

Name: Shubham Lad

Class: MSc Computer Science (Part 1)

Roll No: 512

Semester: II

Subject: Big Data

Topic: Mongo DB

Introduction

MongoDB was developed by 10gen and was founded in 2007 in New York city. MongoDB, the most popular NoSQL database, is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON (like JSON format).

RDBMS vs MongoDB:

RDBMS has a typical schema design that shows number of tables and the relationship between these tables whereas MongoDB is document oriented. There is no concept of schema or relationship. Complex transactions are not supported in MongoDB because complex join operations are not available. MongoDB allows a highly flexible and scalable document structure. For example, one data document of a collection in MongoDB can have two fields whereas the other document in the same collection can have four.

MongoDB is faster as compared to RDBMS due to efficient indexing and storage techniques. There are a few terms that are related in both databases. What's called Table in RDBMS is called a Collection in MongoDB. Similarly, a Table is called a Document and A Column is called a Field. MongoDB provides a default '_id' (if not provided explicitly) which is a 12-byte hexadecimal number that assures the uniqueness of every document. It is like the Primary key in RDBMS.

Features of MongoDB:

- Document Oriented: MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like CPU, RAM, Hard disk, etc.

- Indexing: Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.
- Scalability: MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. Also, new machines can be added to a running database.
- Replication and High Availability: MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.
- Aggregation: Aggregation operations process data records and return the computed results. It is like the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

Where do we use MongoDB?

MongoDB is preferred over RDBMS in the following scenarios:

- **Big Data**: If you have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharding your database.
- **Unstable Schema**: Adding a new column in RDBMS is hard whereas MongoDB is schema-less. Adding a new field does not affect old documents and will be very easy.
- **Distributed data** Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

Language Support by MongoDB:

MongoDB currently provides official driver support for all popular programming languages like C, C++, Rust, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go, and Erlang.

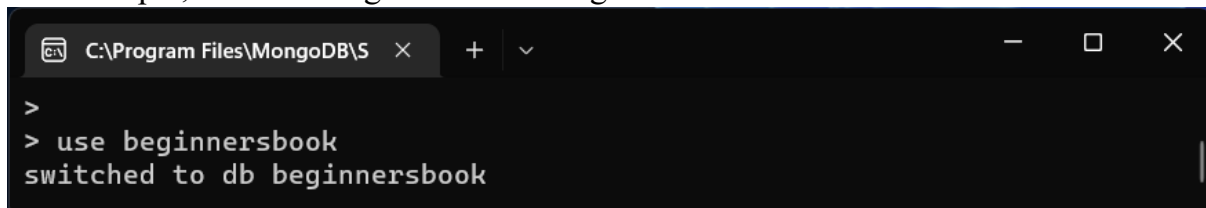
After installation of the MongoDB Compass on your PC, navigate to the following location “C:\Program Files\MongoDB\Server\4.0\bin” and double-click the mongo file present at that location. This will open a new command prompt of MongoDB wherein we can perform our queries.

Create Database

Once you are in the MongoDB shell, create the database in MongoDB by typing this command:

```
use database_name
```

For example, I am creating a database “beginnersbook” so the command should be:

A screenshot of a MongoDB shell command prompt window. The title bar shows the path 'C:\Program Files\MongoDB\Server\4.0\bin' and standard window controls. The command prompt shows a prompt character '>' followed by the command 'use beginnersbook'. The output of the command is 'switched to db beginnersbook'.

Note: If the database name you mentioned is already present then this command will connect you to the database. However, if the database doesn’t exist then this will create the database with the given name and connect you to it.

To list down all the databases, use the command

```
show dbs
```

This command lists down all the databases and their size on the disk.

```
C:\Program Files\MongoDB\>
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
```

As you can see that the database “beginnersbook” that we have created is not present in the list of all the databases. This is because a database is not created until you save a document in it

Now we are creating a collection user and inserting a document in it.

```
C:\Program Files\MongoDB\>
> db.user.insert({name:"Shubham",age:21})
WriteResult({ "nInserted" : 1 })
>
```

```
C:\Program Files\MongoDB\>
> show dbs
admin      0.000GB
beginnersbook 0.000GB
config     0.000GB
```

You can now see that the database “beginnersbook” is created.

Drop Database

The syntax to drop a Database is:

```
db.dropDatabase()
```

We do not specify any database name in this command, because this command deletes the currently selected database.

Now, the currently selected database is “beginnersbook” so the command `db.dropDatabase()` would delete this database.

```
C:\Program Files\MongoDB\>
> db.dropDatabase()
{ "dropped" : "beginnersbook", "ok" : 1 }
>
```

To verify that the database is deleted successfully. Execute the `show dbs` command again to see the list of databases after deletion.

```
C:\Program Files\MongoDB\>
>
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
```

The database “beginnersbook” is not present in the list that means it has been dropped successfully.

