Non Relational Databases

Introduction

This assignment will give us a chance to become familiar with NoSQL databases in general, and MongoDB in particular. We will install MongoDB on our local machines and then learn some of the more common database management and data manipulation commands.

Installing MongoDB

Let's install a MongoDB server on our computer and then running it as local standalone database server. Download MongoDB Community Server[1] for your operating system.

Installing MongoDB on Windows

To install MongoDB Community Server[1] on Windows, download the ***msi*** Windows Installer file[2], run the installer following the instructions and accept all terms and conditions. Install the ***Complete*** version including all features and choose to run Mongo as a service so that it will start automatically when your computer boots up. Restart your computer when the installation is done and confirm the database installed under ***Program Files***, e.g., ***C:\Program Files\MongoDB\Server\4.4***. Under the ***mongodb*** directory there's a ***bin*** directory containing several executables including the server, ***mongod***, and the command line mongo client, ***mongo***. Add the ***bin*** directory to the system's ***PATH*** so that *mongod* and *mongo* can be executed from the terminal.

Installing MongoDB on MacOS

To install MongoDB Community Server[1] on macOS, download the compressed ***tgz*** file[2]. Using macOS Finder[3], navigate to the ***Downloads*** directory, double click the downloaded ***tgz*** file to decompress it, and rename the resulting directory as ***mongodb***. The mongodb directory can live anywhere in the filesystem, but a good place to put it is in ***/usr/local***. You can navigate to the ***/usr/local*** directory from macOS's ***Finder*** by pressing ***Command+Shift+G***, and then typing ***/usr/local*** in the ***Go to the folder*** dialog, and then clicking ***Go***. You might need to provide your password, or authenticate when trying to navigate to ***/usr/local***. Using the Finder, move the ***mongodb*** directory into ***/usr/local***. Under the ***mongodb*** directory there's a ***bin*** directory containing several executables including the server, ***mongod***, and the command line mongo client, ***mongo***. Add the ***bin*** directory to the system's PATH so that *mongod* and *mongo* can be executed from the terminal. To add ***mongodb/bin*** to the PATH, edit your ***~/.bash_profile*** in any editor and add the following line at the end of the file. Note the period at the beginning of the file, i.e., it's a hidden file.

~/.bash_profile

```
export PATH="$PATH:/usr/local/mongodb/bin/"
```

[1] https://docs.mongodb.com/manual/administration/install-community/
[2] https://www.mongodb.com/try/download/community
[3] https://support.apple.com/en-us/HT201732

Testing the MongoDB installation

Start the MongoDB Server

After restarting your machine, using your file explorer or finder, create a directory on your file system called ***mongo-assignment***. Under the new directory, create another directory called ***data***. The ***data*** folder is where your databases will be stored. Start a terminal or console in your OS, and navigate to your new ***mongo-assignment*** directory. From your console and in your ***mongo-assignment*** directory, start the ***MongoDB server*** telling it to use our ***data*** folder with the ***dbpath*** option as shown below. NOTE: that's ***double dash*** in front of ***dbpath***.

```
mongod  --dbpath=data
```

If the OS gives you a warning about running ***mongod***, give the application all necessary permissions. Do not uninstall the application. On ***macOS*** you might need to go to ***System Preferences***, ***Security and Privacy***, ***General*** tab, ***Allow apps downloaded from***, click on ***Allow Anyway***.

If the server had failed to start, try again. The server should start and listen at port 27017 for connections from database clients or applications. Leave this console or terminal alone for now. If you close it, the server will shutdown. Everytime you want to work with the server, you will have to restart it as described in the steps above.

Start the MongoDB Client

After successfully starting the server from the terminal, start a separate terminal or console to start the mongo client. In the console, type mongo and enter/return at the command prompt. The mongo client should start, connect to the server, and give you a prompt for you to interact with the database server. The client will keep a log of all the commands you type under ***~/.dbshell*** (under your home/profile directory). As you interact with the client, all your commands will be logged there. At the end of this assignment, submit this file as part of your deliverables. You will also export the content of your database and submit that as well. The shell commands, together with the database export, will allow TAs to confirm your work.

Create Database

Now that the client and server are up and running, let's practice some simple database management commands to get our feet wet. Let's first list all the databases. Type the following at the command prompt in the database client.

```
show dbs
```

Confirm that the following databases display. Your actual list might differ somewhat.

```
admin   0.000GB
config  0.000GB
local   0.000GB
```

Now use the ***use*** command to create a database called ***db_design*** as shown below

```
use db_design
```

Confirm that the command was acknowledged with the following response.

```
switched to db db_design
```

List all the databases again and note that the database you just created is NOT listed. It wont show until you actually insert data in the database as we'll confirm in a later step.

