Intro to Big Data – Lab 4 (MongoDB)

What we will cover in this lab:

- **Getting familiar with MongoDB**
- **Basic MongoDB security (enabling user authentication on the database server)**
- **Basic data manipulation, basic queries**

Useful links:

https://docs.mongodb.com/manual/

Preamble:

MongoDB is a NoSQL document database management system. It stores data in flexible, JSON-like documents (the actual format is BSON or binary JSON). You can have many databases on a MongoDB server instance. Every database contains collections of documents. A collection is similar to a relational table, but it has a flexible structure. This means that the structure of the documents stored in a collection may differ. You don't have to define a collection's schema when you create the collection, like you do when you create a relational table. A document is similar to a row from a relational table, but it has a rich structure (a field can store complex structures like an embedded document, an array of values or an array of embedded documents). Each document has an immutable field called "_id" which acts as a unique identifier (or primary key) at the collection level. If a value for this field is not specified, MongoDB will generate a value of type ObjectId when the document is inserted. MongoDB has a very expressive query language based on JavaScript. MongoDB offers high availability and horizontal scalability when it runs on a cluster of commodity servers. Also, MongoDB is case sensitive.

Your tasks:

Start the MongoDB Database Server specifying the --auth flag (this means authentication is enabled):

```
mongod --auth
```

Start the MongoDB Client or mongo shell:

```
mongo
```

You are by default connected to a database called 'test'. If you want to see the name of the current database, just type 'db' in mongo shell:

```
> db
```

Since by default MongoDB doesn't give you any database users, you must create them yourself. At this moment, your database server doesn't have any users, so there is no way to authenticate yourself to the server, nor can you be authorized to do any work. So, you have to use the localhost exception to connect to the database server (this means that you must connect to the mongo shell from the same host that is running the database server) in order to create the first database user. After you create the first user, the localhost exception closes, and you won’t be able to create other users, so make sure you create a user with an administrative role, like ‘root’. ‘Root’ is a built-in role that belongs to the ‘Superuser Roles’ category. To create your first database user, you have to switch to the ‘admin’ database:

```
> use admin
```

After that, you will create the first user on this database server:

```
> db.createUser({
  user: "superuser",
  pwd: "superpass",
  roles : [ "root" ]
})
```

Now, you have to authenticate as this user in order to proceed with further operations (you must be connected to the ‘admin’ database in order to authenticate as this user):

```
> db.auth(“superuser”,”superpass”)
```

After this step, you can continue your work on this database server. You will switch back to ‘test’ database:

```
> use test
```

Now, in this database called ‘test’, you must create a collection called ‘persons’. A collection of documents resembles a relational table, but there are some key differences. A collection has a flexible schema, which means that it can contain documents that don’t have the same structure.

You will list the name of the collections that are stored in your current database:

```
> show collections
```

Now, you will create a new collection called ‘persons’. You can either create a collection explicitly by using the ‘db.createCollection()’ database method or you can simply create it by inserting the first document in it, like this:

```
> db.persons.insert({
"firstname":"Anna",
"lastname":"Brown",
hobbies:["yoga", "cooking", "sailing"],
"address":{
              "street": "Dekalb Avenue",
              "number":10,
              "city":"New York"
          }
})
```

You have inserted your first document. To return all documents from the ‘persons’ collection, you have to use the collection method called ‘find()’ :

```
> db.persons.find()
```

To return all documents from the ‘persons’ collection in a more readable form, you have to append the ‘pretty()’ method to the previous one, like this:

```
> db.persons.find().pretty()
```

To return a random document from the ‘persons’ collection, you can use the collection method called ‘findOne()’:

```
> db.persons.findOne()
```

You will insert another document in this collection:

```
> db.persons.insert({
"firstname":"Sam",
"lastname":"Cricket",
hobbies:["hiking", "rafting"],
"address":{
              "street": "Alameda Street",
              "number":1,
              "city":"Los Angeles"
          },
"state": "California"
})
```

As you can see, the second document contains a field called ‘state’ that is not present in the first document. This is allowed because MongoDB has a flexible schema.

To count the number of documents in the ‘persons’ collection, you can use the collection method called ‘count()’:

```
> db.persons.count()
```

To return only those documents for which state is equal to “California”, you can execute the following query:

```
> db.persons.find({"state":"California"}).pretty()
```

To return only the ‘firstname’, ’lastname’ and ‘state’ fields of the documents for which state is equal to “California”, you can execute the following query:

```
> db.persons.find({"state":"California"},{"firstname":1,"lastname":1,"state":1,"_id":0}).pretty()
```

The second JSON document included in the previous query is called **projection document** and it specifies or restricts fields to return. As you already know, the ‘_id’ field acts as a primary key for every document and it appears by default in a query result (even if you don’t specify it in the projection document). If you want to exclude the ‘_id’ field from the query result, you can do so by writing “_id”: 0 in the projection document.