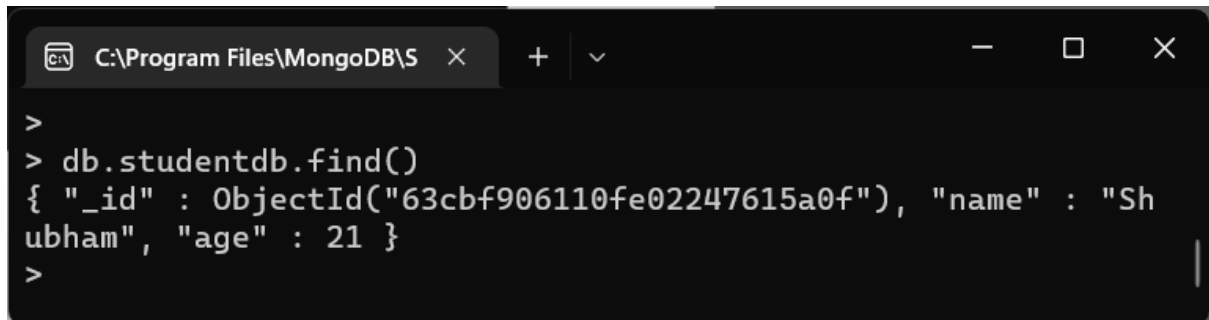
Creating Collection

```
C:\Program Files\MongoDB\>
> use studentdb
switched to db studentdb
>
```

```
C:\Program Files\MongoDB\>
> db.studentdb.insert({
... name:"Shubham",
... age:21,
... })
WriteResult({ "nInserted" : 1 })
```

To check whether the document is successfully inserted, type the following command. It shows all the documents in the given collection.

Syntax: `db.collection_name.find()`

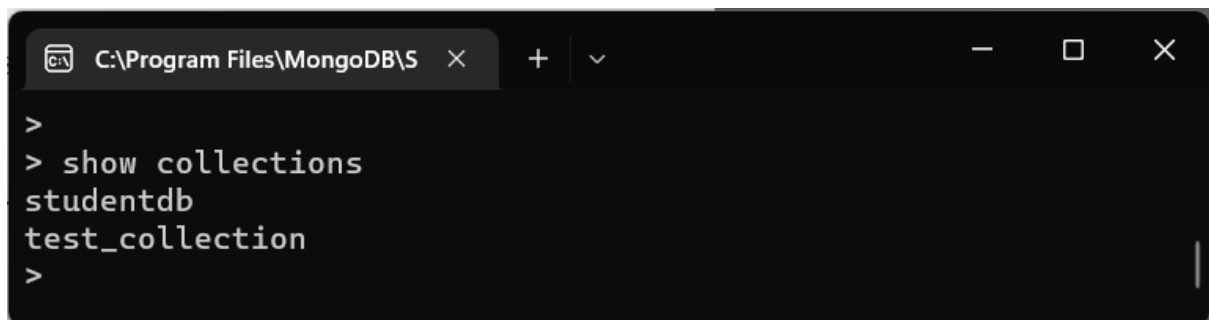


```
C:\Program Files\MongoDB\>
> db.studentdb.find()
{ "_id" : ObjectId("63cbf906110fe02247615a0f"), "name" : "Shubham", "age" : 21 }
>
```

To check whether the collection is created successfully, use the following command.

`show collections`

This command shows the list of all the collections in the currently selected database.



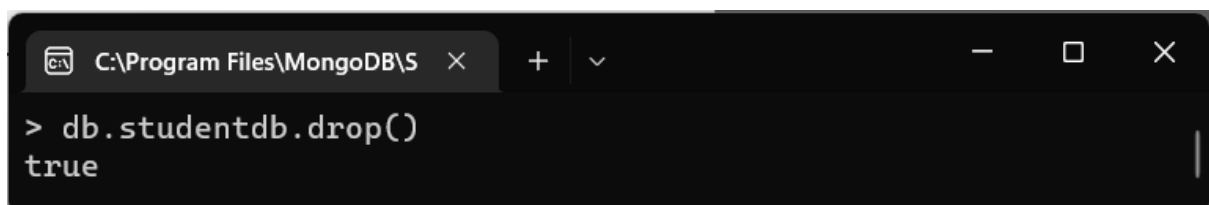
```
C:\Program Files\MongoDB\>
> show collections
studentdb
test_collection
>
```

