Command Operations in MongoDB

This lesson introduces all of the basic Mongo Shell command operations that can be performed (e.g., creating, reading, updating and deleting from a database).

WE'LL COVER THE FOLLOWING

CRUD operations from the shell

Once the MongoDB server is up and running, one can connect to it using Mongo shell client. On the terminal, on our platform, the server automatically connects, so that you don't have to set it up.

All the functionalities that are provided in the shell client are provided through MongoDB drivers too. Drivers are provided for all popular programming languages, which means that all features that are shown in this chapter can (and should) be done through the code. However, for demonstrative purposes, the shell client is used.

> **Note:** To get a good understanding of all the commands, and for good practice, make sure to try out all of the commands in this lesson in the terminal provided at the end of this lesson.

It's important to note that MongoDB commands are *case-sensitive*.

Using `mongo`

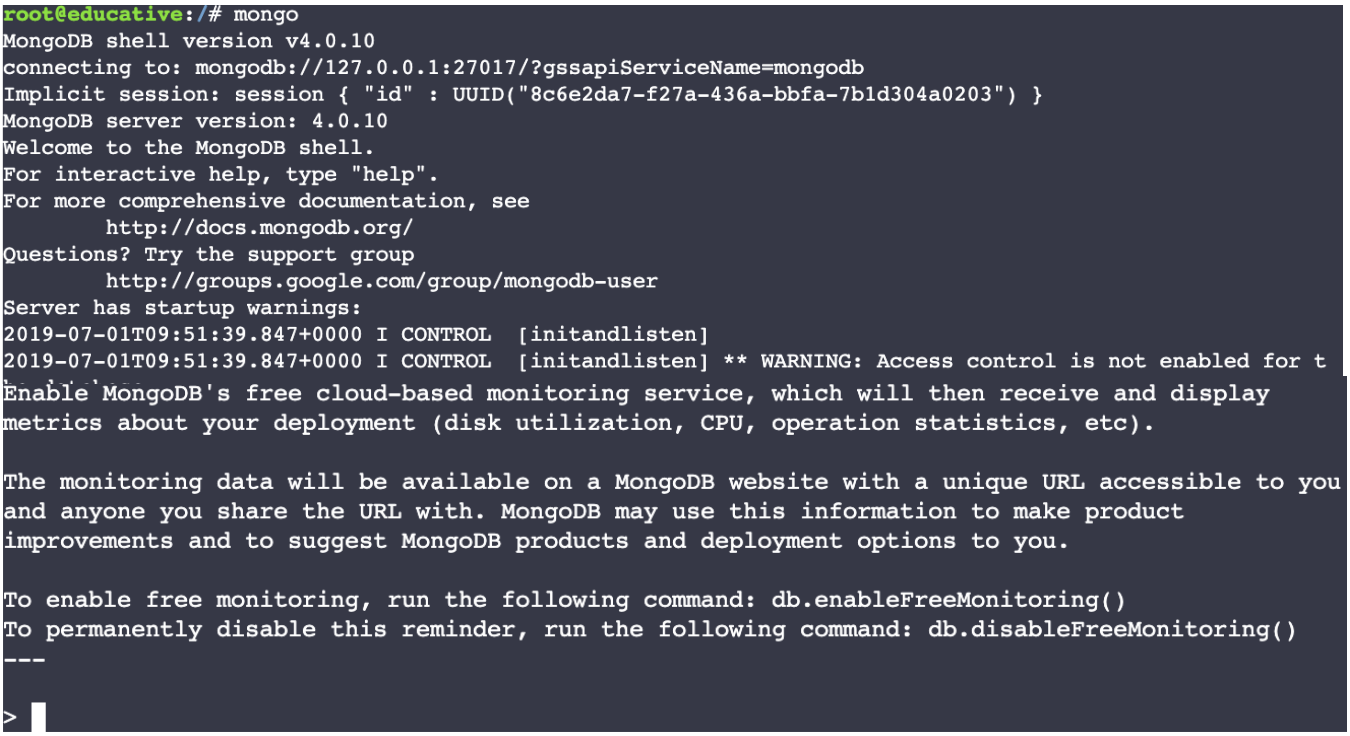
To start the shell client, type the following command in the terminal :

```
mongo
```


```
mongo
```


Starting the mongo shell

Running the above command in the terminal should give the following result:



Running the mongo command

You'll see that the client displays some basic information, such as *client* and *server* versions. Finally, the shell client brings the user to a prompt, where commands may be typed.

Creating a Database

At the top of the hierarchy, in the MongoDB server, is *database* entity. There can be *multiple* databases on a single server.

To create a database, MongoDB provides the command:

```
use DATABASE_NAME
```


```
use DATABASE_NAME
```


This command:

- is used when a user wants to switch from one database to another.
- will create a new database if it doesn't exist; otherwise, it will select the existing database.

