Assignment: Practice Using MongoDB

Avraham Leff

COM3580: Spring 2024

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 3580" on Piazza).

This assignment's "DIR" **must be named** *Mongo1*. Your report **must be named** *$DIR/Mongo1.pdf*.

2 Motivation

The purpose of the assignment is for you to get experience installing MongoDB, designing and populating MongoDB collections, and formulating queries against the MongoDB database.

Per lecture and the MongoDB documentation there are two ways to query MongoDB data.

> The MongoDB Query API is the mechanism that you use to interact with your data. The Query API comprises two ways to query data in MongoDB: C/R/U/D Operations and aggregation pipelines

You will use both mechanisms in this assignment, be sure to use the one that I specify.

In addition, I hope that the assignment will deepen your understanding of the differences (including advantages and disadvantages) between the relational and document database models.

3 Approach

In the first part of the assignment, you will populate a MongoDB database with data derived from a relational database. That data is not in a form suitable "as is" for MongoDB. I'm not only referring to *syntactic* differences: I'm also referring to the fact that the relational *data model* differs from the MongoDB document model.

You can find the "source" relational data here. Download the SQL population script, examine the DDL, and "port" the SQL data into an equivalent MongoDB database.

You may find that the transition from relational to MongoDB data is eased if you instantiate the relational database and experiment with queries against the RDB. Alternatively: jump right into the MongoDB population.

Each document **must have the following structure**:

```
{
  "_id" : ObjectId("59064fa0a3e67042eadf759a"),
  "name" : "WorkerName",
  "street" : "Liebig",
  "city" : "Hudson Park",
  "manages" : [ ObjectId("59064fa0a3e67042eadf75a1") ],
  "companies" : [
    { "name" : "Apple", "city" : "Cupertino", "salary" :
        126672.27 }
  ]
}
```

Semantics:

- Key point: the "real world" model (which must be reflected in both the SQL and MongoDB renderings) is that "working" and "managing" relationships are *many-to-many*.
- People can work at zero or more companies and can have zero or more managers.
- Workers are remote consultants who can work for multiple companies at the same time, and can live in a city that differs from the company's city.

- Companies can have branches in multiple locations.

4 Requirements

> Note: in all such exercises, if necessary, you are responsible for (re)initializing the database before executing your MongoDB commands. All subsequent interactions with the database must execute the specified **in the specified order**!

4.1 Step 0

Include this screenshot into your writeup:

1. `mongosh -version`
    - I get `2.1.3`
2. Once in the shell, `db.version()`
    - I get `7.0.2`

You don't have to be using the exact versions (above), but the "major" versions should match.

4.2 Step 1

Populate the MongoDB collection per the above requirements.

4.3 Step 2

- For each of the following *natural language* queries, you must
    1. Show me a formatted MongoDB "translation" of that query.

2. Show a **clear screenshot** of the query execution and the results therefrom. All queries **must** use format their output via the `pretty` directive!
   Example: `db.collection.find({}).pretty();`

> Important: by default include all of a document's properties. Otherwise, include **only** the document properties that I specify.
>
> All queries except the first two **must omit** the `_id` property.
>
> I will deduct points if you fail to follow the specified format.

- Note: your grade depends only on "correctness": i.e., the formatted result of your query!

> Queries 1-4 **must be** implemented with the MongoDB C/R/U/D API; subsequent queries **must be** implemented with the *aggregation* APIs.

1. *Return the document with "worker name" Kimiko.*
2. *Return the document with "worker name" Luiza.*
3. *Return all workers living either in Shengzhou or Fresno.* Each document in the result must include only the worker name, street, and city properties. The result must be sorted by increasing "name".
4. *Return all workers that don't work for Deloitte.* The result-set must be sorted by increasing worker name, and display a maximum of four documents.
5. *Return the set of workers that directly manage other workers.* Each document in the result set must contain a name and direct_reports property, the latter being the number of workers that are directly managed by name.
6. *Return all cities with companies located in that city, along with the average salary for people working in that city.*

The salary should be rounded to two digits of precision after the decimal point.

Each document in the result-set should have these properties: `_id` (the name of the city) and `rounded_avg_salary`.

7. *Which company location has the most people working for it? How many people work for it?*

   The result-set should be an array containing a single document: the document has an `_id` property containing the name of the company along with a `num_people` property.

8. *Which company has the most people working for it across all of its locations? How many people work for it?*

   The result-set should be an array containing a single document: the `_id` of that document should be an object containing `company_name` and `company_city` properties. In addition, the top-level document should contain a `num_people` property.

9. *Return an array containing a document for each company. Each document specifies the company name in its `_id` property. The document also contains `payroll` (the sum of worker salaries), `num_people` (number of people working for it across all locations), and `min_salary`, `max_salary`, `avg_salary` properties.*

   The result set should be sorted in descending payroll values.