Dropping Collection

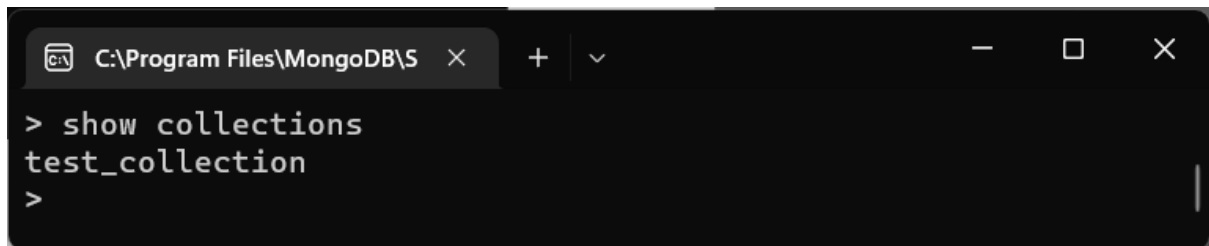
To drop a collection, first connect to the database in which you want to delete collection and then type the following command to delete the collection:

`db.collection_name.drop()`

Note: Once you drop a collection all the documents and the indexes associated with them will also be dropped.



```
C:\Program Files\MongoDB\>
> db.studentdb.drop()
true
```

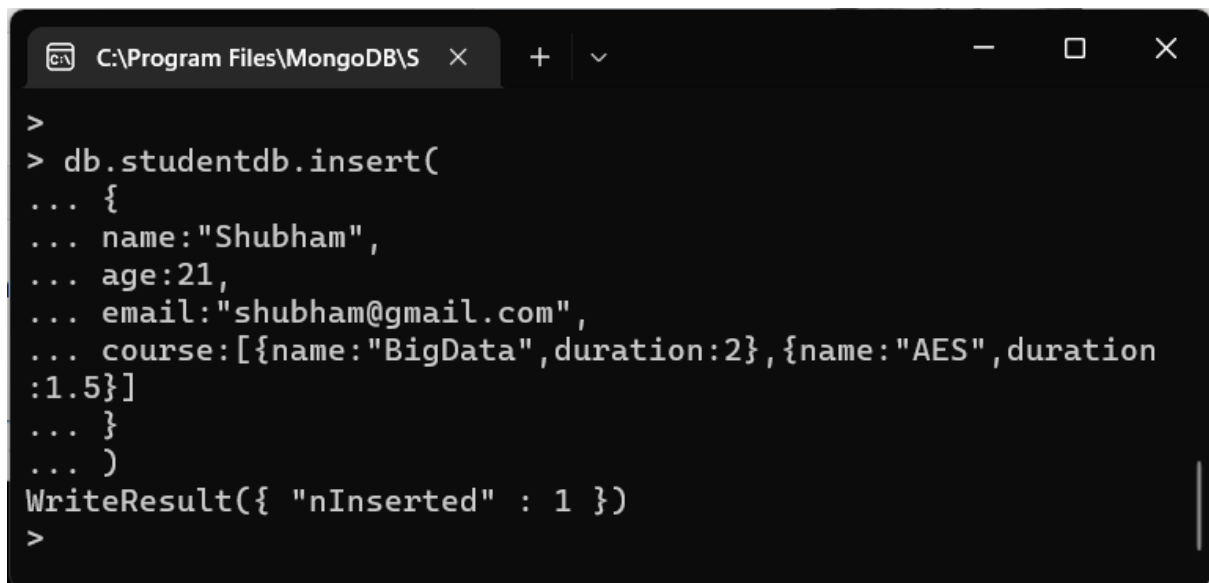


```
C:\Program Files\MongoDB\> show collections
test_collection
>
```

Inserting Document

Syntax to insert a document into the collection:

```
db.collection_name.insert()
```



```
C:\Program Files\MongoDB\>
> db.studentdb.insert(
... {
...   name:"Shubham",
...   age:21,
...   email:"shubham@gmail.com",
...   course:[{name:"BigData",duration:2},{name:"AES",duration
:1.5}]
... }
... )
WriteResult({ "nInserted" : 1 })
>
```

The `insert()` method creates the collection if it doesn't exist but if the collection is present then it inserts the document into it.