If one needs to check on which database the client is connected to, `db`

command will display that information.

```
db
```

Displaying database name

Example

So, let's create our first database by typing this command in the terminal:

```
use blog
db
```

Creating a database called "blog"

Running the "use" & "db" commands

Using `db.help()`

Once the database is created and connected to, a user can manipulate the data inside of that database. This is done by using one of the many options of the `db` entity.

All the options will be displayed by typing in the

```
db.help()
```

method.

```
db.help()
```

Displaying options of the "db" entity

Displaying options of the "db" entity

Example

Try out the command given in the terminal above. You should see the following result:

```
> use blog
switched to db blog
> db
blog
> db.help()
DB methods:
        db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [ just calls db.runC
ommand(...) ]
        db.auth(username, password)
        db.cloneDatabase(fromhost)
        db.commandHelp(name) returns the help for the command
        db.copyDatabase(fromdb, todb, fromhost)
        db.createCollection(name, { size : ..., capped : ..., max : ... } )
        db.createUser(userDocument)
        db.currentOp() displays currently executing operations in the db
        db.dropDatabase()
        db.eval(func, args) run code server-side
        db.fsyncLock() flush data to disk and lock server for backups
        db.fsyncUnlock() unlocks server following a db.fsyncLock()
        db.getCollection(cname) same as db['cname'] or db.cname
        db.getCollectionInfos()
        db.getCollectionNames()
        db.getLastError() - just returns the err msg string
        db.getLastErrorObj() - return full status object
```

Running db.help() command

Collection Operations

Creating a Collection

Now, the first thing a user would need to do is create a collection.

This is achieved by using the

```
db.createCollection(COLLECTION_NAME, OPTIONS)
```

method.

Let's try it out. Type out the following command in the terminal:

```
db.createCollection("users");
```

Creating a collection "users"

Showing Collections

The list of existing collections can also be displayed using the

```
show collections
```

command.

You can try this command in the terminal:

```
show collections
```

Showing the collection "users"

Dropping Collections

Existing collections can be removed with the

```
db.COLLECTION_NAME.drop()
```

function.

Example

So, let's drop one collection, and use `show collections` to verify the result.

Type the commands below, in the terminal, to try it out:

```
db.users.drop();
show collections
```

Dropping the collection "users"

Running createCollection(), show collections & drop() commands

Once a collection is created, a user can *insert*, *read*, *update* and *remove*

documents from it.

Inserting a Document

For adding a document to the collection, the

```
db.COLLECTION_NAME.insert(JSON_DATA)
```

method is provided.

Alternatively, the

```
db.COLLECTION_NAME.save(JSON_DATA)
```

function can be used.

Example

This feature is displayed below:

```
db.users.insert({"name" : "Nikola Zivkovic", "blog" : "rubikscode.net", "numberofArticles" :
```

Inserting in a document in "users"

Typing the above command in the terminal should give the following result:

```
> db.users.insert({"name" : "Nikola Zivkovic", "blog" : "rubikscode.net", "numberOfArticles" : 10, "com
pany" : "Vega IT"});
WriteResult({ "nInserted" : 1 })
>
```

Running insert() command

Querying Documents

For a user to query documents from the database, the following functions are provided:

```
db.COLLECTION_NAME.find(QUERY)
```

and

```
db.COLLECTION_NAME.findOne(QUERY)
```

Data can also be presented in a formatted manner by appending the following method:

```
pretty()
```

> A query that is passed into the function is a JSON object too.

Example

MongoDB has an additional query language that gives a user the ability to do something similar to the `where` clause.

For example, if we wanted to read all the users that have the blog name – rubikscode.net, we could do something like this on terminal:

```
db.users.findOne({"blog" : "rubikscode.net"});
db.users.find({"blog" : "rubikscode.net"});
db.users.find({"blog" : "rubikscode.net"}).pretty();
```

Querying the document inserted

```
> db.users.findOne({"blog" : "rubikscode.net"});
{
        "_id" : ObjectId("5d2c2b3668a929b65ecbfc3e"),
        "name" : "Nikola Zivkovic",
        "blog" : "rubikscode.net",
        "numberofArticles" : 10,
        "company" : "Vega IT"
}
> db.users.find({"blog" : "rubikscode.net"});
{ "_id" : ObjectId("5d2c2b3668a929b65ecbfc3e"), "name" : "Nikola Zivkovic", "blog" : "rubikscode.net", "
numberofArticles" : 10, "company" : "Vega IT" }
> db.users.find({"blog" : "rubikscode.net"}).pretty();
{
        "_id" : ObjectId("5d2c2b3668a929b65ecbfc3e"),
        "name" : "Nikola Zivkovic",
        "blog" : "rubikscode.net",
        "numberofArticles" : 10,
        "company" : "Vega IT"
}
>
```

Running findOne(), find() & pretty() commands

As you can see, using `find()` returns data in a dense format whereas appending `pretty()`, to it returns data in an easy to read format.

