IMY 220 ● Lecture 21

MongoDB

The previous lecture introduced MongoDB and Atlas and focused on retrieving data (the R in CRUD)

Next we will look at creating, updating, and deleting data…

…as well as using MongoDB inside NodeJS

Adding new data

The most basic command for adding a new document in MongoDB is as follows:

db.<collection>.insertOne(document)

Adding new data

E.g. we can add a new document to our “users” collection as follows

```
db.users.insertOne({
   "id":12345678,
   "name":"Test",
   "surname":"User"
})
```

(~MySQL: *INSERT INTO users () VALUES (<document>)*)

Adding new data

Similar to adding new documents on Atlas, we can omit the _id key for Mongo to automatically generate one

Attempting to add new documents that have the same _id value as an existing document will cause an insert error and will not insert the new document

Adding new data

We can also add multiple documents at the same time as follows:

db.<collection>.insertMany([document1, document2,…])

Adding new data

NB: it is important to note that MongoDB does not enforce a document structure upon insert

You could, theoretically insert a new document with any sets of keys and values and it will be inserted (as long as it does not contain an existing _id)

Updating data

Similar to inserting data, we have two MQL statements for updating either one document or multiple documents. The syntax for updating a single document is:

db.<collection>.updateOne(filter, update)

filter refers to the data we want to match (similar to the query parameter for find)

update refers to the modifications we want to make

Updating data

E.g. to change the id of the user we added earlier

```
db.users.updateOne(
   {"id":12345678},
   {"$set":{"id":23456789}}
)
```

(~MySQL: *UPDATE users SET id=23456789 WHERE id=12345678*)

Updating data

In the previous example we used $set which is an update operator

MongoDB has many update operators for making updates, e.g. we can increase the age of a user with 1 as follows

```
db.users.updateOne(
   {"id":49671845},
   {"$inc":{"age":1}}
)
```

Updating data

Read up on more update operators here:
https://docs.mongodb.com/manual/reference/operator/update/

Updating data

Unlike the insertOne and insertMany statements, the general syntax for updateOne and updateMany are the same

The difference is that updateOne will match the first document according to the filter and only update said document…

…whereas updateMany will update all documents that match the filter

Updating data

NB: updating a non-existent field with $set will **not** cause an error

Instead, it will add the field and given value, e.g.

```
db.users.updateOne(
   {"id":23456789},
   {"$set":{"email":"test@gmail.com"}}
)
```

After executing this statement, the matched document should have an email field with the given value

Deleting data

Lastly, deleting documents from a collection follows a similar syntax than update

db.<collection>.deleteOne(filter)

db.<collection>.deleteMany(filter)

filter refers to the data we want to match

Deleting data

Similar to updating data, deleteOne finds the first document that matches the filter and deletes it…

…whereas deleteMany deletes all documents that match the filter

E.g., to delete our test user:

```
db.users.deleteMany({"id":23456789})
```

(~MySQL: *DELETE FROM users WHERE id=12345678*)

Deleting data

We can also delete entire collections with the drop statement

db.<collection>.drop()

NB: *“Dropping collections in the admin database or the config database can leave your cluster in an unusable state.”*

https://docs.mongodb.com/manual/reference/method/db.collection.drop/#behavior

Using with NodeJS

We can use NodeJS to connect to our database and make calls, similar to using the shell commands

For this example, we will assume that we already have a basic setup that serves a page with Node, Express, and React (similar to what we would have at the end of our Node+React lecture)

First step is to install mongo: npm install mongodb

Using with NodeJS

We then need to connect to our cluster. Use the connection string that we got in the previous slides.

For this example we will only use grab the basic details, so make sure that “Include full driver code example” is unchecked

Using with NodeJS

You should see a line that looks something like this:

```
mongodb+srv://username:<password>@demo.xxxxx.mongodb.net
/?retryWrites=true&w=majority
```

Where username is your username and xxxxx is your unique connection string.

Note that you will need to replace <password> with your password (refer to the previous lecture).

Using with NodeJS

Next we need to use our connection string to connect to the cluster as follows inside *index.js*:

```
const {MongoClient} = require("mongodb");
const uri =
"mongodb+srv://username:IMY220@demo.xxxxx.mongodb.net/DBExample?retryWrites=true&w=majority";
const client = new MongoClient(uri);
```

Using with NodeJS

We also need to install a package called regenerator-runtime, which helps with transpiling async functions

npm install --save-dev regenerator-runtime

We also need to include the following in our index.js file

```
import "regenerator-runtime/runtime";
```

Using with NodeJS

So far your index.js should look something like this:

```
const express = require('express');
const app = express();
const http = require('http').Server(app);


const {MongoClient} = require("mongodb");
const uri =
"mongodb+srv://diffie:IMY220@demo.flv1y.mongodb.net/DBExample?retry
Writes=true&w=majority";
const client = new MongoClient(uri);


import "regenerator-runtime/runtime";
```

Using with NodeJS

Next, we want to be able to query our database

We will do this whenever a user connects to our server

```
http.listen(3000, async () => {
  // query will go here
  console.log('listening on *:3000');
});
```

Using with NodeJS

Next, we want to be able to query our database

We will do this whenever a user connects to our server

```
http.listen(3000, async () => {
  // query will go here
  console.log('listening on *:3000');
});
```

Using with NodeJS

We will define a function that runs a find query and returns the result

Since the query will happen asynchronously, we will define it as an async function containing a try…finally block

```
async function runFindQuery(collection, query, options) {
    try {
        // query code comes here
    } finally {
        await client.close();
    }
}
```

Using with NodeJS

First, we need to ensure that we have a connection to the database

```
await client.connect();
```

Then we define our database and collection to query from

```
const database = client.db('DBExample');
const col = database.collection(collection);
```

Using with NodeJS

Next, we use what is called a cursor to retrieve data

“Because a query can potentially match very large sets of documents, these operations rely upon an object called a cursor. A cursor fetches documents in batches to reduce both memory consumption and network bandwidth usage. Cursors are highly configurable and offer multiple interaction paradigms for different use cases.”

https://www.mongodb.com/docs/drivers/node/current/fundamentals/crud/read-operations/cursor/#std-label-cursor-methods

Using with NodeJS

We will approach this simply by using the cursor to retrieve the results from a single find query…

```
const cursor = col.find(query, options);
```

…and returning the results as an array

```
return await cursor.toArray();
```

Using with NodeJS

```
async function runFindQuery(collection, query, options) {
  try {
    await client.connect();
    const database = client.db('DBExample');
    const col = database.collection(collection);

    const cursor = col.find(query, options);

    return await cursor.toArray();
  } finally {
    await client.close();
  }
}
```

Using with NodeJS

Now we can execute our query whenever a user connects to our server

```
http.listen(3000, async () => {
  let results = await runFindQuery("collection",
                                   {"position":"student"},
                                   {"projection":{"name":1}}
  );
  // do something with results
  console.log('listening on *:3000');
});
```

That’s it!

Good luck with the semester test, project, and exam

References

https://university.mongodb.com/

https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/

https://docs.mongodb.com/manual/reference/operator/update/

https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/

https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/

https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/

References

https://stackoverflow.com/questions/63001989/referenceerror-cant-find-variable-regeneratorruntime-on-react

https://stackoverflow.com/questions/65378542/what-is-regenerator-runtime-npm-package-used-for

https://www.mongodb.com/docs/drivers/node/current/quick-start/

https://www.mongodb.com/docs/drivers/node/current/usage-examples/find/

https://www.mongodb.com/docs/drivers/node/current/fundamentals/crud/read-operations/cursor/#std-label-cursor-methods