

NRI INSTITUTE OF TECHNOLOGY

MONGO DB – LAB MANUAL

List of experiments

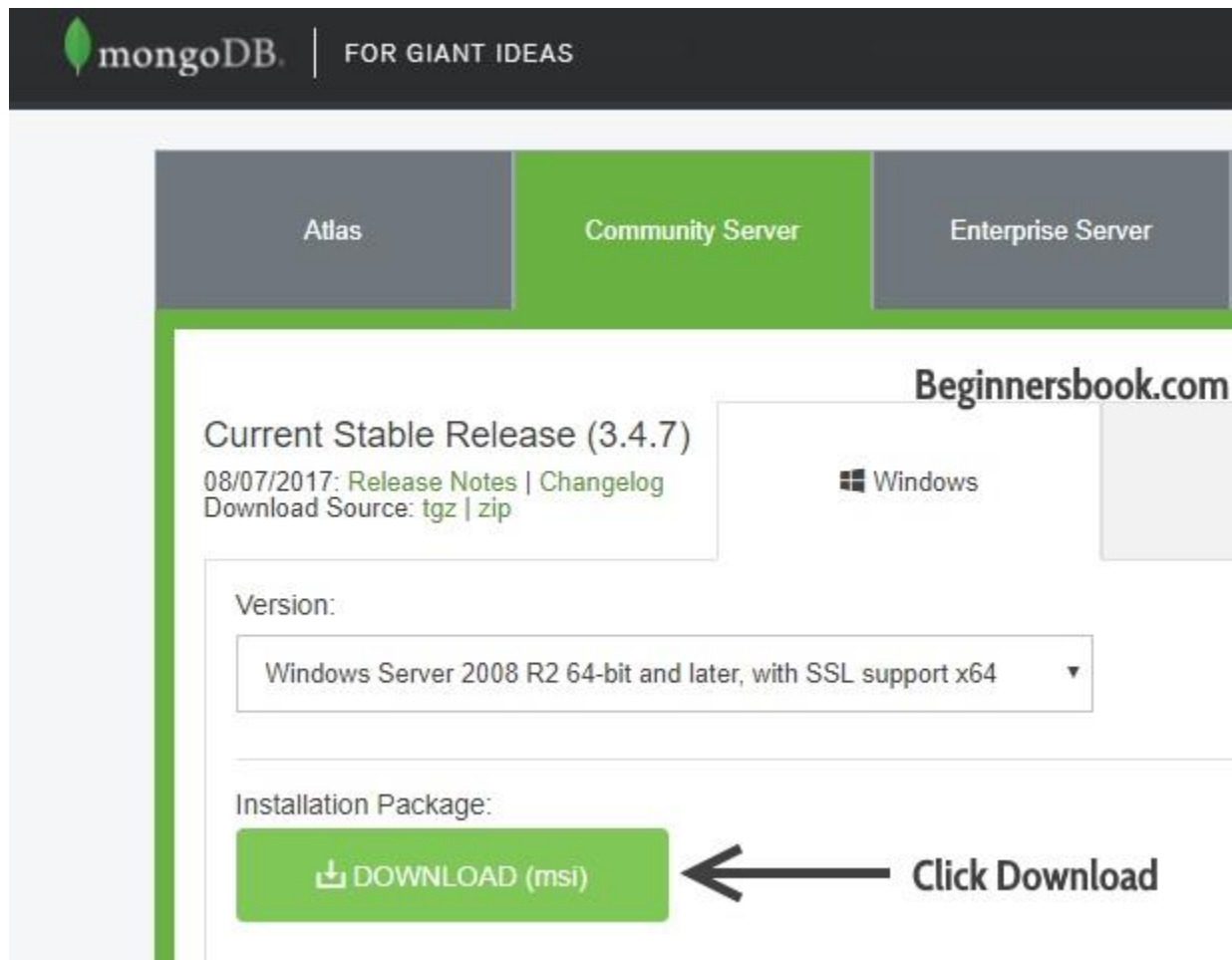
1. Mongo DB installation and configuration in windows.
2. Demonstrate how to create and drop a database in MongoDB.
3. Creating the Collection in MongoDB on the fly.
4. Creating collection with options before inserting the documents and drop the collection created.
5. MongoDB insert document
 - a.Insert single document
 - b.Insert multiple documents in collection
6. Querying all the documents in json format and Querying based on the criteria.
7. MongoDB update document
 - a.Using update()method.
 - b.Using save()method
8. MongoDB delete document from a collection .a.Using remove() method.
 - b.Remove only one document matching your criteriatic. Remove all documents
9. MongoDB Projection
- 10.limit(),skip(),sort()methods in MongoDB
- 11.MongoDB indexing
 - a.Create index in Mongo DB
 - b.Finding the indexes in a collection
 - c.Drop indexes in a collection
 - d.Drop all the indexes
- 12.MongoDB with java and PHP
 - a.Create a simple application that uses MongoDB with Java
 - b.Create a simple application that uses MongoDB with PHP

1. Mongo DB installation and configuration in windows.

Aim: To perform Mongo DB installation and configuration in windows.

Procedure:

Step 1: Go to [MongoDB download Page](#) and click download as shown in the screenshot. A .msi file like this **mongodb-win32-x86_64-2008plus-ssl-3.4.7-signed** will be downloaded in your system. Double click on the file to run the installer.



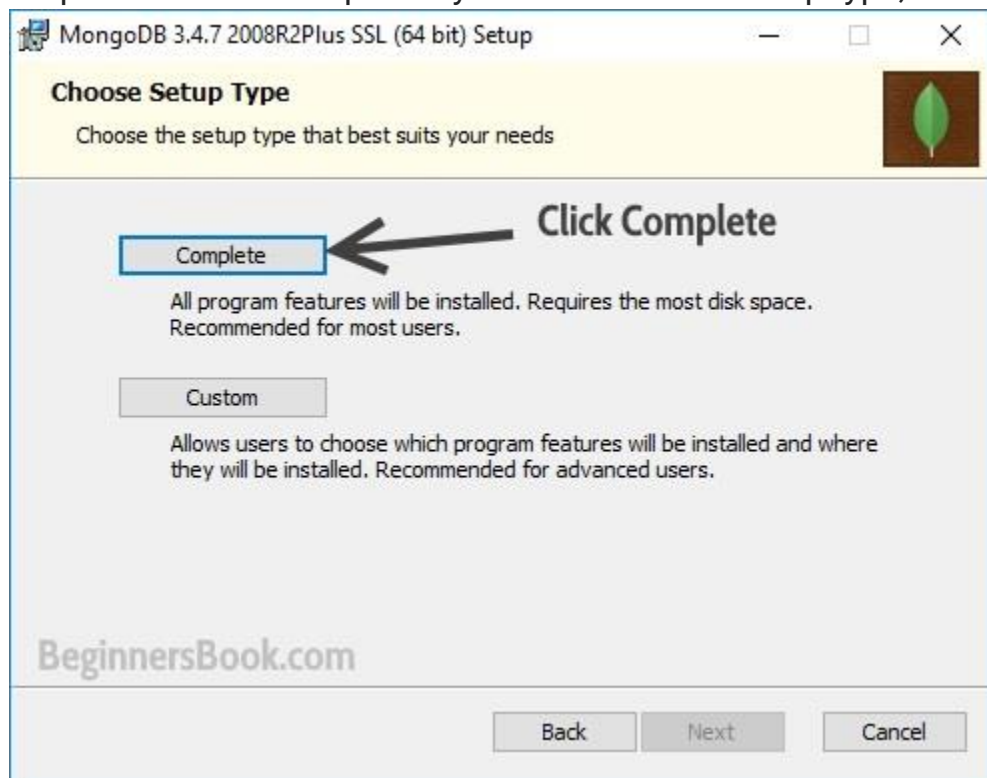
Step 2: Click Next when the MongoDB installation windows pops up.



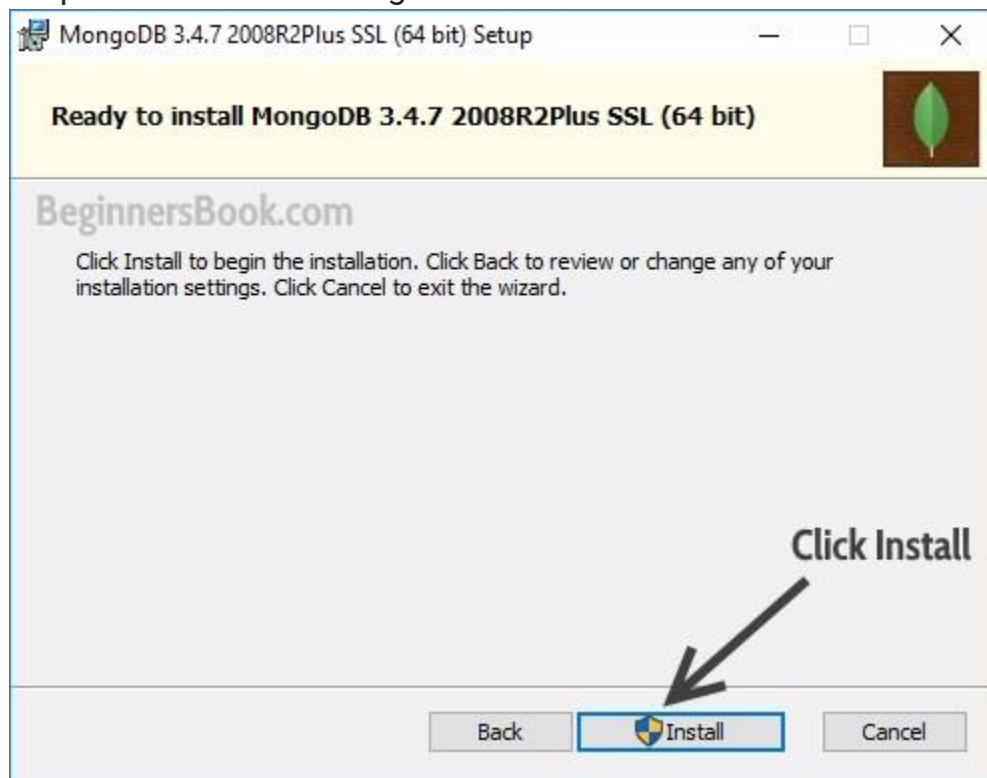
Step 3: Accept the MongoDB user Agreement and click Next.



Step 4: When the setup asks you to choose the Setup type, choose Complete.



Step 5: Click Install to begin the installation.



Step 6: That's it. Click Finish once the MongoDB installation is complete.

We are not done here. There are couple of steps we need to do before we can start using MongoDB.