To insert multiple documents in collection, we define an array of documents and later we use the `insert()` method on the array variable. Here we are inserting three documents in the collection named “test_collection”.


```
C:\Program Files\MongoDB\>
>
> var some_array=
... [
... {
...   "Name":"Test 1",
...   "Desc":"Description 1"
... },
... {
...   "Name":"Test 2",
...   "Desc":"Description 2"
... },
... {
...   "Name":"Test 3"
...   "Desc":"Description 3"
... },
... ];
>
```

```
C:\Program Files\MongoDB\> db.test_collection.insert(some_array)
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 3,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```

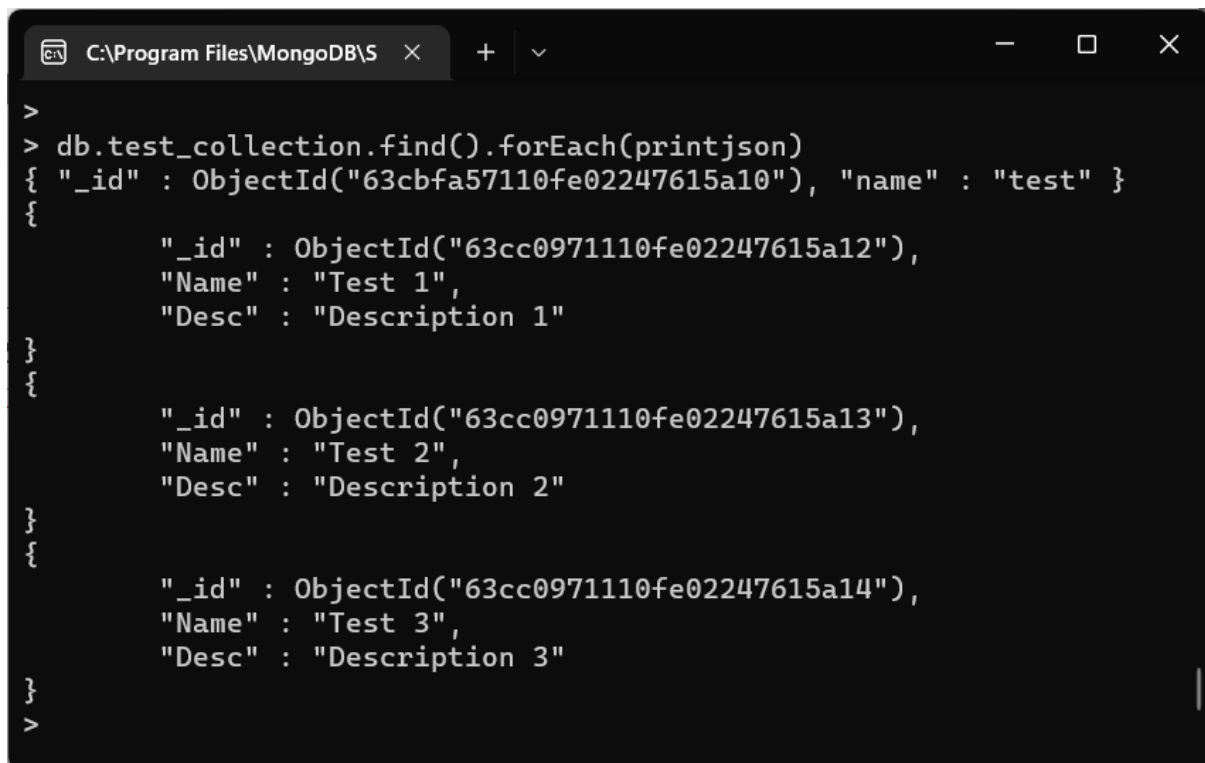
To verify that the documents are there in collection. Run this command:

`db.test_collection.find()`

```
C:\Program Files\MongoDB\> db.test_collection.find()
{ "_id" : ObjectId("63cbfa571110fe02247615a10"), "name" : "test" }
{ "_id" : ObjectId("63cc09711110fe02247615a12"), "Name" : "Test 1", "Desc" : "Description 1" }
{ "_id" : ObjectId("63cc09711110fe02247615a13"), "Name" : "Test 2", "Desc" : "Description 2" }
{ "_id" : ObjectId("63cc09711110fe02247615a14"), "Name" : "Test 3", "Desc" : "Description 3" }
>
```

We can print the output data in a JSON format so that you can read it easily. To print the data in JSON format run the command:

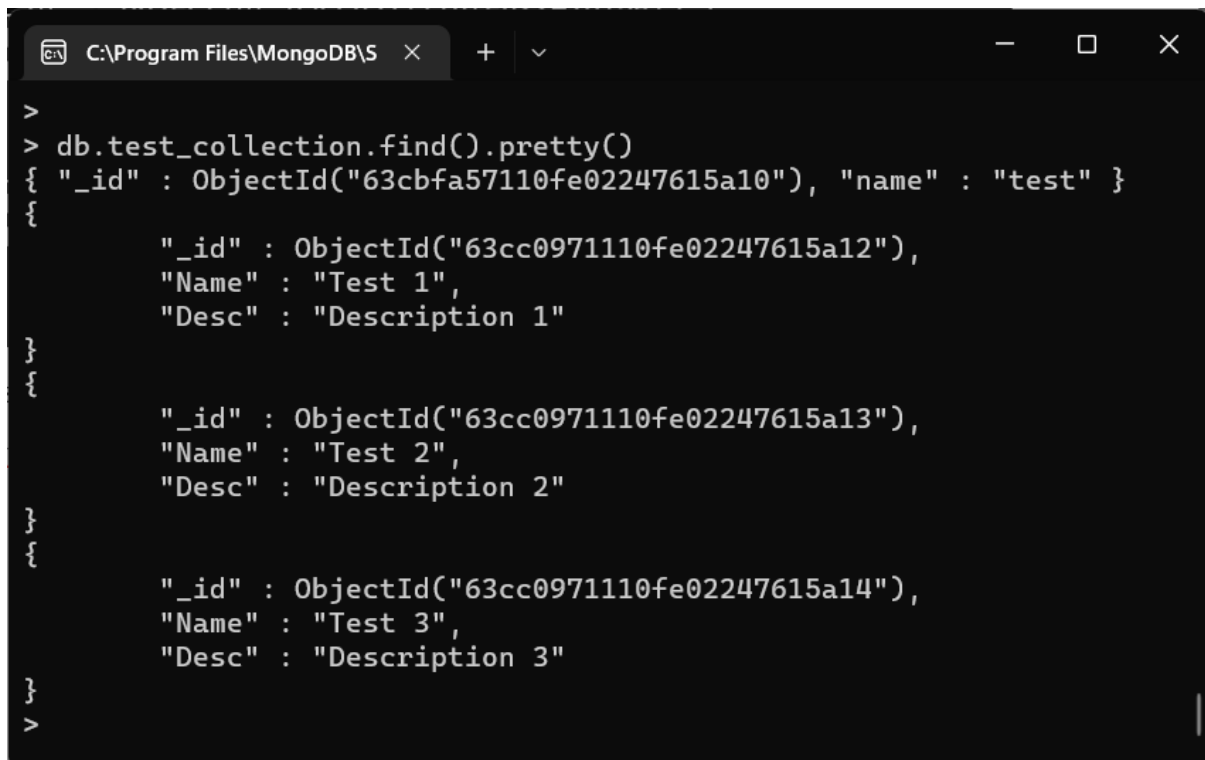
```
db.collection_name.find().forEach(printjson)
```



