

MongoDB Basics

MongoDB is completely based on JavaScript, all JavaScript syntax and coding is accepted in MongoDB.

a. Write a MongoDB query to create and drop database.

Open “cmd” and type “mongo” and click enter.

```
Command Prompt - mongo
C:\Users\Welcome>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongod
Implicit session: session { "id" : UUID("cd1ffe25-9777-4849-8631-9698118316c2") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-07-24T00:16:59.960-0700 I CONTROL [initandlisten]
2019-07-24T00:16:59.960-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-07-24T00:16:59.960-0700 I CONTROL [initandlisten] **           Read and write access to data and configuration is unrestricted.
2019-07-24T00:16:59.960-0700 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

You can get list of all databases in mongoDB by following command.

>show dbs;

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

For creating database we have to use following command.

```
> use itvoyagersDB
switched to db itvoyagersDB
>use itvoyagersDB
```

In above command “**use**” keyword has been use to create database with name “**itvoyagersDB**”.

Note: If database is not present in mongoDB then “**use**” keyword will create new database, and if database is present then “**use**” keyword will switch us to that database.

Just type **db** and click enter it will show you the current database i.e.

itvoyagersDB.

```
> db
itvoyagersDB
```

And now if we check the list of database using “**show dbs**”. We can see that our newly created database is not in the list. This is because we have to insert at least one data in our database.

```
> use itvoyagersDB
switched to db itvoyagersDB
>
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
```

For that type following command.

```
> itv = {website : "itvoyagres.in"}
{ "website" : "itvoyagres.in" }
```

Once we created data now it's time to insert it in our database. For that type following command.

```
Command Prompt - mongo
> itv = {website : "itvoyagres.in"}
{ "website" : "itvoyagres.in" }
>
> db.itvoyagersDB.insert(itv);
WriteResult({ "nInserted" : 1 })
```

Now our data has been inserted in our database, type “**show dbs**” to check it.

```
Command Prompt - mongo
> itv = {website : "itvoyagres.in"}
{ "website" : "itvoyagres.in" }
>
> db.itvoyagersDB.insert(itv);
WriteResult({ "nInserted" : 1 })
>
> show dbs
admin    0.000GB
config   0.000GB
itvoyagersDB 0.000GB
local    0.000GB
>
```

Now to drop a database use following command.

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

You can check database list with help of “show dbs;” command.

```
Command Prompt - mongo  
> show dbs  
admin          0.000GB  
config         0.000GB  
itvoyagersDB   0.000GB  
local          0.000GB  
>  
> db.dropDatabase()  
{ "dropped" : "itvoyagersDB", "ok" : 1 }  
>  
> show dbs  
admin          0.000GB  
config         0.000GB  
local          0.000GB  
>
```

b. Write a MongoDB query to create, display and drop collection.

In SQL Databases like MySQL, Oracle, etc. we save tables in database but mongoDB in Non-SQL database here if we have to store data then we have to store it in {key : value} pair, and this is called as **Collection**, and the data which is stored in collections is called as **document** (like a row in table of SQL database).

Create

Now create database.

```
> use itvoyagersDB  
switched to db itvoyagersDB
```

To create collection type following commands.

```
> use itvoyagersDB  
switched to db itvoyagersDB  
>  
> db.pages.insert({ website : "itvoyagers.in", socialmedia : ["facebook","instagram","twitter"] })  
WriteResult({ "nInserted" : 1 })  
>
```

In above example

db stands for database we are currently using.

pages is name of collection and

insert is the process.

To view collections use **“show”** command.

```
> show collections
pages
```

We can create collection with following command as well

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

Here create Collection method will create collection name **“social”** in database. And to insert items in **“social”** collection we can follow following commands.

```
> db.social.insert({ message : "please visit itvoyagers.in" })
WriteResult({ "nInserted" : 1 })
```

Display

To display all data from collection we use **find()** method.