MongoDB Configuration

Step 1: Locate the folder where you have installed MongoDB. If you have followed the above steps then you can find the folder at this location:

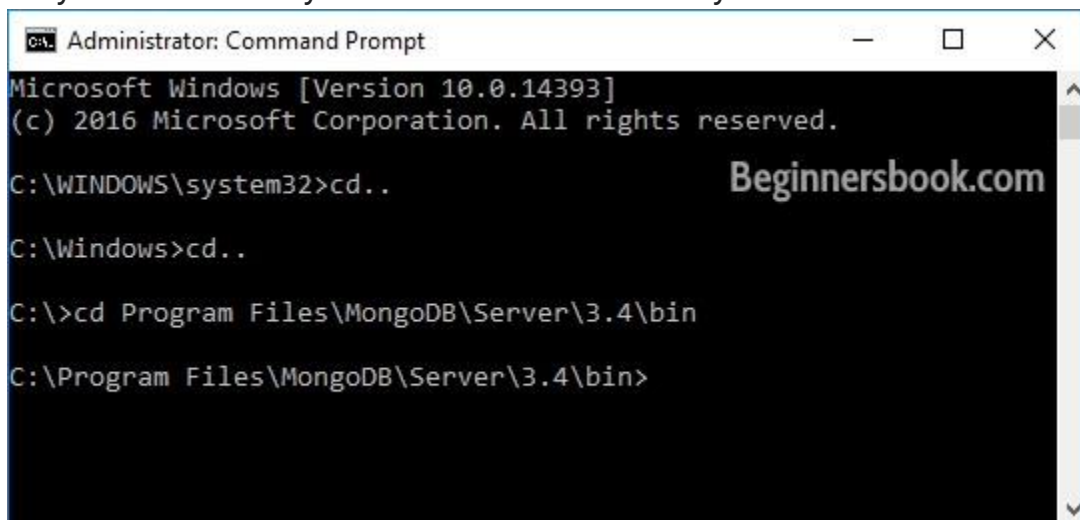
C:\Program Files\MongoDB

Here you need to create couple of folders that we need for MongoDB configuration.

1. Create two folders here, name them **data** and **log**.
2. Create another folder inside **data** and name it as **db**, that's where all the data will be stored.

That's it close the window.

Step 2: Open command prompt (right click and run as administrator). Navigate to the **bin** folder of MongoDB as shown in the screenshot. The path to the bin folder may be different in your case based on where you have installed the MongoDB.

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window has a black background with white text. The text shows the following commands and their outputs: "C:\WINDOWS\system32>cd.." resulting in "C:\Windows>"; "C:\Windows>cd.." resulting in "C:\>"; and ">cd Program Files\MongoDB\Server\3.4\bin" resulting in "C:\Program Files\MongoDB\Server\3.4\bin>". A watermark "Beginnersbook.com" is visible in the center-right of the window. The window includes standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\WINDOWS\system32>cd..  
C:\Windows>cd..  
C:\>cd Program Files\MongoDB\Server\3.4\bin  
C:\Program Files\MongoDB\Server\3.4\bin>
```

Step 3: Configure the data & log folder and set MongoDB as service by typing this command. **Note:** This is a one line command.

```
mongod --directoryperdb --dbpath "C:\Program Files\MongoDB\data\db"  
--logpath "C:\Program Files\MongoDB\log\mongo.log" --logappend --rest --  
install
```

```
Administrator: Command Prompt

C:\Program Files\MongoDB\Server\3.4\bin>mongod --directoryperdb --dbpath "C:\Program Files\MongoDB\data\db" --logpath "C:\Program Files\MongoDB\log\mongo.log" --logappend --rest --install
2017-09-14T18:23:44.014+0530 I CONTROL [main] ** WARNING: --rest is specified without --httpinterface,
2017-09-14T18:23:44.015+0530 I CONTROL [main] ** enabling http interface

C:\Program Files\MongoDB\Server\3.4\bin>
```

Beginnersbook.com

Step 4: Now you can start MongoDB as a service by typing this command:

```
net start MongoDB
```

You should see a message “MongoDB service was started successfully”.

```
Administrator: Command Prompt

C:\Program Files\MongoDB\Server\3.4\bin>mongod --directoryperdb --dbpath "C:\Program Files\MongoDB\data\db" --logpath "C:\Program Files\MongoDB\log\mongo.log" --logappend --rest --install
2017-09-14T18:23:44.014+0530 I CONTROL [main] ** WARNING: --rest is specified without --httpinterface,
2017-09-14T18:23:44.015+0530 I CONTROL [main] ** enabling http interface

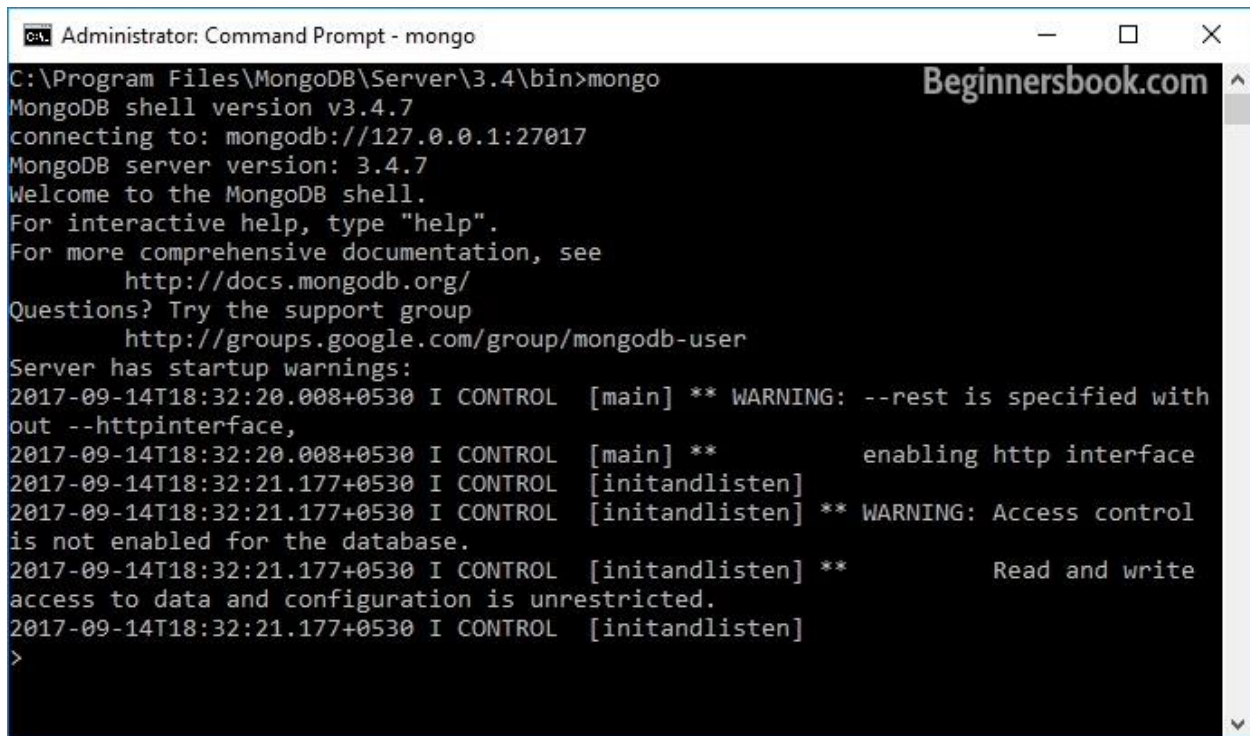
C:\Program Files\MongoDB\Server\3.4\bin>net start MongoDB
The MongoDB service is starting..
The MongoDB service was started successfully.

C:\Program Files\MongoDB\Server\3.4\bin>
```

Beginnersbook.com

That's it everything is done. Now we should be working in the MongoDB shell and we can run that by typing this command within the bin directory.

```
mongo
```

```
Administrator: Command Prompt - mongo
C:\Program Files\MongoDB\Server\3.4\bin>mongo
MongoDB shell version v3.4.7
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.7
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-09-14T18:32:20.008+0530 I CONTROL [main] ** WARNING: --rest is specified with
out --httpinterface,
2017-09-14T18:32:20.008+0530 I CONTROL [main] **          enabling http interface
2017-09-14T18:32:21.177+0530 I CONTROL [initandlisten]
2017-09-14T18:32:21.177+0530 I CONTROL [initandlisten] ** WARNING: Access control
is not enabled for the database.
2017-09-14T18:32:21.177+0530 I CONTROL [initandlisten] **          Read and write
access to data and configuration is unrestricted.
2017-09-14T18:32:21.177+0530 I CONTROL [initandlisten]
>
```

In the next tutorials we will learn how to work in the MongoDB shell.

If you want to exit the shell, you can do that by typing `quit()` or use `Ctrl-C` and then you can stop the MongoDB service with this command:

Conclusion: MONGODB is Installed on Windows

2. Demonstrate how to create and drop a database in MongoDB

Aim: To Demonstrate how to create and drop a database in MongoDB

PROCEDURE:

MongoDB Create Database

Start the MongoDB service by typing this command:

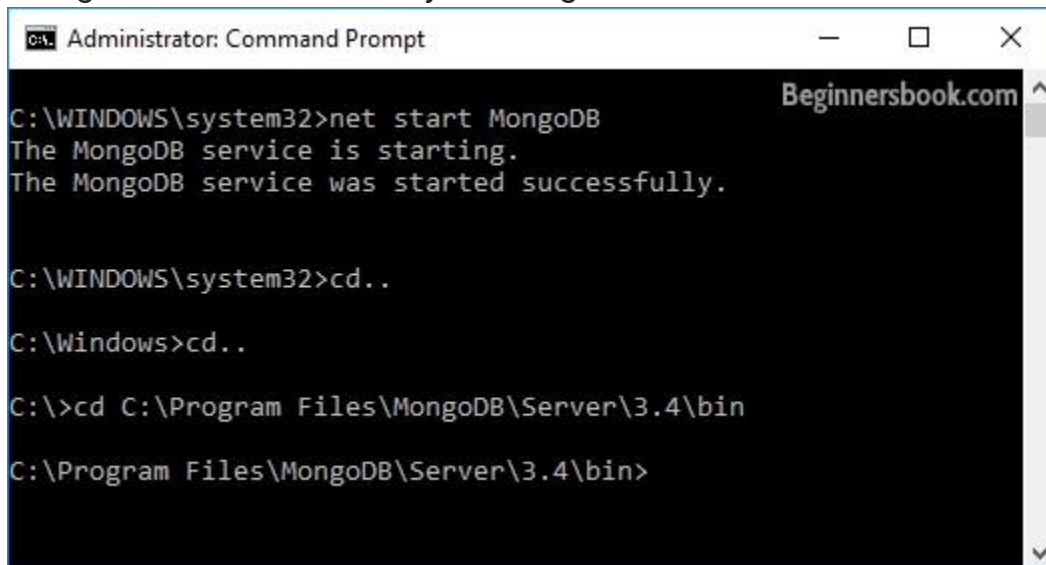
```
net start MongoDB
```

You should see a message like this:

The MongoDB service is starting..

The MongoDB service was started successfully.

Navigate to the bin directory of MongoDB as shown in the screenshot below:



```
Administrator: Command Prompt
C:\WINDOWS\system32>net start MongoDB
The MongoDB service is starting.
The MongoDB service was started successfully.

