Ex1:

create database ->

use employee

insert ng document->

db.employee.insert

( { "Employeeid" : 1,

"EmployeeName" : "Martin"

}

)

Inserting an array of values

var myEmployee=

[

{

"Employeeid" : 1,

"EmployeeName" : "Smith"

},

{

"Employeeid" : 2,

"EmployeeName" : "Mohan"

},

{

"Employeeid" : 3,

"EmployeeName" : "Joe"

},

];

db.employee.insert(myEmployee);

display all documents:

db.employee.find().forEach(printjson)

Ex2:

Use emps;

inserting single document:

db.emps.insert( {

id: '2011',

name: 'Ram',

email: 'ram@gmail.com',

department:'IT'

})

inserting multipe documents:

db.emps.insertMany([ {

id: '2011',

name: 'Ram',

email: 'ram@gmail.com',

department:'IT'

},

{

id: '2012',

name: 'Ram',

email: 'ram@gmail.com',

department:'IT'

},

{

id: '2013',

name: 'Ram',

email: 'ram@gmail.com',

department:'IT'

}

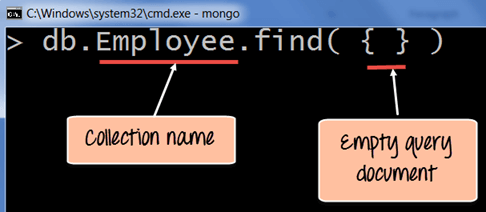
]

)

**MongoDB Basic Query Operations**

The basic MongoDB query operators cover the simple operations such as getting all of the documents in a MongoDB collection. Let’s look at an db.collection.find example of how we can accomplish this.

All of our code will be run in the MongoDB[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)command shell. Consider that we have a collection named ‘Employee’ in our MongoDB database and we execute the below command.



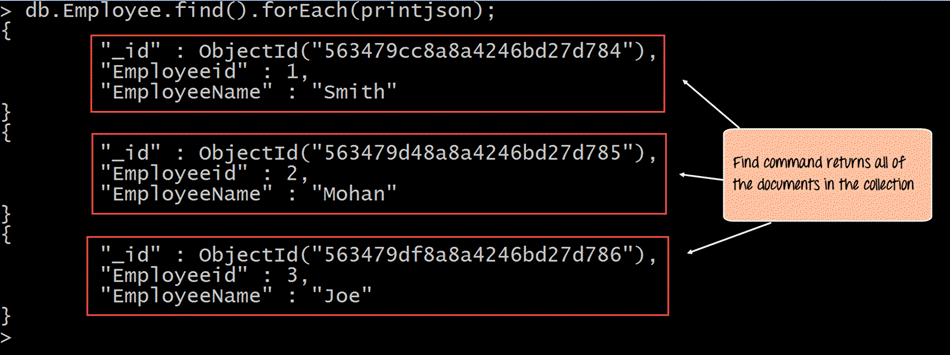
MongoDB Basic query operation

**Code Explanation:**

1. Employee is the collection name in the MongoDB database
2. The MongoDB find query is an in-built function which is used to retrieve the documents in the collection.

If the command is executed successfully, the following Output will be shown for the MongoDB find example

**Output:**



The output shows all the documents which are present in the collection.

We can also add criteria to our queries so that we can fetch documents based on certain conditions.

**MongoDB Query Example – 1**

Let’s look at a couple of MongoDB query examples of how we can accomplish this.

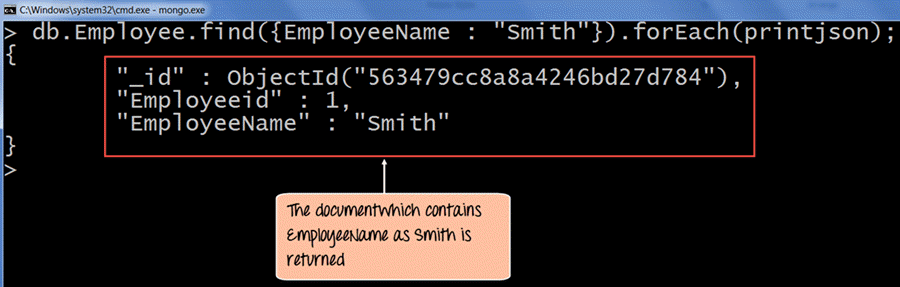
db.Employee.find({EmployeeName : "Smith"}).forEach(printjson);

**Code Explanation:**

1. Here we want to find for an Employee whose name is “Smith” in the collection , hence we enter the filter criteria as EmployeeName : “Smith”

If the command is executed successfully, the following Output will be shown

**Output:**



The output shows that only the document which contains “Smith” as the Employee Name is returned.