```
> db.pages.find()
{ "_id" : ObjectId("5d39c3480c2cb5245878ede2"), "website" : "itvoyagers.in", "socialmedia" : [ "facebook", "instagram", "twitter" ] }
```

and

```
> db.social.find()
{ "_id" : ObjectId("5d3ea189c003d24866ba4d32"), "message" : "please visit itvoyagers.in" }
```

If we want to display this this is more readable format we can use following commands.

```
> db.social.find().forEach(printjson)
{
  "_id" : ObjectId("5d3ea189c003d24866ba4d32"),
  "message" : "please visit itvoyagers.in"
}
```

We can also use **pretty()**.

```
> db.social.find().pretty()
{
  "_id" : ObjectId("5d3ea189c003d24866ba4d32"),
  "message" : "please visit itvoyagers.in"
}
```

We don't have to add **“_id”** key it will be added by default and it will be unique for every document. We can add our own **“_id”**.

Drop

To delete collection from database.

```
> use itvoyagersDB
switched to db itvoyagersDB
>
> db.pages.find()
{ "_id" : ObjectId("5d39c3480c2cb5245878ede2"), "website" : "itvoyagers.in", "socialmedia" : [ "facebook", "instagram", "twitter" ] }
>
> show collections
pages
social
>
> db.pages.drop()
true
>
```

drop() will return true once it drop the collection. Now if we check collections in database we will find that “**pages**” collection has been dropped.

```
> db.pages.drop()
true
>
> show collections
social
```

c. Write a MongoDB query to insert, query, update and delete a document.

Insert Query

There are few methods for inserting data (document) in the collection.

First method

```
> show collections
social
>
> db.itv.insert({website:"itvoyagers.in"})
WriteResult({ "nInserted" : 1 })
>
> show collections
itv
social
```

Here **db** is instance of database, **itv** is the collections and **insert()** helps to insert document in collection. Above command will create collection name **itv** in itvoyagersDB.

We can also create **document** first and then insert it into **collection**, for this follow below steps.

```
> subjects = {tyit:["NGT","BI","GIS"], tycs:["cloud computing","data science"]}
{
  "tyit" : [
    "NGT",
    "BI",
    "GIS"
  ],
  "tycs" : [
    "cloud computing",
    "data science"
  ]
}
```

As we know MongoDB is based on JavaScript so it accepts all JavaScript syntax and coding.

The above variable **subjects** is nothing but the object in JavaScript so we can create this variable using “**var**” keyword as well, and it stores the values in **key : value** pair and this is nothing but the document in MongoDB. Now it's time to insert **subjects** document in **itvsubjects** collection in database.

```
> subjects = {tyit:["NGT","BI","GIS"], tycs:["cloud computing","data science"]}
{
  "tyit" : [
    "NGT",
    "BI",
    "GIS"
  ],
  "tycs" : [
    "cloud computing",
    "data science"
  ]
}
>
> db.itvsubjects.insert(subjects)
WriteResult({ "nInserted" : 1 })
```

Now check collections in database.

```
Command Prompt - mongo
> subjects = {tyit:["NGT","BI","GIS"], tycs:["cloud computing","data science"]}
{
  "tyit" : [
    "NGT",
    "BI",
    "GIS"
  ],
  "tycs" : [
    "cloud computing",
    "data science"
  ]
}
>
> db.itvsubjects.insert(subjects)
WriteResult({ "nInserted" : 1 })
>
> show collections
itv
itvsubjects
social
>
```

Here subjects is document in **itvsubjects** and **itvsubjects** is collection in database.

We can also create object variable and insert data in it afterward just like JavaScript.

```
> var info = {};
>
```

Here **info** is empty object. Let's enter elements in it.

```
> var info = {};
>
> info.name = "ITVoyagers";
ITVoyagers
> info.about = "ITVoyagers is a informative blog for IT and CS studentes";
ITVoyagers is a informative blog for IT and CS studentes
>
```

Here name and about are the key in info objects, type info and enter and it will display complete document (object).