C:\WINDOWS\system32>cd..

C:\Windows>cd..

C:\>cd C:\Program Files\MongoDB\Server\3.4\bin
C:\Program Files\MongoDB\Server\3.4\bin>
```

Now we should be working in the MongoDB shell. To run the MongoDB shell, type the following command:

```
mongo
```

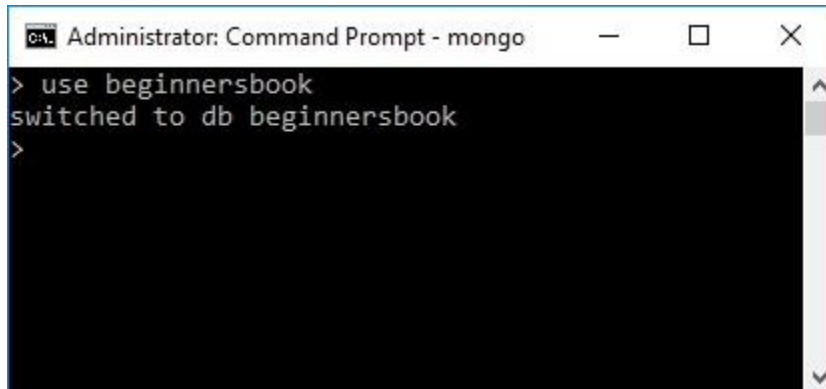
Once you are in the MongoDB shell, create the database in MongoDB by typing this command:

```
use database_name
```

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

```
use 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.



```
Administrator: Command Prompt - mongo
> use beginnersbook
switched to db beginnersbook
>
```

At any point if you want to check the currently connected database just type the command **db**. This command will show the database name to which you are currently connected. This is really helpful command when you are working with several databases so that before creating a collection or inserting a document in database, you may want to ensure that you are in the right database.

```
> db
beginnersbook
```



```
Administrator: Command Prompt - mongo
> db
beginnersbook
>
```

To list down all the databases, use the command **show dbs**. This command lists down all the databases and their size on the disk.

```
> show dbs
admin    0.000GB
local    0.000GB
```

```
Administrator: Command Prompt - mongo
> db
beginnersbook
> show dbs
admin    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.

We will learn how to create collection and document in the next tutorials.

```
> db.user.insert({name: "Chaitanya", age: 30})
WriteResult({ "nInserted" : 1 })
> show dbs
admin          0.000GB
beginnersbook  0.000GB
local          0.000GB
```

```
Administrator: Command Prompt - mongo
> db.user.insert({name: "Chaitanya", age: 30})
WriteResult({ "nInserted" : 1 })
> show dbs
admin          0.000GB
beginnersbook  0.000GB
local          0.000GB
>
```

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

MongoDB 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. Lets see the steps to drop a database in MongoDB.

1. See the list of databases using **show dbs** command.

```
> show dbs
admin          0.000GB
beginnersbook  0.000GB
local          0.000GB
```

It is showing two default databases and one database “beginnersbook” that I have created.

2. Switch to the database that needs to be dropped by typing this command.

```
use database_name
```

For example I want to delete the database “beginnersbook”.

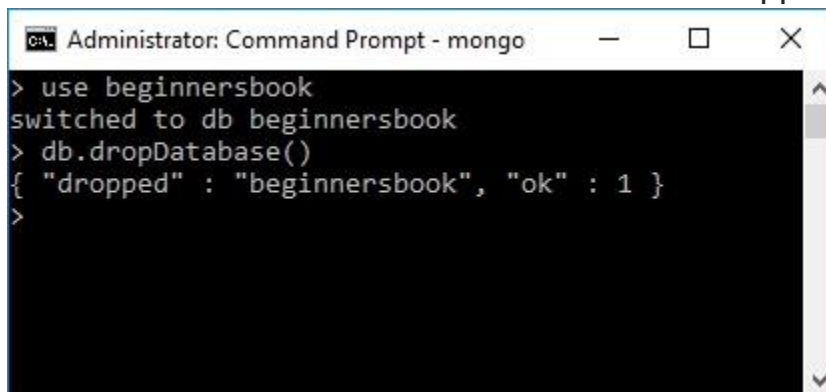
```
> use beginnersbook
switched to db beginnersbook
```

Note: Change the database name in the above command, from beginnersbook to the database that needs to be deleted.

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

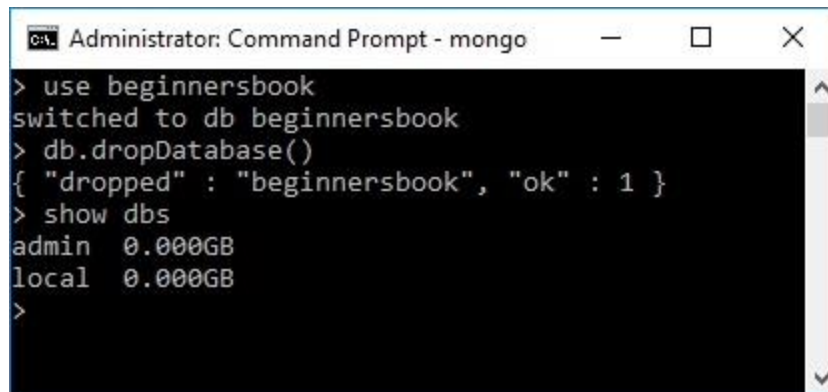
```
> db.dropDatabase()
{ "dropped" : "beginnersbook", "ok" : 1 }
```

The command executed successfully and showing the operation “dropped” and status “ok” which means that the database is dropped.



```
Administrator: Command Prompt - mongo
> use beginnersbook
switched to db beginnersbook
> db.dropDatabase()
{ "dropped" : "beginnersbook", "ok" : 1 }
>
```

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

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - mongo". The window has a black background with white text. The text shows the following sequence of commands and outputs:

```
> use beginnersbook
switched to db beginnersbook
> db.dropDatabase()
{ "dropped" : "beginnersbook", "ok" : 1 }
> show dbs
admin  0.000GB
local  0.000GB
>
```

 The output of the `show dbs` command shows two databases: `admin` and `local`, both with a size of `0.000GB`. The `beginnersbook` database is no longer listed, indicating it has been successfully dropped.

As you can see that the database "beginnersbook" is not present in the list that means it has been dropped successfully.

Conclusion : We have created and dopped database in MONGODB

3. Creating the Collection in MongoDB on the fly.

Aim: To Create a Collection in MongoDB on the fly.

Procedure:

Creating the Collection in MongoDB on the fly

The cool thing about MongoDB is that you need not to create collection before you insert document in it. With a single command you can insert a document in the collection and the MongoDB creates that collection on the fly.

Syntax: **db.collection_name.insert({key:value, key:value...})**

For example:

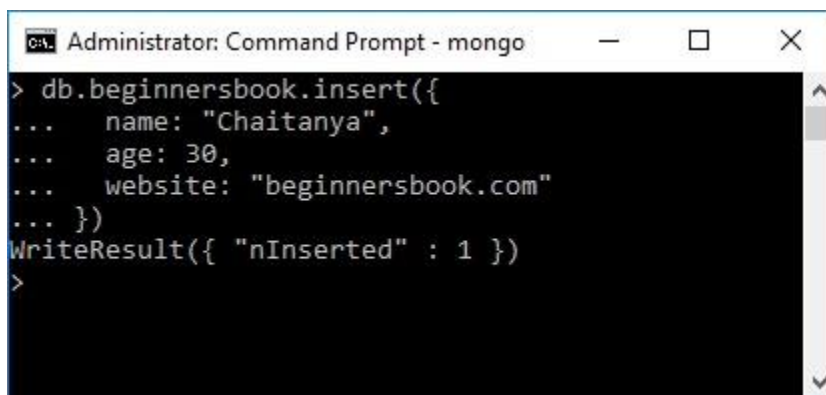
We don't have a collection `beginnersbook` in the database `beginnersbookdb`. This command will create the collection named "beginnersbook" on the fly and insert a document in it with the specified key and value pairs.

```
> use beginnersbookdb
switched to db beginnersbookdb
```

```
db.beginnersbook.insert({
  name: "Chaitanya",
  age: 30,
  website: "beginnersbook.com"
})
```

You would see this response in the command prompt.

```
WriteResult({ "nInserted" : 1 })
```

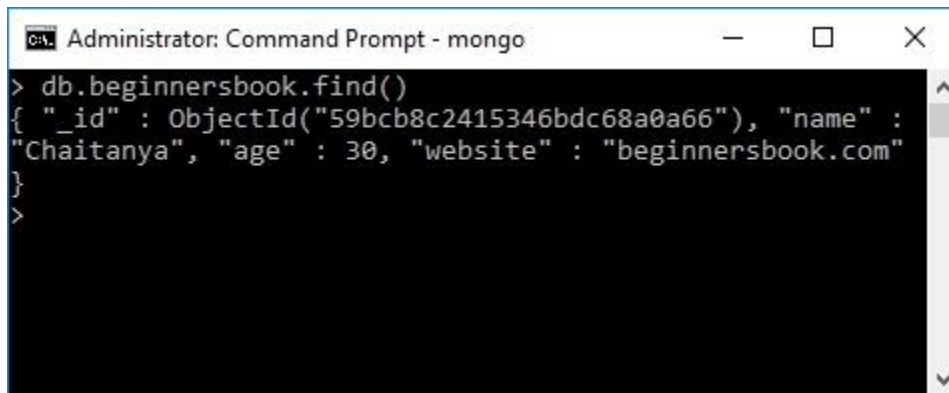


```
Administrator: Command Prompt - mongo
> db.beginnersbook.insert({
...   name: "Chaitanya",
...   age: 30,
...   website: "beginnersbook.com"
... })
WriteResult({ "nInserted" : 1 })
>
```

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()**

```
> db.beginnersbook.find()
{ "_id" : ObjectId("59bcb8c2415346bdc68a0a66"), "name" : "Chaitanya",
  "age" : 30, "website" : "beginnersbook.com" }
```



```
Administrator: Command Prompt - mongo
> db.beginnersbook.find()
{ "_id" : ObjectId("59bcb8c2415346bdc68a0a66"), "name" :
"Chaitanya", "age" : 30, "website" : "beginnersbook.com"
}
>
```

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.