To return all documents for which number is greater than or equal to 1 and less than 10, you can run the following query:

```
> db.persons.find({"address.number":{"$gte":1, "$lt":10}}).pretty()
```

As you can see, to access embedded fields you have to use dot notation. In this case, because the field ‘number’ appears in the embedded document that corresponds to the ‘address’ field, you have to write ‘address.number’ in order to have access to its value.

To return all documents for which ‘address.number’ is greater than or equal to 1 and less than or equal to 10, sorted by ‘address.number’ in descending order, run the following query:

```
> db.persons.find({"address.number":{"$gte":1, "$lte":10}}).sort({"address.number":-1}).pretty()
```

The cursor method called ‘sort()’ receives a JSON document in which you specify the fields to sort by and the sort order (‘1’ means ascending, ‘-1’ means descending).

To return the document that has the greatest number (‘address.number’), you can use a cursor method called ‘limit()’:

```
> db.persons.find().sort({"address.number":-1}).limit(1).pretty()
```

You must specify a numeric value and pass it to the ‘limit()’ cursor method.

You insert another document in the ‘persons’ collection:

```
> db.persons.insert({
"firstname":"Sam",
"lastname":"Smith",
hobbies:["photography", "fishing"],
"address":{
              "street": "Alameda Street",
              "number":3,
              "city":"Los Angeles"
          },
"state": "California"
})
```

To update all documents that have ‘firstname’ equal to “Sam” and set the ‘address.street’ field value to “First Street”, you can use the collection method called ‘update()’:

```
> db.persons.update(
                     {"firstname":"Sam"},
                     {"$set":{"address.street":"First Street"}},
                     {"multi":true}
                   )
```

The last JSON document passed to the ‘update()’ method can contain one or more options. In the above case, setting the ‘multi’ option to true means that if the query criteria specified in the first JSON document (the first parameter passed to the ‘update()’ method) matches more than one document, all documents will be updated. By default, ‘multi’ is set to false.

To remove the field called ‘state’ from all documents, you will run the following update operation:

```
> db.persons.update(
                     {},
                     {"$unset":{"state":1}},
                     {"multi":true}
                   )
```

To add a field called ‘country’ to all documents and set its value to “United States of America”, you will run the following update operation:

```
> db.persons.update(
                     {},
                     {"$set":{"country":"United States of America"}},
                     {"multi":true}
                   )
```

To delete all documents that have ‘address.city’ equal to “Los Angeles”, you can use the collection method called ‘remove()’:

```
> db.persons.remove({"address.city":"Los Angeles"})
```

To delete the collection called ‘persons’, you can use the collection method called ‘drop()’:

```
> db.persons.drop()
```

Assignment:

1. Execute all commands listed above.
2. Insert the following documents in a collection called ‘students’ that belongs to the ‘test’ database:

{

"firstname" : "Bob",

"lastname" : "Tabor",

"city" : "New York",

"admission" : {"grade" :10, "year" : 2015},

"domain" : "Computer Science",

"yearofstudy" : 3,

"hobbies" : ["reading","camping"]

}

{

"firstname" : "Bob",

"lastname" : "Black",

"city" : "London",

"admission" : {"grade":9.4, "year": 2016},

```
"domain" : "Computer Science",
"yearofstudy" : 2,
"hobbies" : ["hiking","swimming","camping"]
}
{
"firstname" : "Anna",
"lastname" : "Grey",
"city" : "Padstow",
"admission" : {"grade":6.4, "year": 2017},
"domain" : "Computer Science",
"yearofstudy" : 1,
"hobbies" : ["surfing","cooking","photography"]
}
{
"firstname" : "Jane",
"lastname" : "Reed",
"city" : "Edingburg",
"admission" : {"grade":8.9, "year": 2016},
"domain" : "Biology",
"yearofstudy" : 2,
"hobbies" : ["travel","camping","photography"]
}
{
"firstname" : "James",
"lastname" : "Horner",
"admission" : {"grade":8.9, "year": 2015},
"domain" : "Biology",
"yearofstudy" : 3,
"hobbies" : ["travel","martial arts"]
```

```
}
```

3. Return all documents for which grade is greater than or equal to 6.4 and less than 10 and domain is equal to “Computer Science”. The documents must be sorted by firstname in ascending order.
4. Return only the ‘firstname’, ‘lastname’,’domain’ and ‘admission’ fields of the document that has the highest grade.
5. Set the ‘city’ field value to “London” for all documents that have the domain equal to “Biology”.
6. Delete all documents that have the admission year equal to 2015.