```
> var info = {};
>
> info.name = "ITVoyagers";
ITVoyagers
> info.about = "ITVoyagers is a informative blog for IT and CS studentes";
ITVoyagers is a informative blog for IT and CS studentes
>
> info
{
  "name" : "ITVoyagers",
  "about" : "ITVoyagers is a informative blog for IT and CS studentes"
}
```

To enter this document in **itvinfo** collection type following command.

```
> var info = {};
>
> info.name = "ITVoyagers";
ITVoyagers
> info.about = "ITVoyagers is a informative blog for IT and CS studentes";
ITVoyagers is a informative blog for IT and CS studentes
>
> info
{
  "name" : "ITVoyagers",
  "about" : "ITVoyagers is a informative blog for IT and CS studentes"
}
> db.itvinfo.insert(info)
WriteResult({ "nInserted" : 1 })
```

Another way to create and insert Collection in database is to create collection first and then insert document in it.

For this follow below commands.

```
> db.createCollection("mediaplatforms")
{ "ok" : 1 }
>
> db.mediaplatforms.insert({platforms:["facebook","twitter","instagram"]})
WriteResult({ "nInserted" : 1 })
```

Here "**mediaplatform**" is the collection.

```
> show collections
itv
itvsubjects
mediaplatforms
social
```

We can insert multiple documents in one collection at same time, this is nothing but inserting array of document in collection.

```
> db.editor.insert([ { name : "vijay", dept : "IT"}, { name : "Gauri", dept : "CS"}])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

To view data from editor collection.

```
> db.editor.find().forEach(printjson)
{
  "_id" : ObjectId("5d3eb572c003d24866ba4d35"),
  "name" : "vijay",
  "dept" : "IT"
}
{
  "_id" : ObjectId("5d3eb572c003d24866ba4d36"),
  "name" : "Gauri",
  "dept" : "CS"
}
```

Queries

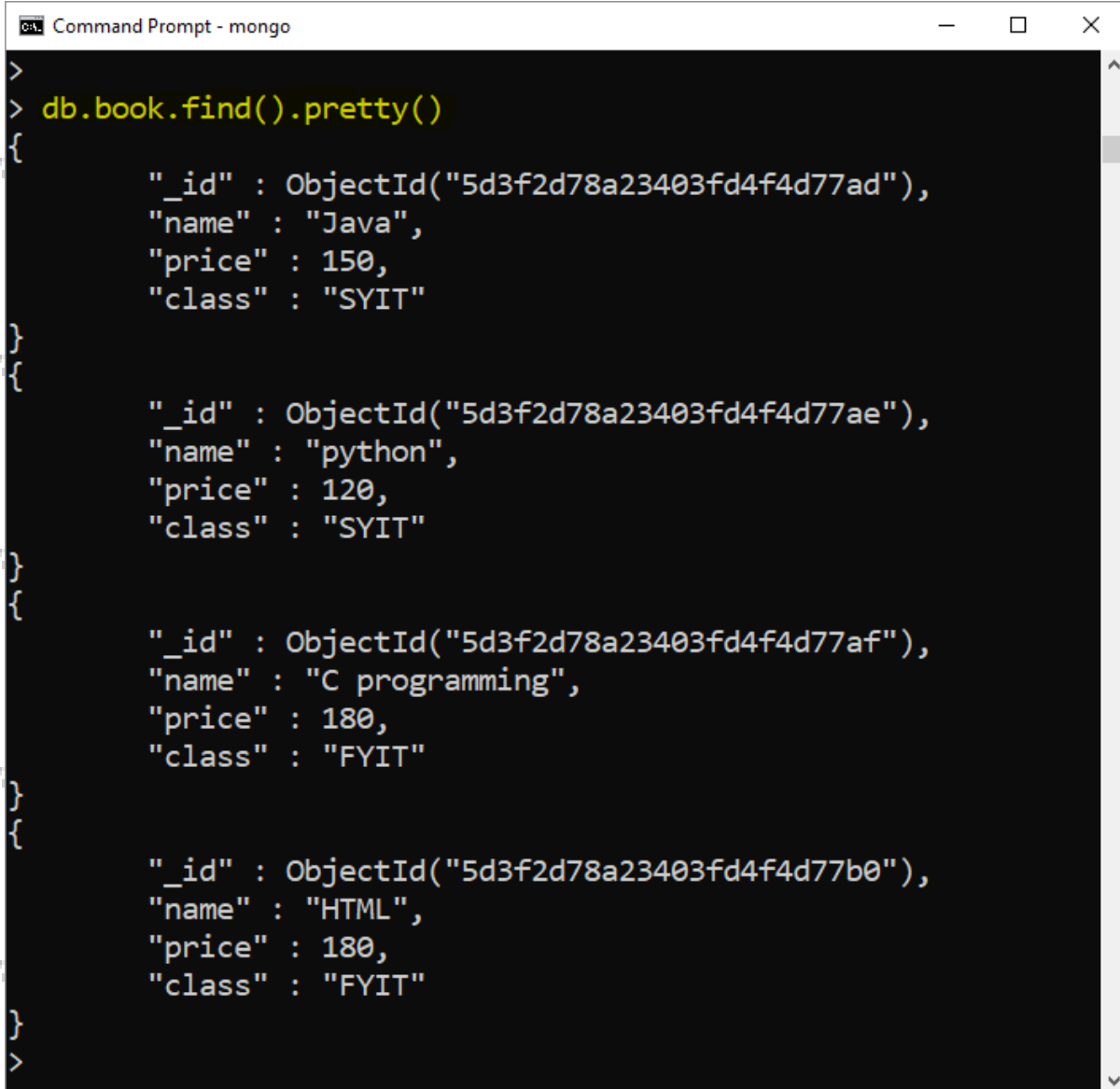
Let's create new collection named book.

```
> db.book.insert( [
...   {name:"Java", price:150, class : "SYIT"},
...   {name:"python", price:120, class: "SYIT"},
...   {name:"C programming",price:180, class: "FYIT"},
...   {name: "HTML", price: 180, class: "FYIT"}
... ]);
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 4,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```


To fetch all documents from collection we will use **find()** method we will use **pretty()** to make result readable.