```
> show collections
beginnersbook
```

Conclusion: Creating the Collection in MongoDB on the fly done

4. Creating collection with options before inserting the documents and drop the collection created.

Aim: To Create a collection with options before inserting the documents and drop the collection created.

Procedure:

Creating collection with options before inserting the documents:

We can also create collection before we actually insert data in it. This method provides you the options that you can set while creating a collection.

Syntax:

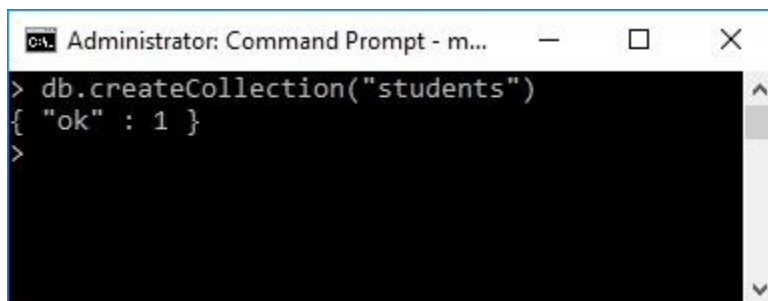
```
db.createCollection(name, options)
```

name is the collection name

and **options** is an optional field that we can use to specify certain parameters such as size, max number of documents etc. in the collection.

First lets see how this command is used for creating collection without any parameters:

```
> db.createCollection("students")
{ "ok" : 1 }
```

A screenshot of a Windows Command Prompt window. The title bar reads "Administrator: Command Prompt - m...". The command prompt shows the command `> db.createCollection("students")` being entered, followed by the output `{ "ok" : 1 }` on the next line. The prompt is currently on a new line after the output.

Lets see the options that we can provide while creating a collection:

capped: type: boolean.

This parameter takes only true and false. This specifies a cap on the max entries a collection can have. Once the collection reaches that limit, it starts overwriting old entries.

The point to note here is that when you set the capped option to true you also have to specify the size parameter.

size: type: number.

This specifies the max size of collection (capped collection) in bytes.

max: type: number.

This specifies the max number of documents a collection can hold.

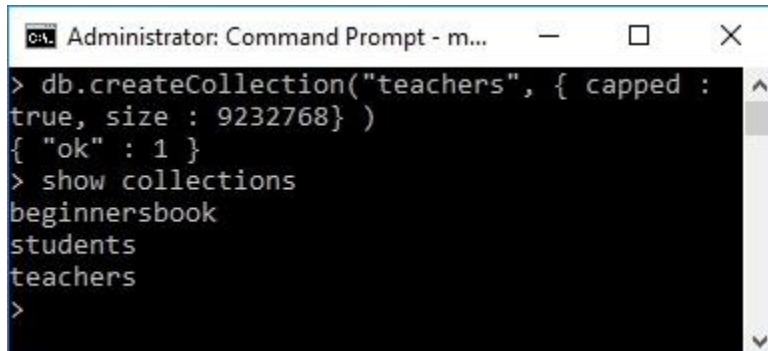
autoIndexId: type: boolean

The default value of this parameter is false. If you set it true then it automatically creates index field `_id` for each document. We will learn about index in the MongoDB indexing tutorial.

Lets see an example of capped collection:

```
db.createCollection("teachers", { capped : true, size : 9232768 } )
{ "ok" : 1 }
```

This command will create a collection named "teachers" with the max size of 9232768 bytes. Once this collection reaches that limit it will start overwriting old entries.

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - m...". The window has a black background with white text. The text shows the execution of MongoDB commands: first, `db.createCollection("teachers", { capped : true, size : 9232768 })` followed by `{ "ok" : 1 }`, and then `> show collections` which lists `beginnersbook`, `students`, and `teachers`. The prompt `>` is visible at the end of the last line.

```
Administrator: Command Prompt - m...
> db.createCollection("teachers", { capped :
true, size : 9232768} )
{ "ok" : 1 }
> show collections
beginnersbook
students
teachers
>
```

Conclusion: We have Created a collection with options before inserting the documents and dropped the collection created.

5.MongoDB insert document

- a.Insert single document
- b.Insert multiple documents in collection

Aim: To insert a Document in MONOGDB

Procedure:

MongoDB Insert Document

In this tutorial, we will see how to insert a document into the collection. We will understand this with the help of multiple examples.

Syntax to insert a document into the collection:

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

Lets take an example to understand this .

MongoDB Insert Document using insert() Example

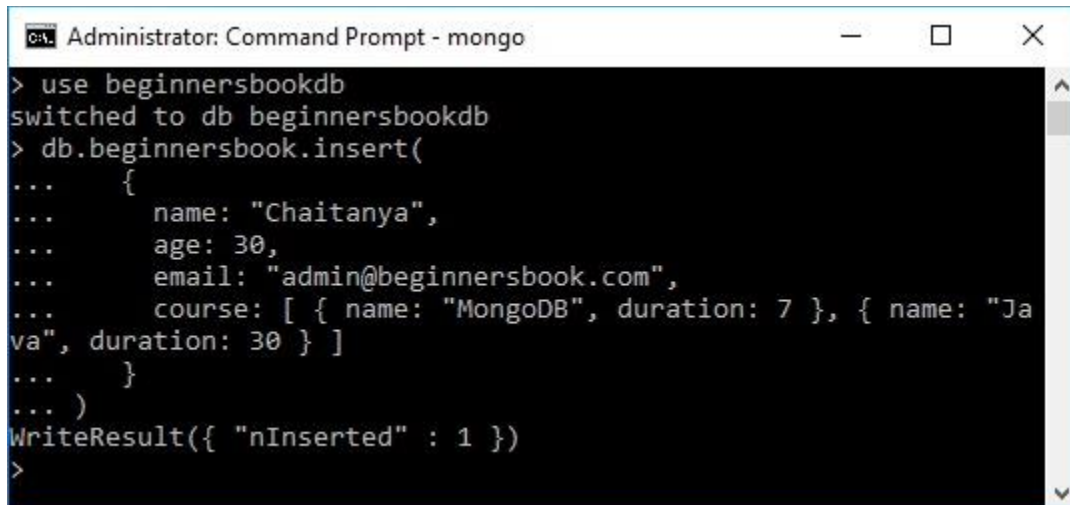
Here we are inserting a document into the collection named “beginnersbook”. The field “course” in the example below is an array that holds the several key-value pairs.

```
db.beginnersbook.insert(  
  {  
    name: "Chaitanya",  
    age: 30,  
    email: "admin@beginnersbook.com",  
    course: [ { name: "MongoDB", duration: 7 }, { name: "Java", duration: 30 } ]  
  }  
)
```

You should see a successful write message like this:

```
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



```
Administrator: Command Prompt - mongo
> use beginnersbookdb
switched to db beginnersbookdb
> db.beginnersbook.insert(
...   {
...     name: "Chaitanya",
...     age: 30,
...     email: "admin@beginnersbook.com",
...     course: [ { name: "MongoDB", duration: 7 }, { name: "Java", duration: 30 } ]
...   }
... )
WriteResult({ "nInserted" : 1 })
>
```

Verification:

You can also verify whether the document is successfully inserted by typing following command:

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

In the above example, we inserted the document in the collection named "beginnersbook" so the command should be:

```
> db.beginnersbook.find()
{ "_id" : ObjectId("59bce797668dcce02aaa6fec"), "name" : "Chaitanya", "age" : 30, "email" : "admin@beginnersbook.com", "course" : [ { "name" : "MongoDB", "duration" : 7 }, { "name" : "Java", "duration" : 30 } ] }
```

B)Inserting Multiple Documents in collection

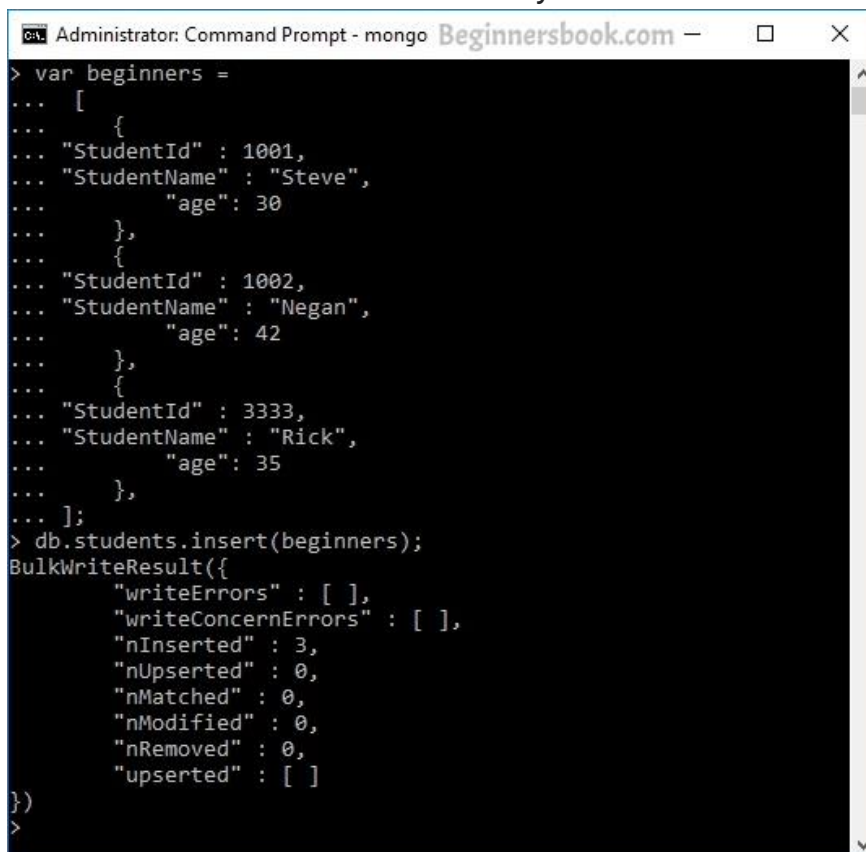
To insert multiple documents in collection, we define an array of documents and later we use the insert() method on the array variable as shown in the example below. Here we are inserting three documents in the collection named “students”. This command will insert the data in “students” collection, if the collection is not present then it will create the collection and insert these documents.

```
var beginners =  
[  
  {  
    "StudentId" : 1001,  
    "StudentName" : "Steve",  
    "age": 30  
  },  
  {  
    "StudentId" : 1002,  
    "StudentName" : "Negan",  
    "age": 42  
  },  
  {  
    "StudentId" : 3333,  
    "StudentName" : "Rick",  
    "age": 35  
  },  
];  
db.students.insert(beginners);
```

You would see this output:

```
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 3,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

As you can see that it shows number 3 in front of **nInserted**. this means that the 3 documents have been inserted by this command.



```
Administrator: Command Prompt - mongo Beginnersbook.com
> var beginners =
... [
...   {
...     "StudentId" : 1001,
...     "StudentName" : "Steve",
...     "age" : 30
...   },
...   {
...     "StudentId" : 1002,
...     "StudentName" : "Negan",
...     "age" : 42
...   },
...   {
...     "StudentId" : 3333,
...     "StudentName" : "Rick",
...     "age" : 35
...   },
... ];
> db.students.insert(beginners);
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 3,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```

Conclusion: We have inserted a document in MONGODB.

6. Querying all the documents in json format and Querying based on the criteria.

Aim: To Querying all the documents in json format and Querying based on the criteria.

