data to MongoDB

As the first task, let us load our JSON dataset to the MongoDB database. More precisely, your task is to insert each `laureate` JSON object in our dataset into the `laureates` collection of the `nobel` database as an individual document. MongoDB supports bulk-loading from a file using the "mongoimport" command-line tool.

For example, assume that you have a file named `test-json.import` with the following content:

```
{ "id": 3, "name": "Julia" }
{ "id": 4,
  "name": "Robert"
}
```

Note that `test-json.import` file contains a list of JSON objects, optionally separated by white spaces or newlines.

You can run the following `mongoimport` command in the ***Unix shell*** to bulk-load the two JSON objects in the file into the `testcol` collection of the `testdb` database:

```
$ mongoimport --drop --db=testdb --collection=testcol test-json.import
2021-02-15T17:53:32.457+0000    connected to: mongodb://localhost/
2021-02-15T17:53:32.459+0000    dropping: testdb.testcol
2021-02-15T17:53:32.473+0000    2 document(s) imported successfully. 0 document(s) fa
```

Notes:

1. `mongoimport` is a command-line tool, not a MongoDB shell command. It must be executed *in the Unix shell, not in the MongoDB shell*.
2. `--drop` option drops the `testcol` collection if it exists before import.

Once loaded, the JSON documents can be queried using the `find()` command inside the MongoDB shell:

```
$ mongo
> use testdb;
> db.testcol.find({"id": 3});
{ "_id" : ObjectId("6027e7e00416cb0ba45f3588"), "id" : 3, "name" : "Julia" }
```

Now, your job is to write a simple program to create a `laureates.import` file ***in the current directory*** with the data in `/home/cs143/data/nobel-laureates.json`. This file will be used as the input file to the `mongoimport` command and must be formatted such that every `laureate` in our

data will be loaded as a separate document. Write your program starting from the template code in the convert.zip, such that the import file can be produced simply by running `./convert.sh`.

The following JSON decoding/encoding functions may help writing your program:

PHP:

- `json_decode($json_string)`: decode a JSON string to a PHP object
- `json_encode($value)`: encode a PHP object to a JSON string

Python:

- `json.loads(json_string)`: decode a JSON string to a Python object
- `json.dumps(object)`: encode a Python object to a JSON string

JavaScript:

- `JSON.parse(json_string)`: decode a JSON string to a JavaScript object
- `JSON.stringify(object)`: encode a JavaScript object to a JSON string

Once you finish writing your program, produce the import file and load it into the `laureates` collection of the `nobel` database using the following commands:

```
$ ./convert.sh
$ mongoimport --drop --db=nobel --collection=laureates laureates.import
2021-02-15T17:54:46.213+0000    connected to: mongodb://localhost/
2021-02-15T17:54:46.214+0000    dropping: nobel.laureates
2021-02-15T17:54:46.506+0000    955 document(s) imported successfully. 0 document(s)
```

Congratulations! You have successfully load the first real data into MongoDB. Open the MongoDB shell and issue a few find() commands to make sure that the data has been loaded as expected.

Note: The type of the "id" field of a laureate object is string. When you look up a laureate based on its id, make sure that its value type is string, not integer. Otherwise, MongoDB may return no matching document.

Part C: Implement Web-service JSON API

Your second task is to implement a "Web service" that takes a Nobel laureate's ID and returns the JSON data of the laureate. The Web service should be accessible at `http://localhost:8889/laureate.php?id=XXX` and return the corresponding JSON data.

To learn how to access a MongoDB database from PHP, read our tutorial on PHP MongoDB Driver. Once you finish learning the PHP-MongoDB Driver API, start from the "skeleton PHP code" laureate.php and modify it to return the correct JSON data for the input ID. To check if the JSON

data returned from your PHP is in a valid format, you can use one of many online JSON validators, such as JSONLint.

Notes:

1. Again, keep in mind that the `id` attribute of our JSON data is of `string` type (e.g., `"745"`), not an integer type (e.g., `745`). When you "query" MongoDB for a document based on `id`, the provided id value should be of `string` type as well.
2. Make sure that the returned document is as close as possible to the original data. This means that it must include all attributes in the original dataset and ***does not include any extra attribute that did not exist, such as the `_id` attribute.***

Part D: Write aggregate queries

Your final task is to write the following "MongoDB queries":

1. What is the id of "Marie Curie"?
   Answer: `{ "id" : "6" }`
2. What country is the affiliation "CERN" located in?
   Answer: `{ "country" : "Switzerland" }`
3. Find the family names associated with five or more Nobel prizes?
   Answer: `{ "familyName" : "Wilson" } { "familyName" : "Smith" }`
4. How many different locations do the affiliation "University of California" have?
   Assume that a location is uniquely identified by its city and country. **Answer:** `{ "locations" : 6 }`
5. In how many years a Nobel prize was awarded to an organization (as opposed to a person) in at least one category?
   Answer: `{ "years" : 26 }`

Please make sure that the results from your queries have ***the same attribute names and values***.