```
> db.book.find()
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
```

Retrieve using **pretty()**.



```
C:\> Command Prompt - mongo
>
> db.book.find().pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"),
  "name" : "Java",
  "price" : 150,
  "class" : "SYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"),
  "name" : "python",
  "price" : 120,
  "class" : "SYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"),
  "name" : "C programming",
  "price" : 180,
  "class" : "FYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
>
```

findOne() method will return first document from the collection.

```
> db.book.findOne()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"),
  "name" : "Java",
  "price" : 150,
  "class" : "SYIT"
}
>
```

If we want to retrieve specific documents from collection we have to mention those in parameters of **find()** like shown below.

```
> db.book.find(
... {"class" : "FYIT"}
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"),
  "name" : "C programming",
  "price" : 180,
  "class" : "FYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
```

Above command retrieves the documents from collection which has **key:value** pair of **"class" : "FYIT"**. Above query returned 2 documents but if we want to narrow it down to one we can pass more arguments in **find()**, as shown below.

```
> db.book.find(
... { "class" : "FYIT",
...   "name" : "HTML" }
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
```

Now above query will retrieve only those document which has **"class" : "FYIT"** and **"name" : "HTML"**.

If we want to retrieve documents which price is **greater than 150**, we will use following command.

```
> db.book.find(
... { "price" : { $gt : 150 } }
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"),
  "name" : "C programming",
  "price" : 180,
  "class" : "FYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
>
```

Here “\$gt” stands for **Greater Than**.

If we want to retrieve documents which price is **greater than or equal** to 150, we will use following command.

```
> db.book.find(
... { "price" : { $gte : 150 } }
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"),
  "name" : "Java",
  "price" : 150,
  "class" : "SYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"),
  "name" : "C programming",
  "price" : 180,
  "class" : "FYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
>
```

Here “\$gte” stands for **Greater Than or equal**.

If we want to retrieve book documents which price is **less than** 150, we will use following command.