```
C:\Program Files\MongoDB\>
> db.test_collection.find().forEach(printjson)
{ "_id" : ObjectId("63cbfa57110fe02247615a10"), "name" : "test" }
{
  "_id" : ObjectId("63cc0971110fe02247615a12"),
  "Name" : "Test 1",
  "Desc" : "Description 1"
}
{
  "_id" : ObjectId("63cc0971110fe02247615a13"),
  "Name" : "Test 2",
  "Desc" : "Description 2"
}
{
  "_id" : ObjectId("63cc0971110fe02247615a14"),
  "Name" : "Test 3",
  "Desc" : "Description 3"
}
```

Or simply use pretty() – It does the same thing.

```
db.collection_name.find().pretty()
```



```
C:\Program Files\MongoDB\>
> db.test_collection.find().pretty()
{ "_id" : ObjectId("63cbfa57110fe02247615a10"), "name" : "test" }
{
  "_id" : ObjectId("63cc0971110fe02247615a12"),
  "Name" : "Test 1",
  "Desc" : "Description 1"
}
{
  "_id" : ObjectId("63cc0971110fe02247615a13"),
  "Name" : "Test 2",
  "Desc" : "Description 2"
}
{
  "_id" : ObjectId("63cc0971110fe02247615a14"),
  "Name" : "Test 3",
  "Desc" : "Description 3"
}
```

Instead of fetching all the documents from collection, we can fetch selected documents based on a criterion.

```
C:\Program Files\MongoDB\>
> db.studentdb.find({name:"Shubham"}).pretty()
{
  "_id" : ObjectId("63cc004b110fe02247615a11"),
  "name" : "Shubham",
  "age" : 21,
  "email" : "shubham@gmail.com",
  "course" : [
    {
      "name" : "BigData",
      "duration" : 2
    },
    {
      "name" : "AES",
      "duration" : 1.5
    }
  ]
}
```

```
C:\Program Files\MongoDB\>
> db.studentdb.find({age:{$gt:22}}).pretty()
{
  "_id" : ObjectId("63cc17c2110fe02247615a15"),
  "name" : "Mayuresh",
  "age" : 24,
  "email" : "mayuresh@gmail.com",
  "course" : [
    {
      "name" : "Compiler",
      "duration" : 2
    },
    {
      "name" : "SNA",
      "duration" : 2
    }
  ]
}
```

```
C:\Program Files\MongoDB\S x + v
>
> db.studentdb.find({age:{$lt:22}}).pretty()
{
  "_id" : ObjectId("63cc004b110fe02247615a11"),
  "name" : "Shubham",
  "age" : 21,
  "email" : "shubham@gmail.com",
  "course" : [
    {
      "name" : "BigData",
      "duration" : 2
    },
    {
      "name" : "AES",
      "duration" : 1.5
    }
  ]
}
```