Inserting Data into MongoDB

Now that we have a database to work with, let's insert some data into the database. We'll insert a couple of users into a collection called ***users***. You can list all the collections in a database by using the show collections command as shown below

```
show collections
```

confirm that there aren't any collections yet since you have not yet inserted any data. Once you insert data you should be able to see the collection.

Inserting users

All data manipulation commands have the following structure

```
db.<collection>.<command>(<data>/<parameters>)
```

Where the parameters are JSON objects that contain data and/or parameters particular to the command. Let's use the ***insert*** command to insert ***data*** into the ***users*** collection as shown below.

```
db.users.insert({firstName: "Thomas", lastName: "Sowell"})
```

Now that we've inserted data into a collection, let's confirm that the show collections command now does in deed show the new collection.

```
show collections
```

Confirm that now it shows ***users*** collection. Now that we've created at least one collection, list all the databases

```
show dbs
```

Confirm that now the ***db_design*** database shows.

```
admin      0.000GB
config     0.000GB
db_design  0.000GB
local      0.000GB
```

Let's insert two more users as shown below

```
db.users.insert({firstName: "Cathie", lastName: "Wood"})
db.users.insert({firstName: "Adam", lastName: "Smith"})
```

Let's confirm that the users were indeed inserted by using the find command on the users collection as shown below.

```
db.users.find()
```

Confirm that all the documents in the users collection are listed as shown below. Your values for the _id field will be different.

```
{ "_id" : ObjectId("606111fd09c8d49beb1aa3be"), "firstName" : "Thomas", "lastName" : "Sowell" }
{ "_id" : ObjectId("6061127009c8d49beb1aa3bf"), "firstName" : "Cathie", "lastName" : "Wood" }
{ "_id" : ObjectId("606112a809c8d49beb1aa3c0"), "firstName" : "Adam", "lastName" : "Smith" }
```

Inserting blogs related to users

Now that we have a users collection, let's create another collection containing blog documents related to the user documents. The user documents will be the authors of the blog documents. Let's create a blog document with ***Thomas Sowell*** as the author. Note that the value of the ***author*** field below is the ***_id*** for ***Thomas Sowell*** in the ***users*** collection. This value will be different for you. Use your actual values instead of the ones shown below. If you use my values, the assignment won't work. Also note that the document has a ***revenue*** field we'll be using later.

```
db.blogs.insert({name: "Basic Economics", topic: "ECONOMICS", author: "606111fd09c8d49beb1aa3be",
revenue: 1234567})
```

Create another blog document with ***Cathie Wood*** as the ***author*** as shown below. Use your actual value for the ***author*** field below. ***DO NOT*** use my author value otherwise the assignment will break.

```
db.blogs.insert({name: "Ark Invest", topic: "INVESTMENT", author: "6061127009c8d49beb1aa3bf", revenue:
2345678})
```

Finally create another blog document with ***Adam Smith*** as the ***author***. Again, use your author value

```
db.blogs.insert({name: "Wealth of Nations", topic: "ECONOMICS", author: "606112a809c8d49beb1aa3c0",
revenue: 3456789})
```

Use the find command to confirm that the blogs were created as shown below

```
db.blogs.find()
```

Here's a sample of the data you should see. Your values of ***_id*** and ***author*** will be different.

```
{ "_id" : ObjectId("606115f509c8d49beb1aa3c1"), "name" : "Basic Economics", "topic" : "ECONOMICS",
"author" : "606111fd09c8d49beb1aa3be", revenue: 1234567 }
{ "_id" : ObjectId("6061260209c8d49beb1aa3c2"), "name" : "Ark Invest", "topic" : "INVESTMENT", "author" :
"6061127009c8d49beb1aa3bf", revenue: 2345678 }
{ "_id" : ObjectId("6061260a09c8d49beb1aa3c3"), "name" : "Wealth of Nations", "topic" : "ECONOMICS",
"author" : "606112a809c8d49beb1aa3c0", revenue: 3456789 }
```

Inserting posts

In the previous sections we have shown you how to create collections and then populate them with documents stored in collections. ***Now you try it***. Create a collection called ***posts*** that contains several posts for each of the blogs created earlier. Create ***two*** posts for each of the blog. Find articles on economics and investment and insert posts with the structure shown below. Two of the posts should have less than ***5000*** likes, two of the posts should have likes between ***5000*** and ***15000***, and two posts should have more than ***15000*** likes. In the end you should have at least 6 posts. Feel free to create more if you like.

```
{
   title: "The title of the post",
   body: "A paragraph from the article",
   likes: integer value number of likes,
   blog: "THE _id OF THE BLOG"
}
```

Querying Data

Now that we have data in our database, let's practice retrieving the data from the database. We'll practice retrieving all the documents from a collection, retrieving a particular document by their primary key, and then using a predicate.