```
> db.book.find(
... { "price" : { $lt : 150 } }
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"),
  "name" : "python",
  "price" : 120,
  "class" : "SYIT"
}
```

Here “\$lt” stands for **Less Than**.

If we want to retrieve book documents which price is **Less than or Equal to** 150, we will use following command.

```
> db.book.find(
... { "price" : { $lte : 150 } }
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"),
  "name" : "Java",
  "price" : 150,
  "class" : "SYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"),
  "name" : "python",
  "price" : 120,
  "class" : "SYIT"
}
```

Here “\$lte” stands for **Less Than or Equal to**.

If we want to retrieve book documents which price is **not equal to** 150, we will use following command.

```
> db.book.find(
... { "price" : { $ne : 150 } }
... ).pretty()
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"),
  "name" : "python",
  "price" : 120,
  "class" : "SYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"),
  "name" : "C programming",
  "price" : 180,
  "class" : "FYIT"
}
{
  "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
```

Here “\$ne” stands for **Not Equal to**.

If we want select those document which has “name” equal to “Java”
OR “HTML”.

```
> db.book.find( {
... $or:[
... {"name" : "Java"},
... {"name" : "HTML"}
... ]
... } )
{ "_id" : ObjectId("5d3f222fa23403fd4f4d77a9"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f222fa23403fd4f4d77ac"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
```

Above query will return those documents which has “name” equal to
“Java” OR “HTML”. Now if we want to make our output more readable
we will use **pretty()**.

```
> db.book.find( {
... $or:[
... {"name" : "Java"},
... {"name" : "HTML"}
... ]
... } ).pretty()
{
  "_id" : ObjectId("5d3f222fa23403fd4f4d77a9"),
  "name" : "Java",
  "price" : 150,
  "class" : "SYIT"
}
{
  "_id" : ObjectId("5d3f222fa23403fd4f4d77ac"),
  "name" : "HTML",
  "price" : 180,
  "class" : "FYIT"
}
```

If we want to retrieve the “name” values from documents where
“class”: “FYIT” than we have to pass another argument in **find()**, which
will only display “name” values.

```
> db.book.find(
... { "class" : "FYIT" },
... {"name":1}
... )
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"), "name" : "C programming" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"), "name" : "HTML" }
```

In “name”:1, “name” refers to key and if we set it to 1 that means we
want to retrieve “name” values, but we can see that mongoDB will also
returns “_id” values as well and to eliminate that we add “_id” : 0 with
“name” : 1.

```
> db.book.find(
... { "class" : "FYIT" },
... { "name" : 1, _id : 0 }
... )
{ "name" : "C programming" }
{ "name" : "HTML" }
>
```

And this states that display only “**name**” values and if we want to display “**price**” as well with “**name**” then we will add “**price**” : 1.

Note: Please remember that 1 will act like flag if it is 1 then it says that the element will be displayed and if it 0 then element will not be displayed.

Let’s display all the records using **find()**.

```
> db.book.find()
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
```

If we want to view first 3 documents from result set, we will use **limit()**.

```
> db.book.find().limit(3)
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
>
```

If we don’t want to view first 2 documents from result set, we will use **skip()**.

```
> db.book.find().skip(2)
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
```

Update document

Let’s display all the data.

```
> db.book.find()
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ad"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77ae"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77af"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f2d78a23403fd4f4d77b0"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
```

If we want to update the price of **HTML** book, we can use the following commands.

```
> db.book.update(
... { "name" : "HTML" },
... { $set : { "price" : 200 } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa62"), "name" : "HTML", "price" : 200, "class" : "FYIT" }
>
```

Update method will accept the **condition** in first parameter and in second parameter it will accept the key and its updated value, **\$set** is use to set updated value in document.

Let's try to update multiple documents at the same time, so we are going to update the price of **SYIT** books.