Procedure:

Querying all the documents in JSON format

Lets say we have a collection `students` in a database named `beginnersbookdb`. To get all the documents we use this command:

```
db.students.find()
```

However the output we get is not in any format and less-readable. To improve the readability, we can format the output in JSON format with this command:

```
db.students.find().forEach(printjson);
```

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

```
db.students.find().pretty()
```

As you can see in the screenshot below that the documents are in JSON format


```
Administrator: Command Prompt - mongo Beginnersbook.com
> use beginnersbookdb
switched to db beginnersbookdb
> db.students.find().forEach(printjson);
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fee"),
  "StudentId" : 1002,
  "StudentName" : "Negan",
  "age" : 42
}
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fef"),
  "StudentId" : 3333,
  "StudentName" : "Rick",
  "age" : 35
}
>
```

Query Document based on the criteria

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

Equality Criteria:

For example: I want to fetch the data of "Steve" from students collection. The command for this should be:

```
db.students.find({StudentName : "Steve"}).pretty()
```

This command returns the document matching the given criteria.

```
Administrator: Command Prompt - mongo Beginnersbook.com
> db.students.find({StudentName : "Steve"}).pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
>
```

Greater Than Criteria:

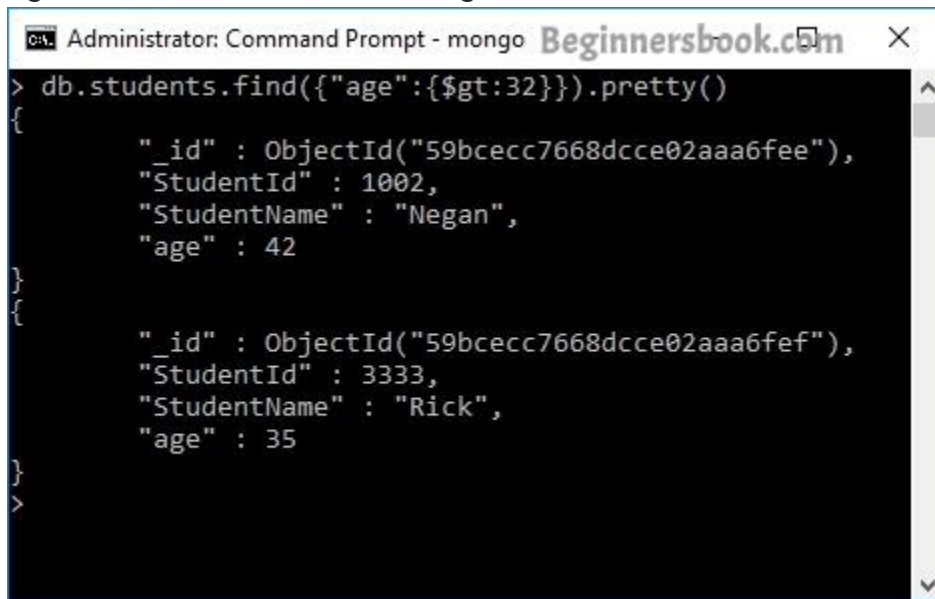
Syntax:

```
db.collection_name.find({"field_name":{"$gt:criteria_value"}}).pretty()
```

For example: I would like to fetch the details of students having age > 32 then the query should be:

```
db.students.find({"age":{"$gt:32"}}).pretty()
```

I got two documents matching the criteria as shown in the screenshot below:

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - mongo". The window shows the execution of a MongoDB query. The command entered is `> db.students.find({"age":{"$gt:32"}}).pretty()`. The output displays two JSON documents. The first document has fields: `"_id" : ObjectId("59bcecc7668dcce02aaa6fee")`, `"StudentId" : 1002`, `"StudentName" : "Negan"`, and `"age" : 42`. The second document has fields: `"_id" : ObjectId("59bcecc7668dcce02aaa6fef")`, `"StudentId" : 3333`, `"StudentName" : "Rick"`, and `"age" : 35`. A watermark "Beginnersbook.com" is visible in the top right corner of the window.

```
> db.students.find({"age":{"$gt:32"}}).pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fee"),
  "StudentId" : 1002,
  "StudentName" : "Negan",
  "age" : 42
}
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fef"),
  "StudentId" : 3333,
  "StudentName" : "Rick",
  "age" : 35
}
>
```

Less than Criteria:

Syntax:

```
db.collection_name.find({"field_name":{"$lt:criteria_value"}}).pretty()
```

Example: Find all the students having id less than 3000. The command for this criteria would be:

```
db.students.find({"StudentId":{"$lt:3000"}}).pretty()
```

Output:

```
> db.students.find({"StudentId":{"$lt:3000}}).pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fee"),
  "StudentId" : 1002,
  "StudentName" : "Negan",
  "age" : 42
}
```

Not Equals Criteria:

Syntax:

```
db.collection_name.find({"field_name":{"$ne:criteria_value}}).pretty()
```

Example: Find all the students where id is not equal to 1002. The command for this criteria would be:

```
db.students.find({"StudentId":{"$ne:1002}}).pretty()
```

Output:

```
> db.students.find({"StudentId":{"$ne:1002}}).pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fef"),
  "StudentId" : 3333,
  "StudentName" : "Rick",
  "age" : 35
}
```

Here are the other two criteria:

Greater than equals Criteria:

```
db.collection_name.find({"field_name":{"$gte:criteria_value"}}).pretty()
```

Less than equals Criteria:

```
db.collection_name.find({"field_name":{"$lte:criteria_value"}}).pretty()
```

The pretty() method that we have added at the end of all the commands is not mandatory. It is just used for formatting purposes.

Conclusion : we have successfully performed Querying all the documents in json format and Querying based on the criteria. '

7. MongoDB update document

- a. Using update() method.
- b. Using save() method

Aim: To update document in MONGODB

Procedure:

MongoDB – Update Document in a Collection

BY CHAITANYA SINGH | FILED UNDER: [MONGODB TUTORIAL](#)

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 criteria in command and the document that matches that criteria is updated. To understand how criteria works in the MongoDB commands, refer the last tutorial: [MongoDB Query Document](#)

A) Updating Document using update() method

Syntax:

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

Example:

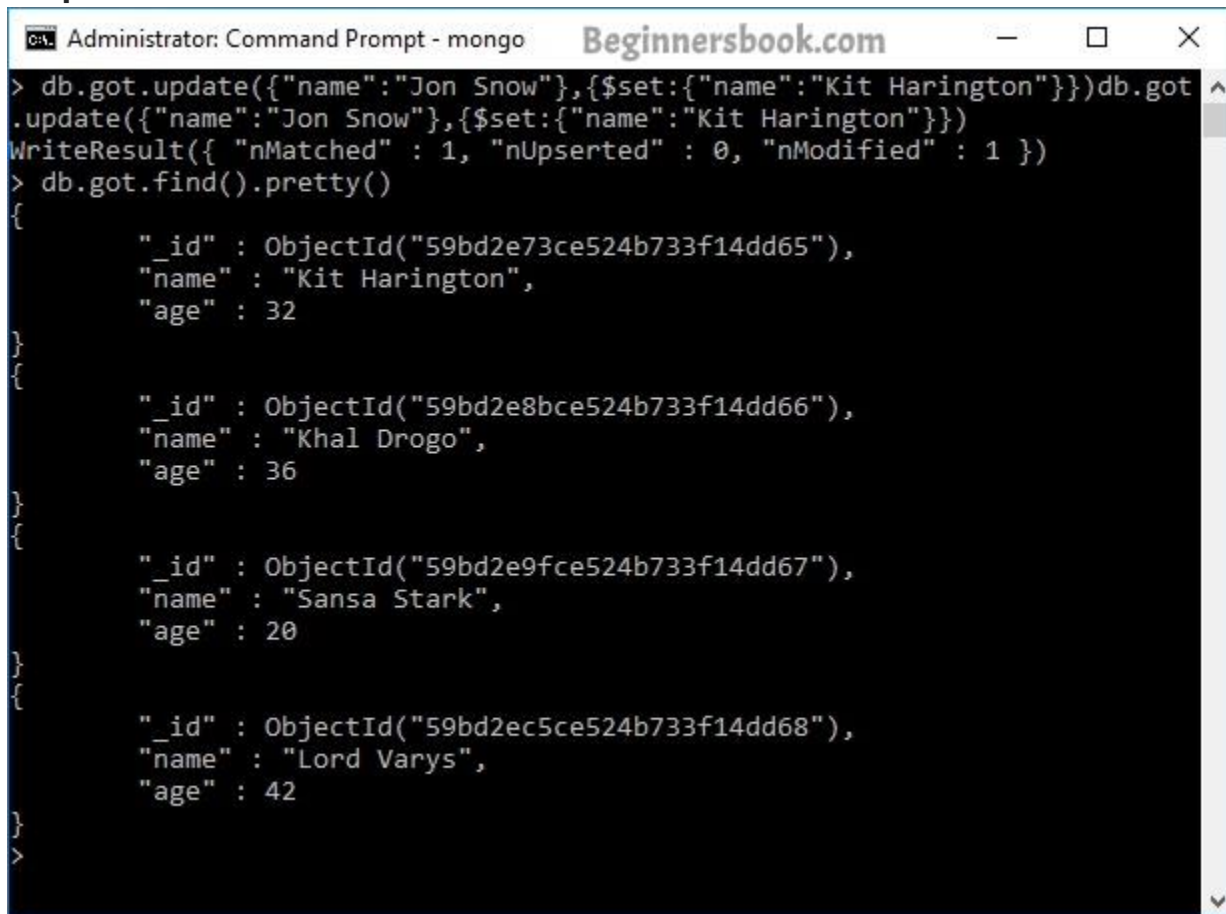
For example: Lets say I have a collection named “got” in the database “beginnersbookdb”. The documents inside “got” are:

```
> db.got.find().pretty()
{
  "_id" : ObjectId("59bd2e73ce524b733f14dd65"),
  "name" : "Jon Snow",
  "age" : 32
}
{
  "_id" : ObjectId("59bd2e8bce524b733f14dd66"),
  "name" : "Khal Drogo",
  "age" : 36
}
{
  "_id" : ObjectId("59bd2e9fce524b733f14dd67"),
  "name" : "Sansa Stark",
  "age" : 20
}
{
  "_id" : ObjectId("59bd2ec5ce524b733f14dd68"),
  "name" : "Lord Varys",
  "age" : 42
}
```

Now suppose if I want to update the name of Jon Snow with the name “Kit Harington”. The command for this would be:

```
db.got.update({"name":"Jon Snow"},{$set:{"name":"Kit Harington"}})
```

Output:



```
> db.got.update({'name':'Jon Snow'},{$set:{'name':'Kit Harington'}})db.got
.update({'name':'Jon Snow'},{$set:{'name':'Kit Harington'}})
WriteResult({'nMatched' : 1, 'nUpserted' : 0, 'nModified' : 1 })
> db.got.find().pretty()
{
  "_id" : ObjectId("59bd2e73ce524b733f14dd65"),
  "name" : "Kit Harington",
  "age" : 32
}

{
  "_id" : ObjectId("59bd2e8bce524b733f14dd66"),
  "name" : "Khal Drogo",
  "age" : 36
}

{
  "_id" : ObjectId("59bd2e9fce524b733f14dd67"),
  "name" : "Sansa Stark",
  "age" : 20
}

{
  "_id" : ObjectId("59bd2ec5ce524b733f14dd68"),
  "name" : "Lord Varys",
  "age" : 42
}
>
```

As you can see that the document has been updated.