Retrieving all documents

As shown in previous steps, we've been using the find command to retrieve all the documents of a collection. To retrieve all the blogs from the blogs table, we used the following command.

```
db.blogs.find()
```

Confirm that you can see all the blog documents.

Retrieving a specific document

To retrieve a specific blog by their primary key, we can use the following query. Instead of using the ID below, use an actual ID from a real blog document inserted in earlier steps.

```
db.blogs.find({_id: ObjectId("606115f509c8d49beb1aa3c1")})
```

Confirm that the correct blog document is displayed

Retrieving documents by criteria

Let's now retrieve all blogs whose topic is "ECONOMICS" as shown below.

```
db.blogs.find({topic: "ECONOMICS"})
```

Confirm that all the ECONOMICS blogs are listed. Now let's retrieve blogs that have a revenue between 1500000 and 2500000 as shown below.

```
db.blogs.find({$and: [ {revenue: {$gt: 1500000}}, {revenue: {$lt: 2500000}} ]})
```

Confirm that the correct blogs are listed.

Retrieving child documents

Documents in the blogs collection are related to documents in the users collection through their author property. Retrieve the blog whose author is Cathie Wood. In my case the ID for Cathie Wood is 6061127009c8d49beb1aa3bf. In your case it will be a different ID. Use your ID for the query.

```
db.blogs.find({author: "6061127009c8d49beb1aa3bf"})
```

Confirm that the correct blog is displayed.

Querying Posts

In the last few sections we have demonstrated various commands to query the blogs collection. ***Now you try it***. Query the posts collection you created in an earlier section. Write the following queries

- Retrieve all documents from the posts collection
- Retrieve a specific post document by their ID
- Retrieve all posts related to a specific blog
- Retrieve all posts with less than 15000 likes
- Retrieve all posts with more than 15000 likes

- Retrieve all posts with likes between 10000 and 50000

Updating Data

Let's now update some documents already existing in the database. Let's change the revenue of one of the blogs to represent its success. Set the revenue of Cathie Wood's blog to 4325436. The ID below is for Cathie Wood's blog in my database. For you it will be a different ID. Use your ID.

```
db.blogs.update({_id: ObjectId("6061260209c8d49beb1aa3c2")}, {$set: {revenue: 4325436}})
```

Now let's favorite all the ECONOMICS blogs:

```
db.blogs.update({topic: "ECONOMICS"}, {$set: {favorite: true}})
```

List all the blogs and confirm that your updates were successful.

Now you try it. Update your posts as follows:

- Choose the most liked post document and double its likes. You'll have to calculate this by hand and hardcode the new value in your command
- Now choose the two least popular posts and add a favorite: false property for both posts

Deleting Data

Finally let's play with removing documents from the database. Let's remove a specific user by their ID. Let's remove Adam Smith. In my database he's ID is 606112a809c8d49beb1aa3c0. Yours will be different. Use your ID.

```
db.users.remove({_id: ObjectId("606112a809c8d49beb1aa3c0")})
```

List all the users and confirm that Adam Smith is no longer in the database

Now let's remove blogs with revenues less than 2000000.

```
db.blogs.remove({revenue: {$lt: 2000000}})
```

List all the blogs and confirm that the "Basic Economics" blog is no longer there

Now you try it. Remove the following posts from the database:

- Pick a post document and remove it by its ID. Confirm that the post was removed.
- Remove all posts less than 10000 likes. If you don't have any posts with less than 10000 likes, then choose another threshold that would remove at least one post. Confirm that the post(s) were removed.

Export the Database

The MongoDB Database Tools is a set of tools for exporting and importing MongoDB databases. Download the tools as a zip file, decompress the file, move the files in the ***bin*** directory to the ***bin*** directory you already added to the ***PATH*** in a previous step. This way the tools will also be available in the terminal. For the tools to work, the database must be running. In a separate terminal window, go to the assignment folder named ***mongodb-assignment***. Run the ***mongodump*** tool to export the content of the ***db_design*** database you have been working with as follows. Note, that's a double dash followed by ***db=db_design***.

```
mongodump  --db=db_design
```

The command will create a directory called ***dump*** containing the data in your database. Zip up the directory and submit it as a deliverable.

Deliverables

Submit the following files to Canvas or Blackboard:

- The data ***dump*** created by mongodump. Zip the directory and upload it to Canvas
- The ***.dbshell*** document containing the log of your interaction with the mongo client. The file is called ***.dbshell*** and it's located in your home directory. Note that the file has a period at the beginning so it might be a hidden file. You might need to configure your file explorer of finder to show hidden files.