Updating a Document

Every document in the database can be updated. This is achieved by using

```
db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

Example #

Let's increase the number of articles of our document.

Type out the following command in the terminal:

```
db.users.update({"blog" : "rubikscode.net"}, {$set : {"numberOfArticles" : 11}});
db.users.find({"blog" : "rubikscode.net"}).pretty();
```

Updating the document

You should see the following result:

```
> db.users.update({"blog" : "rubikscode.net"}, {$set : {"numberOfArticles" : 11}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.users.find({"blog" : "rubikscode.net"}).pretty();
{
        "_id" : ObjectId("5d2c2b3668a929b65ecbfc3e"),
        "name" : "Nikola Zivkovic",
        "blog" : "rubikscode.net",
        "numberofArticles" : 10,
        "company" : "Vega IT",
        "numberOfArticles" : 11
}
>
```

Running the update() command

The additional `$set` filter was used, which helped us specify exactly which fields, inside of the document, should be updated.

Removing a Document

If a user wants to remove a document from the collection, one can do it with the

```
db.collection_name.remove(query)
```

method, as shown in the example below.

Example #

Type out the following commands to implement this operation in the terminal:

```
db.users.find().pretty();
db.users.remove({"blog" : "rubikscode.net"});
```

```
db.users.find().pretty();
```

Removing the document

You should be able to see the following:

```
> db.users.find().pretty();
{
        "_id" : ObjectId("5d106f093e460ca9be93562c"),
        "name" : "Nikola Zivkovic",
        "blog" : "rubikscode.net",
        "numberOfArticles" : 10,
        "company" : "Vega IT"
}
> db.users.remove({"blog" : "rubikscode.net"});
WriteResult({ "nRemoved" : 1 })
> db.users.find().pretty();
>
```

Running the remove() command

Indexes

To achieve better performance, MongoDB provided *indexes*. They are not different from the traditional indexes of relational databases; however, without them, MongoDB must scan every document from a collection to select those documents that match the query statement.

This scan is highly inefficient because of the large volume of data.

> **Indexes** are data structures which store a small portion of the data that is easily accessible through MongoDB.

Creating an Index

In order to create an index on field of a collection, one should run:

```
db.COLLECTION_NAME.createIndex({ FIELD_NAME: 1})
```

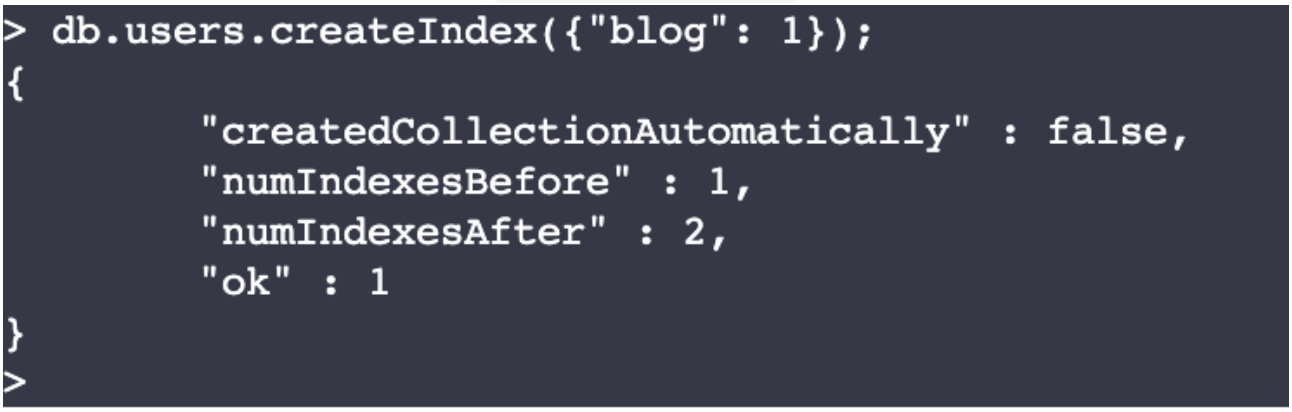
Example

Let's look at an example of how it's done!

Type out the following commands in the terminal below:

```
db.users.createIndex({"blog" : 1});
```

Creating an Index



Running the createIndex command

Terminal

Try out all of the commands mentioned above in the terminal given below!

Terminal

Go back (click here to go back above).

There are many other functionalities of the MongoDB language that we haven't covered here, but these should be enough for the beginning. For more details on the MongoDB API itself, you can check out this page.

> Look at the Appendix chapter to see how to install and run mongo on your local machine.

In the next lesson, we will take a look at the *MongoDB Compass*.