```
Command Prompt - mongo
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa62"), "name" : "HTML", "price" : 200, "class" : "FYIT" }
>
> db.book.update(
... { "class" : "SYIT" },
... { $set : { "price" : 100 } }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 100, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
```

We can see that our command updated only one document, this is because **update()** method will update the first document which satisfies the above given condition. So if we want to update both the records then we must use another parameter of **update()**.

```
Command Prompt - mongo
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 100, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa62"), "name" : "HTML", "price" : 200, "class" : "FYIT" }
>
> db.book.update(
... { "class" : "SYIT" },
... { $set : { "price" : 180 } },
... { multi : true }
... )
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa62"), "name" : "HTML", "price" : 200, "class" : "FYIT" }
```

Here **multi : true** says that it will change all the documents which satisfies the above given condition.

There is one more method to update the document i.e. by using **save()**.

First let's create new collection and this time we are assigning values for **"_id"**.

```
> db.courses.insert(
... [ { _id : 01 , "name" : "IT" },
... { _id : 02 , "name" : "CS" } ]
... )
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

```
> db.courses.find()
{ "_id" : 1, "name" : "IT" }
{ "_id" : 2, "name" : "CS" }
>
```

Now let's try and update the course **"name"** for **"_id:02"** using **save()**.

```
> db.courses.save(
... { _id : 02 , name : "MCA" }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.courses.find()
{ "_id" : 1, "name" : "IT" }
{ "_id" : 2, "name" : "MCA" }
>
```

Save() will take the complete document (we have to enter all the elements in this because if it finds the document with same **"_id"** then it will replace it) as argument and it will matches the **"_id"** in all documents and if it finds a match then it will update (**replace**) that document, and if it did not get any match then it will act as an insert command and will insert document in collection.

```
> db.courses.find()
{ "_id" : 1, "name" : "IT" }
{ "_id" : 2, "name" : "MCA" }
>
> db.courses.save(
... { _id : 03 , name : "PhD" }
... )
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
>
> db.courses.find()
{ "_id" : 1, "name" : "IT" }
{ "_id" : 2, "name" : "MCA" }
{ "_id" : 3, "name" : "PhD" }
>
```

Delete Document

To delete document we use **remove()**. Suppose we want to remove the **"HTML"** book we have to use following command.


```
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa62"), "name" : "HTML", "price" : 200, "class" : "FYIT" }
>
> db.book.remove(
... { "name" : "HTML" }
... )
WriteResult({ "nRemoved" : 1 })
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
>
```

In above example we have pass the condition in **remove()** on bases of which **remove()** will delete the document.

If we want to delete multiple document at the same time we can use following command.

```
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa5f"), "name" : "Java", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa60"), "name" : "python", "price" : 180, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
>
> db.book.remove(
... { "class" : "SYIT" }
... )
WriteResult({ "nRemoved" : 2 })
>
> db.book.find()
{ "_id" : ObjectId("5d3f51377f5e33a47588fa61"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
>
```

But if we want to delete only one document but more then on documents are satisfying the condition, then we have to use another parameter of **remove()**.

For this demonstration we have created new book table.

```
Command Prompt - mongo
> db.book.insert([ {name:"Java", price:150, class : "SYIT"}, {name:"python", price:120, class: "SYIT"}, {name:"C programming", price:180, class: "FYIT"}, {name: "HTML", price: 180, class: "FYIT"} ]);
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 4,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
> db.book.find()
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa64"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa65"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa66"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa67"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
```

Form this book collection we want to delete only first **"SYIT"** document. For this we will use following command.

```
> db.book.find()
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa64"), "name" : "Java", "price" : 150, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa65"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa66"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa67"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
>
> db.book.remove( { "class" : "SYIT" }, 1 )
WriteResult({ "nRemoved" : 1 })
>
> db.book.find()
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa65"), "name" : "python", "price" : 120, "class" : "SYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa66"), "name" : "C programming", "price" : 180, "class" : "FYIT" }
{ "_id" : ObjectId("5d3f62bb7f5e33a47588fa67"), "name" : "HTML", "price" : 180, "class" : "FYIT" }
>
```

In second argument we have to set **1**, which state that only one document is to be deleted.