```
C:\Program Files\MongoDB\S x + v
{
> db.studentdb.find({age:{$ne:21}}).pretty()
{
  "_id" : ObjectId("63cc17c2110fe02247615a15"),
  "name" : "Mayuresh",
  "age" : 24,
  "email" : "mayuresh@gmail.com",
  "course" : [
    {
      "name" : "Compiler",
      "duration" : 2
    },
    {
      "name" : "SNA",
      "duration" : 2
    }
  ]
}
```

Updating Document in a Collection

In MongoDB, we have two ways to update a document in a collection. 1) `update()` method 2) `save()` method. Although both the methods update an existing document, they are being used in different scenarios. The `update()` method is used when we need to update the values of an existing document while `save()` method is used to replace the existing document with the document that has been passed in it.

To update a document in MongoDB, we provide a criterion in command and the document that matches that criteria is updated.

1) `update()`

Syntax: `db.collection_name.update(criteria, update_data)`



```
C:\Program Files\MongoDB\S > db.studentdb.find().pretty()
{
  "_id" : ObjectId("63cc004b110fe02247615a11"),
  "name" : "Shubham",
  "age" : 21,
  "email" : "shubham@gmail.com",
  "course" : [
    {
      "name" : "BigData",
      "duration" : 2
    },
    {
      "name" : "AES",
      "duration" : 1.5
    }
  ]
}
{
  "_id" : ObjectId("63cc17c2110fe02247615a15"),
  "name" : "Mayuresh",
  "age" : 24,
  "email" : "mayuresh@gmail.com",
  "course" : [
    {
      "name" : "Compiler",
      "duration" : 2
    },
    {
      "name" : "SNA",
      "duration" : 2
    }
  ]
}
```

```
C:\Program Files\MongoDB\5  x + v - □
>
> db.studentdb.update({"name":"Mayuresh Mhatre"},{$set:{"name":"Mithun Parab"}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
>
```

```
C:\Program Files\MongoDB\5  x + v - □
>
>
> db.studentdb.update({"name":"Mayuresh"},{$set:{"name":"Mithun"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studentdb.find().pretty()
{
  "_id" : ObjectId("63cc004b110fe02247615a11"),
  "name" : "Shubham",
  "age" : 21,
  "email" : "shubham@gmail.com",
  "course" : [
    {
      "name" : "BigData",
      "duration" : 2
    },
    {
      "name" : "AES",
      "duration" : 1.5
    }
  ]
}
{
  "_id" : ObjectId("63cc17c2110fe02247615a15"),
  "name" : "Mithun",
  "age" : 24,
  "email" : "mayuresh@gmail.com",
  "course" : [
    {
      "name" : "Compiler",
      "duration" : 2
    },
    {
      "name" : "SNA",
      "duration" : 2
    }
  ]
}
>
```

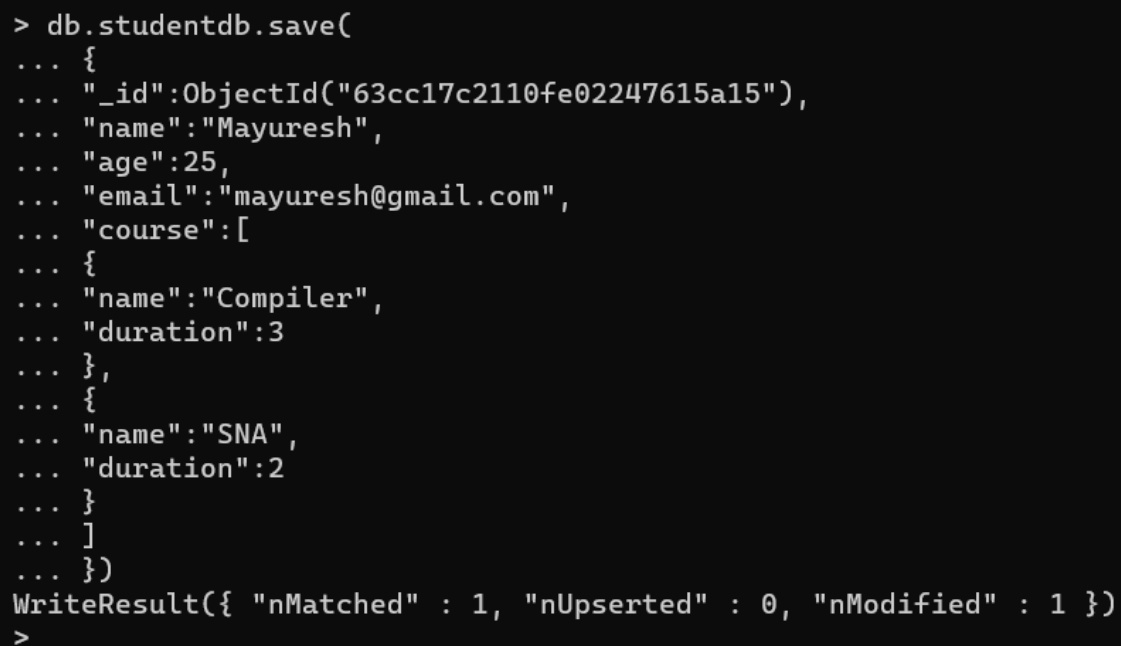
2) save()

Syntax: `db.collection_name.save({_id:ObjectId(), new_document})`

To work with `save()` method we should know the `unique_id` field of that document.

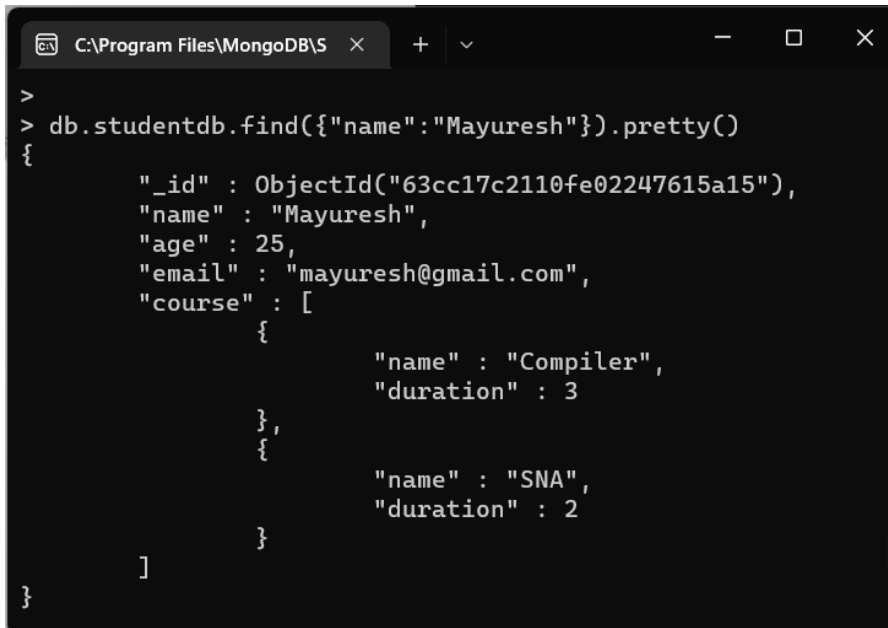
Note: When we do not provide the `_id` field while using `save()` method, it calls `insert()` method and the passed document is inserted into the collection as a new document.