Do you know? By default the update method updates a single document. In the above example we had only one document matching with the criteria, however if there were more then also only one document would have been updated. To enable update() method to update multiple documents you have to set “multi” parameter of this method to true as shown below.

To update multiple documents with the update() method:

```
db.got.update({'name':'Jon Snow'},
  {$set:{'name':'Kit Harington'}},{multi:true})
```

Updating Document using save() method

Syntax:

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


Lets take the same example that we have seen above. Now we want to update the name of “Kit Harington” to “Jon Snow”. To work with save() method you should know the unique _id field of that document.

A very important **point to note** is that when you 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

To get the _id of a document, you can either type this command:

```
db.got.find().pretty()
```

Here got is a collection name. This method of finding unique _id is only useful when you have few documents, otherwise scrolling and searching for that _id in huge number of documents is tedious.

If you have huge number of documents then, to get the _id of a particular document use the criteria in find() method. For example: To get the _id of the document that we want to update using save() method, type this command:

```
> db.got.find({"name": "Kit Harington"}).pretty()
{
  "_id" : ObjectId("59bd2e73ce524b733f14dd65"),
  "name" : "Kit Harington",
  "age" : 32
}
```

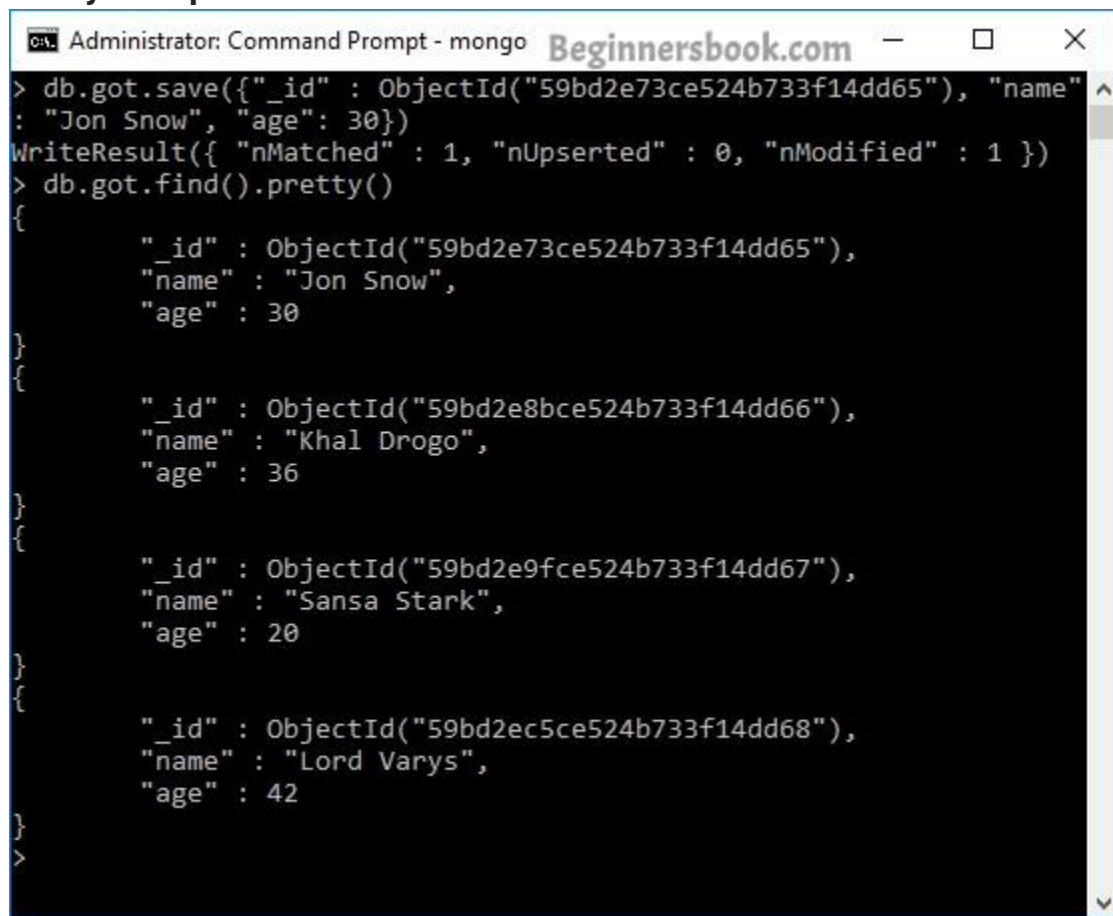
Now we know the unique _id field of that document. Lets write the command using save() method.

```
> db.got.save({"_id" : ObjectId("59bd2e73ce524b733f14dd65"), "name":
  "Jon Snow", "age": 30})
>
```

You should see this output:

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Verify the update:



```
Administrator: Command Prompt - mongo Beginnersbook.com
> db.got.save({"_id" : ObjectId("59bd2e73ce524b733f14dd65"), "name" : "Jon Snow", "age": 30})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.got.find().pretty()
{
  "_id" : ObjectId("59bd2e73ce524b733f14dd65"),
  "name" : "Jon Snow",
  "age" : 30
}

{
  "_id" : ObjectId("59bd2e8bce524b733f14dd66"),
  "name" : "Khal Drogo",
  "age" : 36
}

{
  "_id" : ObjectId("59bd2e9fce524b733f14dd67"),
  "name" : "Sansa Stark",
  "age" : 20
}

{
  "_id" : ObjectId("59bd2ec5ce524b733f14dd68"),
  "name" : "Lord Varys",
  "age" : 42
}
```

Conclusion: We have successfully updated a document

8. MongoDB delete document from a collection .

a. Using remove() method.

b. Remove only one document matching your criteria

c. Remove all documents

Aim: To delete a document from a collection

Procedure:

A) MongoDB Delete Document from a Collection

BY CHAITANYA SINGH | FILED UNDER: [MONGODB TUTORIAL](#)

In this tutorial we will learn how to delete documents 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)
```

To understand how to specify criteria in MongoDB commands, refer this tutorial: [MongoDB Query Document](#).

Delete Document using remove() method

Lets say I have a collection students in my MongoDB database named beginnersbookdb. The documents in students collection are:

```
> db.students.find().pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
```

```
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fef"),
  "StudentId" : 3333,
  "StudentName" : "Rick",
  "age" : 35
}
```

Now I want to remove the student from this collection who has a student id equal to 3333. To do this I would write a command using remove() method like this:

```
db.students.remove({"StudentId": 3333})
```

Output:

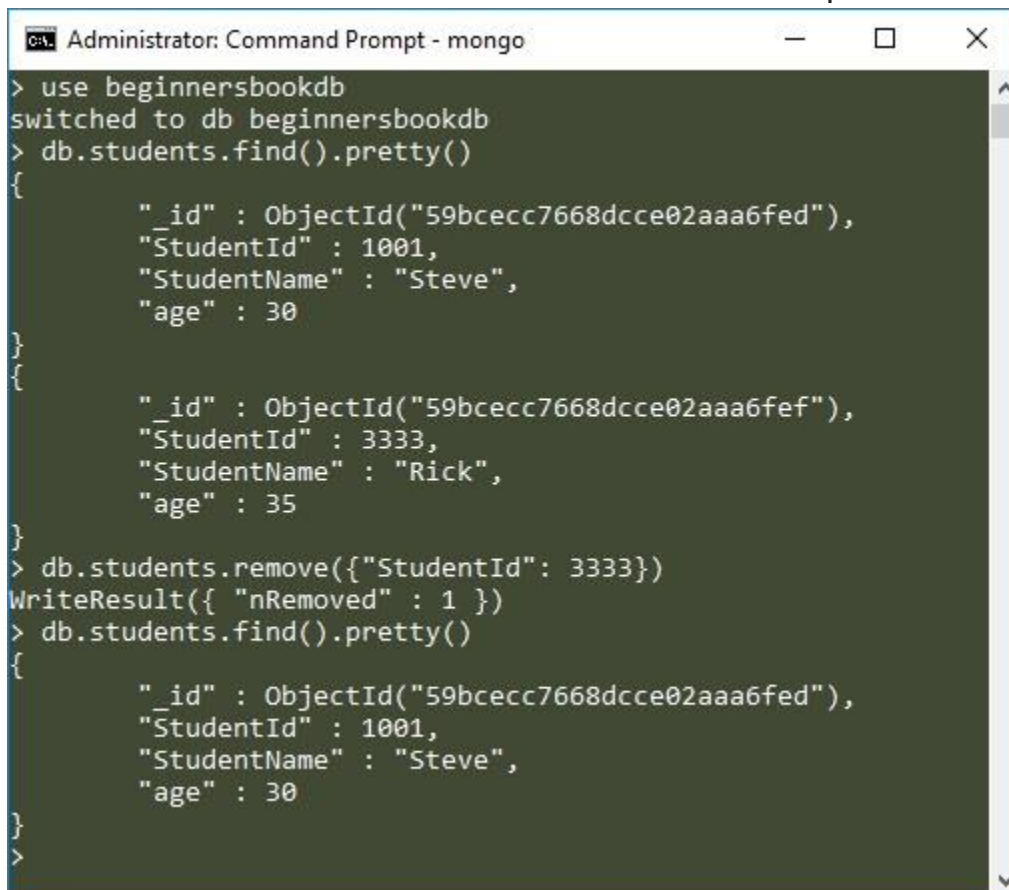
```
WriteResult({ "nRemoved" : 1 })
```

To verify whether the document is actually deleted. Type the following command:

```
db.students.find().pretty()
```

It will list all the documents of students collection.

Here is the screenshot of all the above mentioned steps:



```
Administrator: Command Prompt - mongo
> use beginnersbookdb
switched to db beginnersbookdb
> db.students.find().pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fef"),
  "StudentId" : 3333,
  "StudentName" : "Rick",
  "age" : 35
}
> db.students.remove({"StudentId": 3333})
WriteResult({ "nRemoved" : 1 })
> db.students.find().pretty()
{
  "_id" : ObjectId("59bcecc7668dcce02aaa6fed"),
  "StudentId" : 1001,
  "StudentName" : "Steve",
  "age" : 30
}
>
```

B)How to remove only one document matching your criteria?

When there are more than one documents present in collection that matches the criteria then all those documents will be deleted if you run the remove command. However there is a way to limit the deletion to only one document so that even if there are more documents matching the deletion criteria, only one document will be deleted.