**MongoDB Query Example – 2**

Now in this MongoDB queries tutorial, let’s take a look at another code example which makes use of the greater than search criteria. When this criteria is included, it actually searches those documents where the value of the field is greater than the specified value.

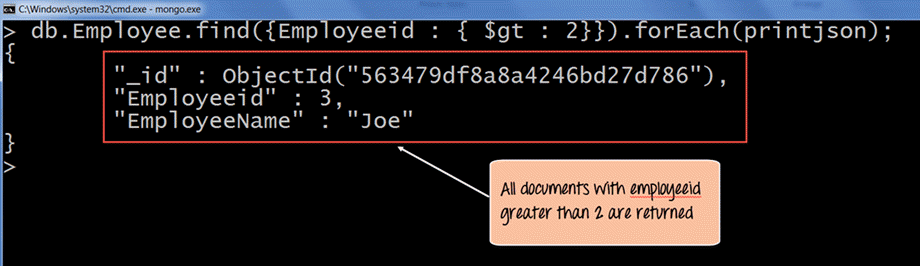
db.Employee.find({Employeeid : {$gt:2}}).forEach(printjson);

**Code Explanation:**

1. Here we want to find for all Employee’s whose id is greater than 2. The $gt is called a query selection operator, and what is just means is to use the greater than expression.

If the MongoDB select fields command is executed successfully, the following Output will be shown

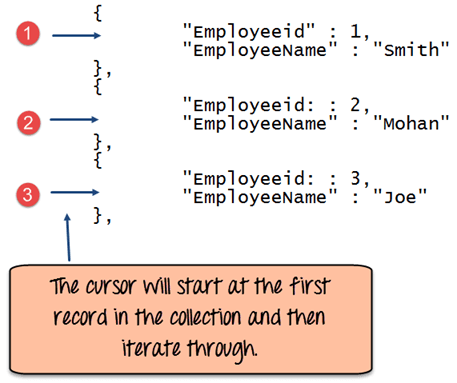
**Output:**



**What is Cursor in MongoDB?**

When the **db.collection.find ()** function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor.

By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one. If you see the below example, if we have 3 documents in our collection, the cursor object will point to the first document and then iterate through all of the documents of the collection.



The following example shows how this can be done.

var myEmployee = db.Employee.find( { Employeeid : { $gt:2 }});

while(myEmployee.hasNext())

{

print(tojson(myEmployee.next()));

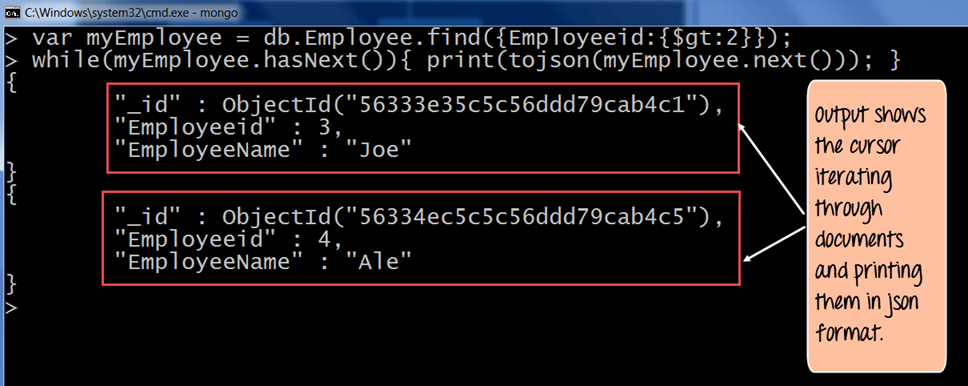
}

**Code Explanation:**

1. First we take the result set of the query which finds the Employee’s whose id is greater than 2 and assign it to the[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)variable ‘myEmployee’
2. Next we use the while loop to iterate through all of the documents which are returned as part of the query.
3. Finally for each document, we print the details of that document in JSON readable format.

If the command is executed successfully, the following Output will be shown

**Output:**



**MongoDB Limit Query Results**

This modifier is used to limit the number of documents which are returned in the result set for a query. The following example shows how this can be done.