Possibly except the first one, your MongoDB queries should be written using the `aggregate()` function of MongoDB. Read this excellent tutorial on MongoDB aggregate framework to learn how to use `aggregate()` functions to write complex queries.

Among many aggregation stages and operators that MongoDB supports, you may find the following particularly useful for writing the above queries. Read the relevant sections of MongoDB references on aggregates to learn about them:

- Aggregation stages: `$match`, `$group`, `$project`, `$limit`, `$count`, `$addFields`, `$unwind`
- Aggregation operators: `$and`, `$or`, `$not`, `$concat`, `$sum`, `$elemMatch`, `$gte` (and other comparison operators), `$exists`

Once you finish writing and debugging the queries, write five MongoDB shell script files, named as `q1.js`, ..., `q5.js`, that contain each of the above MongoDB aggregate functions. For example, we should be able to run `q1.js` and get its output like the following:

```
$ mongo nobel --quiet < q1.js
{ "id" : "6" }
```

Part E: Reflecting on your experience

Now that you have completed the same task twice, once with MySQL and once with MongoDB, take a moment to reflect on your experience. Which system was easier when you tried to perform each of our tasks? How was transformation and loading? How was the Web service that returns JSON objects? How were the analytic queries? Do you feel that one system was better than the other in all tasks? If not, which system will you use for what task in the future?

Submit Your Project

What to Submit

For this project, you will have to submit one zip file:

- `project4.zip`: You need to create this zip file using the packaging script provided below. This file will contain all source codes that you wrote for Project 4

Creating `project4.zip`

Roughly, the zip file that you submit should have the following structure:

```
project4.zip
 +- convert.sh
 +- convert.js/py/php or any other files for conversion
 +- laureate.php
 +- q1.js
 +- q2.js
 +- q3.js
 +- q4.js
 +- q5.js
 +- README.txt (Optional)
```

Make sure that your code meets the following requirements:

1. Your data conversion program should take the JSON data located at `/home/cs143/data/nobel-laureates.json`.

2. Your data conversion program should produce the import file as `laureates.import` in the current directory.
3. We should be able to execute your conversion program simply by running `convert.sh`. Nothing else.
4. Your `laureate.php` file must work if we copy it to the `/home/cs143/www/` directory.
5. All you queries must be executable by `mongo nobel --quiet < q?.js`
6. The README.txt file may be optionally included if you want to include any information regarding your submission.

To help you package your submission zip file, you can download and use our packaging script p4_package. After downloading the script in your Project 4 directory, remove all files that should not be submitted from the directory and execute the packaging script.

When executed, our packaging script will collect all files located in the same directory as the script and create the `project4.zip` file according to our specification like the following:

```
$ ./p4_package
  adding: convert.js (deflated 35%)
  adding: convert.sh (deflated 27%)
  adding: laureate.php (deflated 40%)
  adding: q1.js (deflated 16%)
  adding: q2.js (deflated 41%)
  adding: q3.js (deflated 34%)
  adding: q4.js (deflated 51%)
  adding: q5.js (deflated 20%)
[SUCCESS] Created '/home/cs143/project4/project4.zip'
```

Testing of Your Submission

In order to minimize any surprises, we are providing a simplified version of the "grading script" p4_test that we will use to test the functionality and correctness of your submission. In essence, the grading script unzips your submission to a temporary directory and executes your files to test whether they run OK on the grader's environment. Download the grading script and execute it in the container like:

```
$ ./p4_test project4.zip
Running the conversion program via convert.sh...
Loading data from the generated import file...
2021-02-15T17:54:46.213+0000    connected to: mongodb://localhost/
2021-02-15T17:54:46.214+0000    dropping: nobel.laureates
2021-02-15T17:54:46.506+0000    955 document(s) imported successfully. 0 document(s)

Running your query script q1.js...
{ "id" : "6" }

Running your query script q2.js...
{ "country" : "Switzerland" }

Running your query script q3.js...
{ "familyName" : "Wilson" }
{ "familyName" : "Smith" }

Running your query script q4.js...
{ "locations" : 6 }

Running your query script q5.js...
{ "years" : 26 }

Copying your laureate.php file to ~/www/
All done!

Please ensure that you have the Web service available at http://localhost:8889/laurea
(1) Please make sure that the returned JSON format is valid
(2) Please make sure that the return JSON structure is the same as the original and d
```

You **MUST** run your submission using the script before your final submission and ensure that it runs fine and produces expected results.

Submitting Your Zip File

Visit GradeScope to submit your zip file electronically by the deadline. In order to accommodate the last minute snafu during submission, you will have 1-hour window after the deadline to finish your submission process. That is, as long as you start your submission before the deadline and complete within 1 hour after the deadline, you are OK.