```
db.collection_name.remove(delete_criteria, justOne)
```

Here justOne is a Boolean parameter that takes only 1 and 0, if you give 1 then it will limit the the document deletion to only 1 document. This is an optional parameters as we have seen above that we have used the remove() method without using this parameter.

For example I have the following records in collection.

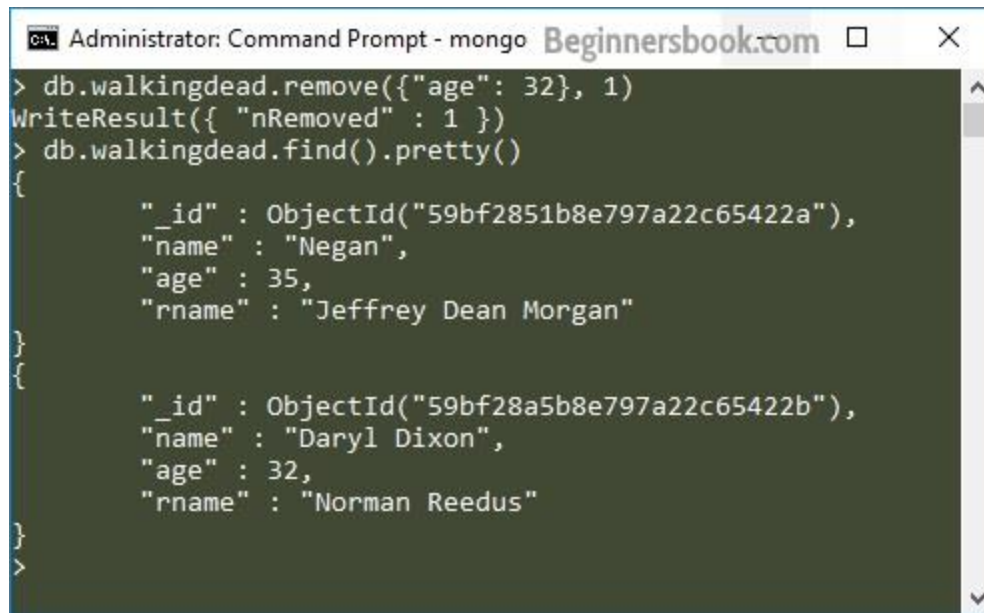
```
> db.walkingdead.find().pretty()
{
  "_id" : ObjectId("59bf280cb8e797a22c654229"),
  "name" : "Rick Grimes",
  "age" : 32,
  "rname" : "Andrew Lincoln"
}
{
  "_id" : ObjectId("59bf2851b8e797a22c65422a"),
  "name" : "Negan",
  "age" : 35,
  "rname" : "Jeffrey Dean Morgan"
}
{
  "_id" : ObjectId("59bf28a5b8e797a22c65422b"),
  "name" : "Daryl Dixon",
  "age" : 32,
  "rname" : "Norman Reedus"
}
```

Lets say I want to remove the document that has age equal to 32. There are two documents in this collection that are matching this criteria. However to limit the deletion to one we are setting justOne parameter to true.

```
db.walkingdead.remove({"age": 32}, 1)
```

Output: As you can see only one document got deleted.

```
WriteResult({ "nRemoved" : 1 })
```



```
Administrator: Command Prompt - mongo Beginnersbook.com
> db.walkingdead.remove({"age": 32}, 1)
WriteResult({ "nRemoved" : 1 })
> db.walkingdead.find().pretty()
{
  "_id" : ObjectId("59bf2851b8e797a22c65422a"),
  "name" : "Negan",
  "age" : 35,
  "rname" : "Jeffrey Dean Morgan"
}
{
  "_id" : ObjectId("59bf28a5b8e797a22c65422b"),
  "name" : "Daryl Dixon",
  "age" : 32,
  "rname" : "Norman Reedus"
}
>
```

C)Remove all Documents

If you want to remove all the documents from a collection but does not want to remove the collection itself then you can use remove() method like this:

```
db.collection_name.remove({})
```

Conclusion: We have deleted the document from Collection

9. MongoDB Projection

Aim: To Perform projection in MONGO DB

Procedure :

MongoDB Projection

In this tutorial, we will learn another interesting topic of MongoDB which is **MongoDB Projection**. This is used when we want to get the selected fields of the documents rather than all fields.

For example, we have a collection where we have stored documents that have the fields: student_name, student_id, student_age but we want to see only the student_id of all the students then in that case we can use projection to get only the student_id.

Syntax:

```
db.collection_name.find({}, {field_key: 1 or 0})
```

Do not worry about the syntax now, we will see an example to understand this.

MongoDB Projection Example

Lets take an example to understand the Projection in MongoDB. We have a collection named studentdata that has following documents.

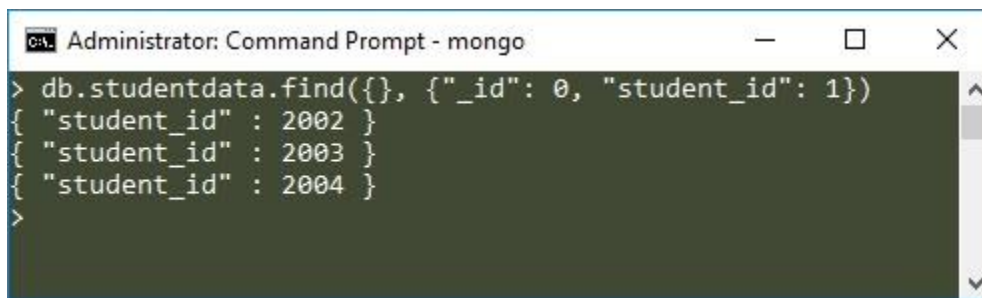
```
> db.studentdata.find().pretty()
{
  "_id" : ObjectId("59bf63380be1d7770c3982af"),
  "student_name" : "Steve",
  "student_id" : 2002,
  "student_age" : 22
}
{
  "_id" : ObjectId("59bf63500be1d7770c3982b0"),
  "student_name" : "Carol",
  "student_id" : 2003,
  "student_age" : 22
}
```



```
{
  "_id" : ObjectId("59bf63650be1d7770c3982b1"),
  "student_name" : "Tim",
  "student_id" : 2004,
  "student_age" : 23
}
```

To get only the student_id for all the documents, we will use the Projection like this:

```
> db.studentdata.find({}, {"_id": 0, "student_id": 1})
{ "student_id" : 2002 }
{ "student_id" : 2003 }
{ "student_id" : 2004 }
```



```
Administrator: Command Prompt - mongo
> db.studentdata.find({}, {"_id": 0, "student_id": 1})
{ "student_id" : 2002 }
{ "student_id" : 2003 }
{ "student_id" : 2004 }
```

Value 1 means show that field and 0 means do not show that field. When we set a field to 1 in Projection other fields are automatically set to 0, except _id, so to avoid the _id we need to specifically set it to 0 in projection. The vice versa is also true when we set few fields to 0, other fields set to 1 automatically(see the below example)

Another way of doing the same thing:

```
> db.studentdata.find({}, {"_id": 0, "student_name": 0, "student_age": 0})
{ "student_id" : 2002 }
{ "student_id" : 2003 }
{ "student_id" : 2004 }
```

Important Note:

Some of you may get this error while using Projection in Query:

```
Error: error: {
  "ok" : 0,
  "errmsg" : "Projection cannot have a mix of inclusion and exclusion.",
  "code" : 2,
  "codeName" : "BadValue"
}
```

This happens when you set some fields to 0 and other to 1, in other words you mix inclusion and exclusion, the only exception is the `_id` field. for example: The following Query would produce this error:

```
db.studentdata.find({}, {"_id": 0, "student_name": 0, "student_age": 1})
```

This is because we have set `student_name` to 0 and other field `student_age` to 1. We can't mix these. You either set those fields that you don't want to display to 0 or set the fields to 1 that you want to display.

Conclusion : WE have performed MONGO DB PROJECTION

10. limit(),skip(),sort()methods in MongoDB

Aim: to perform limit(),skip(),sort()methods in MongoDB

Procedure:

The limit() method in MongoDB

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

Syntax:

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

Lets take an example to understand how to use this method. Lets say, I have a collection studentdata which has following documents:

```
> db.studentdata.find().pretty()
{
  "_id" : ObjectId("59bf63380be1d7770c3982af"),
  "student_name" : "Steve",
  "student_id" : 2002,
  "student_age" : 22
}
{
  "_id" : ObjectId("59bf63500be1d7770c3982b0"),
  "student_name" : "Carol",
  "student_id" : 2003,
  "student_age" : 22
}
{
  "_id" : ObjectId("59bf63650be1d7770c3982b1"),
  "student_name" : "Tim",
  "student_id" : 2004,
  "student_age" : 23
}
```

Lets say I want to find out the list of all the students, having the id > 2002. I would write a query like this using a criteria:

To learn how to specify a criteria while querying documents, read this: [MongoDB Query Document](#)

```
db.studentdata.find({student_id : {$gt:2002}}).pretty()
```



```
Administrator: Command Prompt - mongo Beginnersbook.com
> db.studentdata.find({student_id : {$gt:2002}}).pretty()
{
  "_id" : ObjectId("59bf63500be1d7770c3982b0"),
  "student_name" : "Carol",
  "student_id" : 2003,
  "student_age" : 22
}
{
  "_id" : ObjectId("59bf63650be1d7770c3982b1"),
  "student_name" : "Tim",
  "student_id" : 2004,
  "student_age" : 23
}
>
```

Using limit() method to limit the documents in the result:

Lets say I do not want all the documents matching the criteria. I want only selected number of documents then I can use limit() method to limit the number of documents. For example, if I want only one document in output then I would do this:

```
> db.studentdata.find({student_id : {$gt:2002}}).limit(1).pretty()
{
  "_id" : ObjectId("59bf63500be1d7770c3982b0"),
  "student_name" : "Carol",
  "student_id" : 2003,
  "student_age" : 22
}
```

MongoDB Skip() Method

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

To understand the use of skip() method, let's take the same example that we have seen above. In the above example we can see that by using limit(1) we managed to get only one document, which is the first document that matched the given criteria. What if you do not want the first document matching your criteria. For example we have two documents that have student_id value greater than 2002 but when we limited the result to 1 by using limit(1), we got the first document, in order to get the second document matching this criteria we can use skip(1) here which will skip the first document.

Without using skip():

```
> db.studentdata.find({student_id : {$gt:2002}}).limit(1).pretty()
{
  "_id" : ObjectId("59bf63500be1d7770c3982b0"),
  "student_name" : "Carol",
  "student_id" : 2003,
  "student_age" : 22
}
```

Using skip:

```
> db.studentdata.find({student_id : {$gt:2002}}).limit(1).skip(1).pretty()
{
  "_id" : ObjectId("59bf63650be1d7770c3982b1"),
  "student_name" : "Tim",
  "student_id" : 2004,
  "student_age" : 23
}
```

Sorting Documents using sort() method

Using sort() method, you can sort the documents in ascending or descending order based on a particular field of document.

Syntax of sort() method:

```
db.collection_name.find().sort({field_key:1 or -1})
```

1 is for ascending order and -1 is for descending order. The default value is 1.

For example: collection studentdata contains following documents:

```
> db.studentdata.find().pretty()
{
  "_id" : ObjectId("59bf63380be1d7770c3982af"),
  "student_name" : "Steve",
  "student_id" : 2002,
  "student_age" : 22
}
{
  "_id" : ObjectId("59bf63500be1d7770c3982b0"),
  "student_name" : "Carol",
  "student_id" : 2003,
  "student_age" : 22
}
{
  "_id" : ObjectId("59bf63650be1d7770c3982b1"),
  "student_name" : "Tim",
  "student_id" : 2004,
  "student_age" : 23
}
```

Lets say I want to display the student_id of all the documents in **descending order**:

To display only a particular field of document, I am using [MongoDB Projection](#)

```
> db.studentdata.find({}, {"student_id": 1, _id:0}).sort({"student_id": -1})
{ "student_id" : 2004 }
{ "student_id" : 2003 }
{ "student_id" : 2002 }
```

To display the student_id field of all the students in **ascending order**:

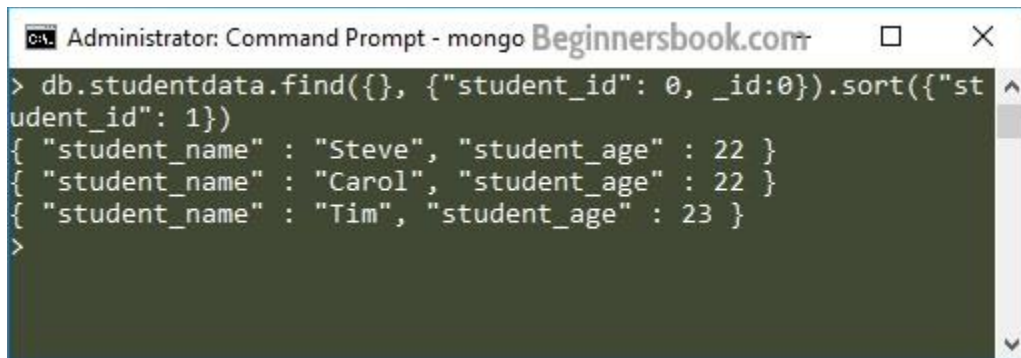
```
> db.studentdata.find({}, {"student_id": 1, _id:0}).sort({"student_id": 1})
{ "student_id" : 2002 }
{ "student_id" : 2003 }
{ "student_id" : 2004 }
```

Default: The default is ascending order so If I don't provide any value in the sort() method then it will sort the records in ascending order as shown below:

```
> db.studentdata.find({}, {"student_id": 1, _id:0}).sort({})
{ "student_id" : 2002 }
```

```
{ "student_id" : 2003 }  
{ "student_id" : 2004 }
```

You can also sort the documents based on the field that you don't want to display: For example, you can sort the documents based on student_id and display the student_age and student_name fields.



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - mongo" with a watermark for "Beginnersbook.com". The prompt contains the following text:

```
> db.studentdata.find({}, {"student_id": 0, "_id": 0}).sort({"student_id": 1})  
{ "student_name" : "Steve", "student_age" : 22 }  
{ "student_name" : "Carol", "student_age" : 22 }  
{ "student_name" : "Tim", "student_age" : 23 }  
>
```

Conclusion : We have performed limit(),skip(),sort() methods in MongoDB

11.MongoDB indexing

- a.Create index in Mongo DB
- b.Finding the indexes in a collection
- c.Drop indexes in a collection
- d.Drop all the indexes

Aim: To perform indexing in MONGO DB

Procedure:

MongoDB Indexing

An **index** in MongoDB is a special data structure that holds the data of few fields of documents on which the index is created. Indexes improve the speed of search operations in database because instead of searching the whole document, the search is performed on the indexes that holds only few fields. On the other hand, having too many indexes can hamper the performance of insert, update and delete operations because of the additional write and additional data space used by indexes.

A)How to create index in MongoDB

Syntax:

```
db.collection_name.createIndex({field_name: 1 or -1})
```

The value 1 is for ascending order and -1 is for descending order.

For example, I have a collection studentdata. The documents inside this collection have following fields:

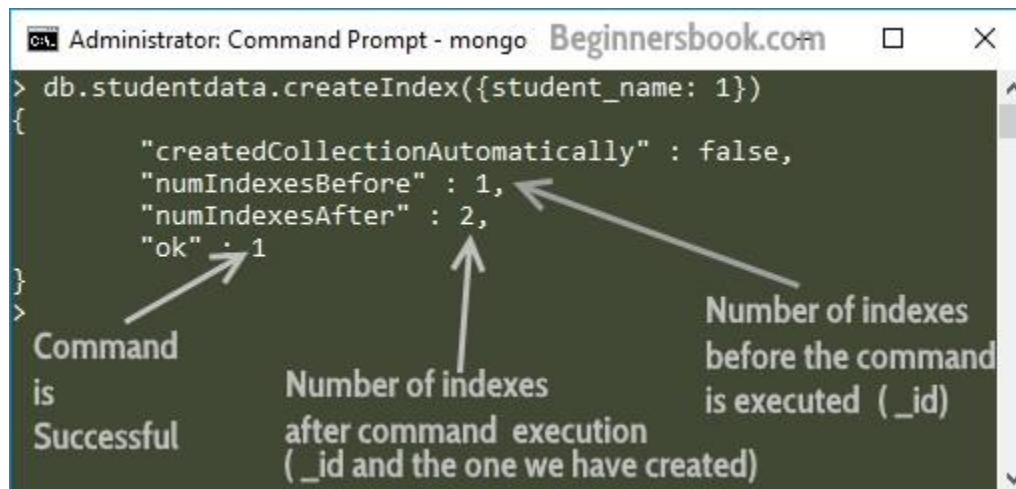
student_name, student_id and student_age

Lets say I want to create the index on student_name field in ascending order:

```
db.studentdata.createIndex({student_name: 1})
```

Output:

```
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```



We have created the index on student_name which means when someone searches the document based on the student_name, the search will be faster because the index will be used for this search. So this is important to create the index on the field that will be frequently searched in a collection.

B) Finding the indexes in a collection

We can use getIndexes() method to find all the indexes created on a collection. The syntax for this method is:

```
db.collection_name.getIndexes()
```

So to get the indexes of studentdata collection, the command would be:

```
> db.studentdata.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
  },
]
```

```

        "ns" : "test.studentdata"
    },
    {
        "v" : 2,
        "key" : {
            "student_name" : 1
        },
        "name" : "student_name_1",
        "ns" : "test.studentdata"
    }
]

```

The output shows that we have two indexes in this collection. The default index created on `_id` and the index that we have created on `student_name` field.

C) Drop indexes in a collection

You can either drop a particular index or all the indexes.

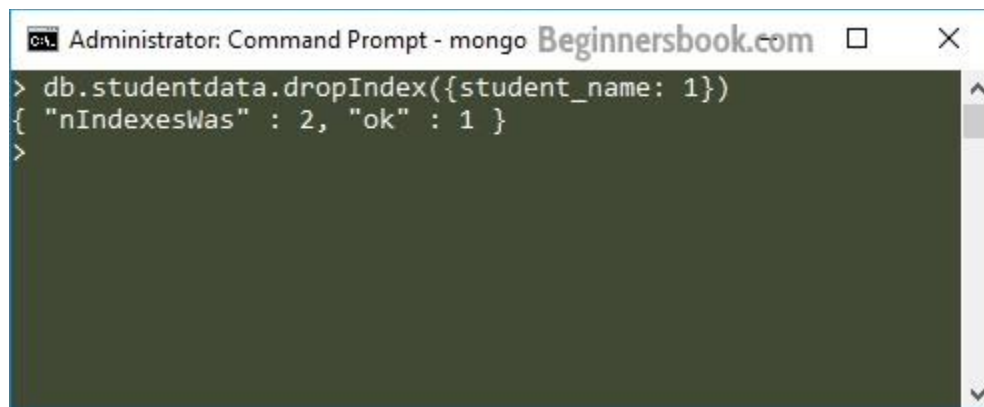
Dropping a specific index:

For this purpose the `dropIndex()` method is used.

```
db.collection_name.dropIndex({index_name: 1})
```

Lets drop the index that we have created on `student_name` field in the collection `studentdata`. The command for this:

```
db.studentdata.dropIndex({student_name: 1})
```



```

Administrator: Command Prompt - mongo Beginnersbook.com
> db.studentdata.dropIndex({student_name: 1})
{ "nIndexesWas" : 2, "ok" : 1 }
>

```

`nIndexesWas`: It shows how many indexes were there before this command got executed

`ok: 1`: This means the command is executed successfully.

d) Dropping all the indexes:

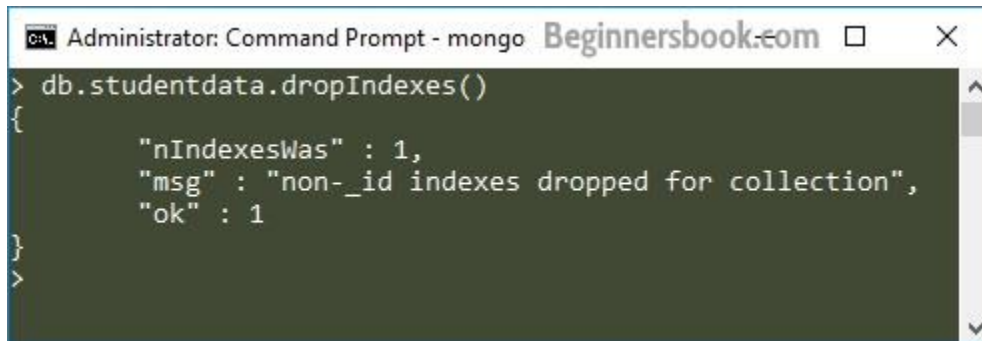
To drop all the indexes of a collection, we use `dropIndexes()` method.

Syntax of `dropIndexes()` method:

```
db.collection_name.dropIndexes()
```

Lets say we want to drop all the indexes of `studentdata` collection.

```
db.studentdata.dropIndexes()
```

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - mongo". The window has a dark background and a light blue border. The text "Beginnersbook.com" is visible in the top right corner. The command prompt shows the execution of the command `db.studentdata.dropIndexes()`. The output is a JSON object:

```
{  "nIndexesWas" : 1,  "msg" : "non-_id indexes dropped for collection",  "ok" : 1}
```

. The prompt is currently at the `>` line.

The message “non-_id indexes dropped for collection” indicates that the default index `_id` will still remain and cannot be dropped. This means that using this method we can only drop indexes that we have created, we can’t drop the default index created on `_id` field.

Conclusion : We have performed Indexing in MONGODB