```
> db.studentdb.find({"name":"Mithun"}).pretty()
{
  "_id" : ObjectId("63cc17c2110fe02247615a15"),
  "name" : "Mithun",
  "age" : 24,
  "email" : "mayuresh@gmail.com",
  "course" : [
    {
      "name" : "Compiler",
      "duration" : 2
    },
    {
      "name" : "SNA",
      "duration" : 2
    }
  ]
}
```



```
C:\Program Files\MongoDB\S > db.studentdb.save(
... {
...   "_id":ObjectId("63cc17c2110fe02247615a15"),
...   "name":"Mayuresh",
...   "age":25,
...   "email":"mayuresh@gmail.com",
...   "course":[
...     {
...       "name":"Compiler",
...       "duration":3
...     },
...     {
...       "name":"SNA",
...       "duration":2
...     }
...   ]
... }
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Verify the update using the `pretty()` command.



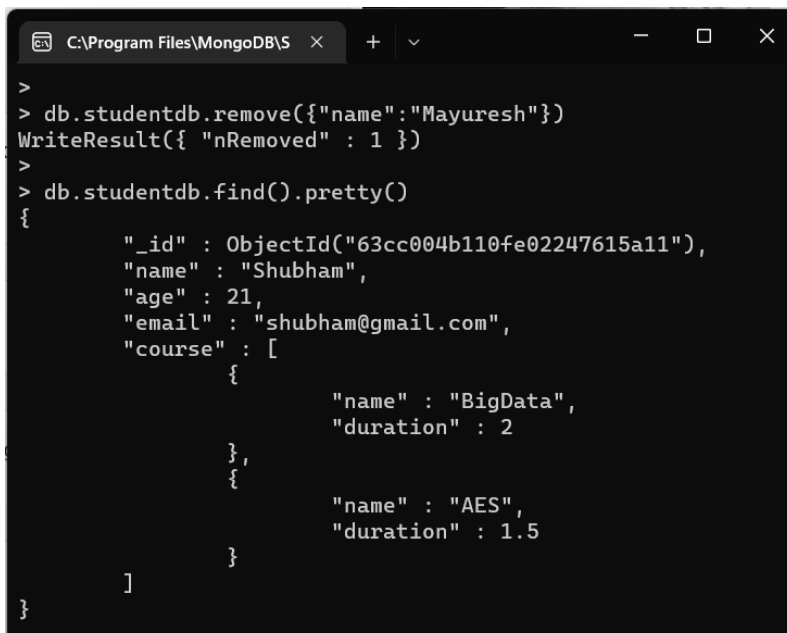
```
>
> db.studentdb.find({"name": "Mayuresh"}).pretty()
{
  "_id" : ObjectId("63cc17c2110fe02247615a15"),
  "name" : "Mayuresh",
  "age" : 25,
  "email" : "mayuresh@gmail.com",
  "course" : [
    {
      "name" : "Compiler",
      "duration" : 3
    },
    {
      "name" : "SNA",
      "duration" : 2
    }
  ]
}
```

