Big Data Management and Analytics
Session: MongoDB

Lecturer: Pedro González and Jovan Varga

January 27th, 2017

In this session we will install a MongoDB instance in our cluster. In-class we will perform all exercises locally and in the optional part you can install it in distributed mode.

It is highly advisable to complete the tasks listed in section 2 prior to attending the class, in order to get full advantage of the available in-class time.

1 Required Tools

- `eclipse` IDE with Java 7 and Maven plugin installed
- The following Java libraries (already included in the provided `pom.xml`):
  - `MongoDB`
  - `JFairy`

2 Tasks To Do Before The Session

- Read the slides and understand the slides.
- Checkout the MongoDB manual
- Checkout the MongoDB Java manual

3 Part A: Examples & Questions (30min)

During the first 30 minutes of session we will go through the proposed examples in the slides. After each one, questions related to that particular topic will be addressed.

4 Part B: In-class Practice (2h30min)

4.1 Exercise 1 (1h30 min): Setting up and shell commands

In the `tarballs` folder in your instance you have already available an instance of MongoDB. In order to install it, issue the following commands:

```
cd ~/tarballs
tar xf mongodb-linux-x86_64-ubuntu1204-3.0.6.tgz
mv mongodb-linux-x86_64-ubuntu1204-3.0.6 ..
cd ~/data
mkdir db
```

Start the `mongod` server!

```
mongodb-linux-x86_64-ubuntu1204-3.0.6/bin/mongod --dbpath data/db/ &
```

In this first exercise we will import a JSON sample dataset using the `mongoimport` command. Particularly, this dataset depicts restaurants and a sample document looks like the following:

```
{
  "address": {
     "building": "1007",
     "coord": [-73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": {"$date": 1393804800000},"grade": "A", "score": 2},
     { "date": {"$date": 1378857600000},"grade": "A", "score": 6},
     { "date": {"$date": 1358985600000},"grade": "A", "score": 10},
     { "date": {"$date": 1322006400000},"grade": "A", "score": 9},
     { "date": {"$date": 1299715200000},"grade": "B", "score": 14}
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

In order to populate the `test` database, you have to download the dataset and execute the `mongoimport` to insert the documents into the restaurants collection in the test database. If the collection already exists in the test database, the operation will drop the restaurants collection first.

```
wget https://raw.githubusercontent.com/mongodb/docs-assets/primer-
    dataset/primer-dataset.json
mongodb-linux-x86_64-ubuntu1204-3.0.6/bin/mongoimport --db test --
    collection restaurants --drop --file primer-dataset.json
```

The `mongoimport` connects to a `mongod` instance running on localhost on port number 27017. To import data into a `mongod` instance running on a different host or port, specify the hostname or port by including the –host and the –port options in your `mongoimport` command.

Now, connect to the `mongod` instance using the `mongo` client.

```
mongodb-linux-x86_64-ubuntu1204-3.0.6/bin/mongo
```

If you imported the data successfully, the `show dbs` command will show the newly created `test` database. After selecting it, `use test`, the restaurants collection should be available, `show collections`, and you should be able to explore it via `db.restaurants.findOne()`.

Now, in this exercise we ask you to issue the following queries in the shell, some hints:

- The MongoDB manual is the best reference for syntax and operation details: `https://docs.mongodb.org/manual/`.
- Some queries may require subqueries (not available in MongoDB), however you can store intermediate results using variables.Note the `find` method returns a cursor, and thus to use a variable in a subresult you should use the `findOne` method which already accesses the cursor.
- To perform aggregations you may use the Aggregation Framework or MapReduce. Read more in `https://docs.mongodb.org/manual/core/aggregation-introduction/`.
- Geospatial queries require indexes, read more about geospatial indexes in `https://docs.mongodb.org/manual/administration/indexes-geo/`.
- If you get many results you can use the `findOne` method to improve readability.

Queries:

Q1: Restaurants whose borough is “Manhattan”.

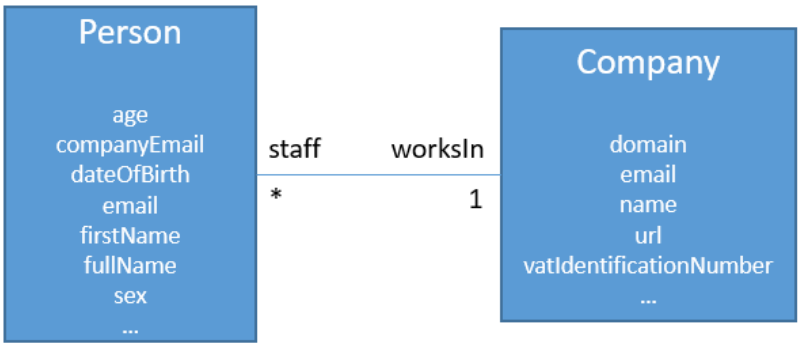
Q2: Number of restaurants in “Manhattan” with some grade scored greater than 10.

Q3: Average score by cuisine type

Q4: Restaurants close (150 meters or less) to the restaurant “The Assembly Bar”.

4.2 Exercise 2 (1h30 min): Modeling and the Java API

In this exercise we will explore the different modeling alternatives in MongoDB. Precisely, we will work with the following conceptual model depicted in UML.



The information requirements, i.e. queries, for this exercise are the following:

Q1: For each person, its name and its company name.

Q2: For each company, the name and the number of employees.

Q3: Average age per company.

In this exercise you are asked to design the database using different modeling patterns, precisely:

1. Two collections, one for each class and reference fields.
2. One collection with nested documents.
3. Hybrid, separating each class into each collection but materializing required aggregations.

Precisely you are asked, for each design pattern, to **implement in Java** the following tasks:

1. Using `JFairy` (a random Java data generator), generate random data persons and companies.
2. Insert the data into MongoDB with the specific model.
3. Program queries Q1, Q2 and Q3 to write its results in the console.

To aid you in doing the exercise, in the provided Java project you can find sample code for generating and inserting data into MongoDB. Precisely, in class `Exercise_2_example.java`. For a full reference on MongoDB Java API you can check `http://mongodb.github.io/mongo-java-driver/3.2/driver/reference/`.

5 Part C: Optional Practice (3h)

5.1 Exercise 3: Querying Barcelona

In this optional exercise you are proposed to import the dataset for Barcelona from OpenStreetMap. There is already a package in JSON format that you can download from `https://s3.amazonaws.com/metro-extracts.mapzen.com/barcelona_spain.imposm-geojson.zip`. It contains several files for places like neighborhoods, buildings, roads, etc.

The first task you must do is to load such files into MongoDB. After, we ask you to implement some complex geospatial queries. Although you can implement your own proposals, our suggestions are:

Q1: For each neighborhood (or town), the number of buildings that are within its coordinates.

Q2: Streets that span along more than 2 neighborhoods.

Q3: Ranking of most common amenities in "Badalona"

You are open to implement them using Java or the shell. Just make sure to submit your solution with clear explanations on how you obtain the results, which collections you use and which indexes (if any) you created.