Deletion of Document from a Collection

The `remove()` method is used for removing the documents from a collection in MongoDB.

Syntax of `remove()` method:

`db.collection_name.remove(delete_criteria)`



```
>
> db.studentdb.remove({"name": "Mayuresh"})
WriteResult({ "nRemoved" : 1 })
>
> db.studentdb.find().pretty()
{
  "_id" : ObjectId("63cc004b110fe02247615a11"),
  "name" : "Shubham",
  "age" : 21,
  "email" : "shubham@gmail.com",
  "course" : [
    {
      "name" : "BigData",
      "duration" : 2
    },
    {
      "name" : "AES",
      "duration" : 1.5
    }
  ]
}
```

limit()

This method limits the number of documents returned in response to a particular query.

Syntax:

```
db.collection_name.find().limit(number_of_documents)
```

```
C:\Program Files\MongoDB\S  ×  +  ∨  -  □  ×

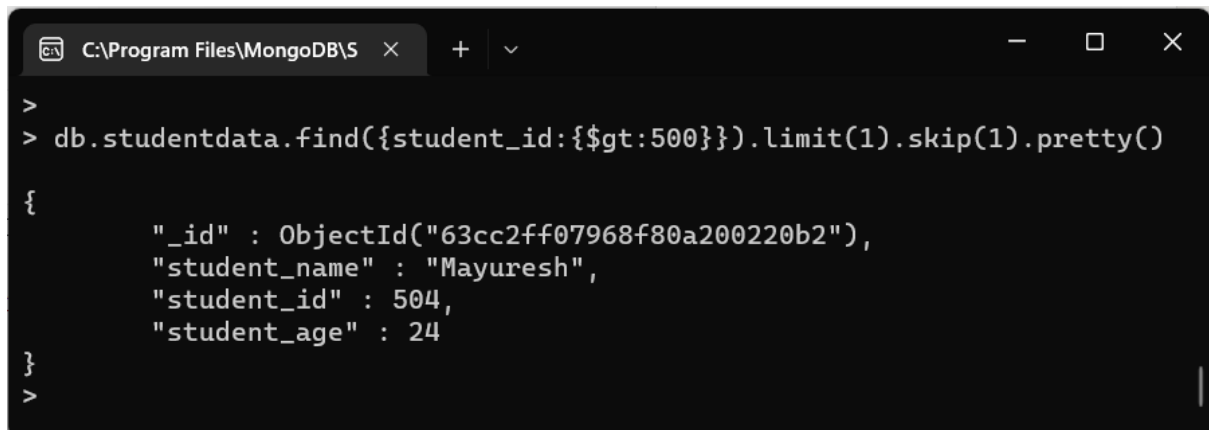
>
> db.studentdata.find().pretty()
{
  "_id" : ObjectId("63cc2fd77968f80a200220b1"),
  "student_name" : "Shubham",
  "student_id" : 512,
  "student_age" : 21
}
{
  "_id" : ObjectId("63cc2ff07968f80a200220b2"),
  "student_name" : "Mayuresh",
  "student_id" : 504,
  "student_age" : 24
}
{
  "_id" : ObjectId("63cc2ffc7968f80a200220b3"),
  "student_name" : "Mithun",
  "student_id" : 535,
  "student_age" : 22
}
>
```

```
C:\Program Files\MongoDB\S  ×  +  ∨  -  □  ×

>
> db.studentdata.find({student_id:{$gt:500}}).limit(1).pretty()
{
  "_id" : ObjectId("63cc2fd77968f80a200220b1"),
  "student_name" : "Shubham",
  "student_id" : 512,
  "student_age" : 21
}
>
```

skip()

The `skip()` method is used for skipping the given number of documents in the Query result.



```
C:\Program Files\MongoDB\5 >
> db.studentdata.find({student_id:{$gt:500}}).limit(1).skip(1).pretty()
{
  "_id" : ObjectId("63cc2ff07968f80a200220b2"),
  "student_name" : "Mayuresh",
  "student_id" : 504,
  "student_age" : 24
}
```