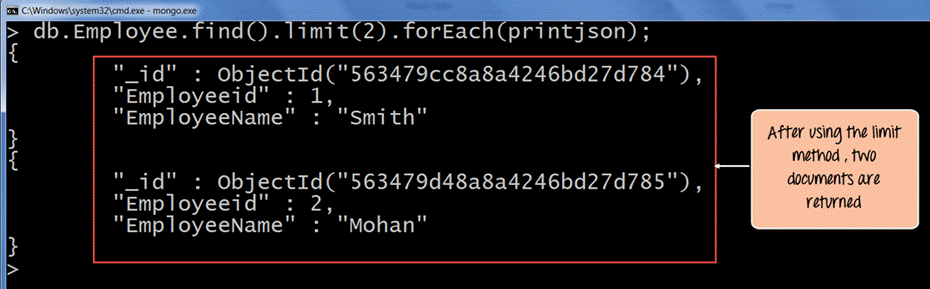
db.Employee.find().limit(2).forEach(printjson);

**Code Explanation:**

* The above code takes the find function which returns all of the documents in the collection but then uses the limit clause to limit the number of documents being returned to just 2.

**Output:**

If the command is executed successfully, the following Output will be shown



The output clearly shows that since there is a limit modifier, so at most just 2 records are returned as part of the result set based on the ObjectId in ascending order.

**MongoDB Sort by Descending Order**

One can specify the order of documents to be returned based on ascending or descending order of any key in the collection. The following example shows how this can be done.

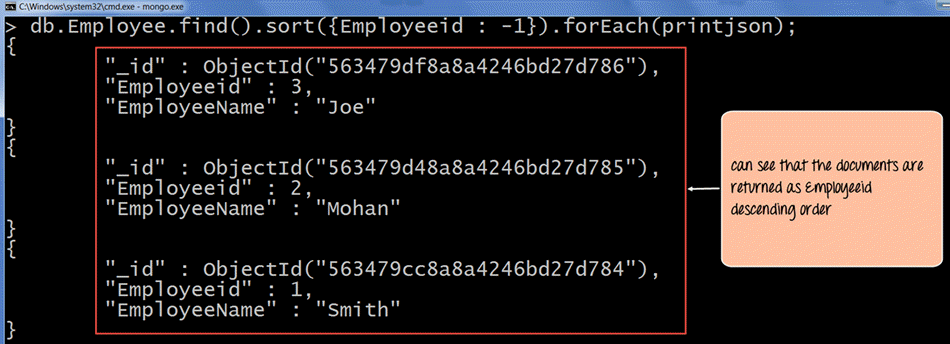
db.Employee.find().sort({Employeeid:-1}).forEach(printjson)

**Code Explanation:**

* The above code takes the sort function which returns all of the documents in the collection but then uses the modifier to change the order in which the records are returned. Here the -1 indicates that we want to return the documents based on the descending order of Employee id.

If the command is executed successfully, the following Output will be shown

**Output:**



**MongoDB Count() Function**

The concept of aggregation is to carry out a computation on the results which are returned in a query. For example, suppose you wanted to know what is the count of documents in a collection as per the query fired, then MongoDB provides the count() function.

**Example of MongoDB Count() Function**

Let’s look at an example of this.

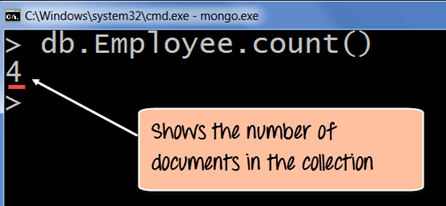
db.Employee.count()

**Code Explanation:**

* The above code executes the count function.

If the command is executed successfully, the following Output will be shown

**Output:**



The output clearly shows that 4 documents are there in the collection.

**Performing Modifications**

The other two classes of operations in MongoDB are the update and remove statements.

The update operations allow one to modify existing data, and the remove operations allow the deletion of data from a collection.

**Remove() Function in MongoDB**

In MongoDB, the **db.collection.remove()** method is used to remove documents from a collection. Either all of the documents can be removed from a collection or only those which matches a specific condition.

If you just issue the remove command, all of the documents will be removed from the collection.

**Example of MongoDB Remove() Function**

The following code example demonstrate how to remove a specific document from the collection.

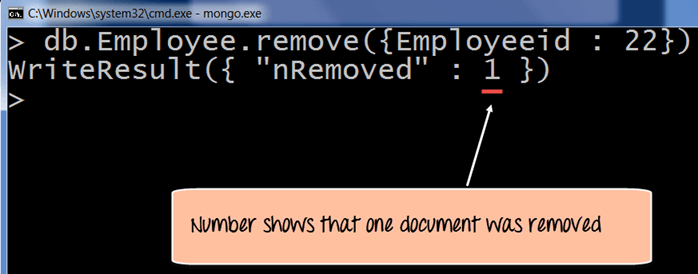
db.Employee.remove({Employeeid:22})

**Code Explanation:**

* The above code use the remove function and specifies the criteria which in this case is to remove the documents which have the Employee id as 22.

If the command is executed successfully, the following Output will be shown

**Output:**



## Basic document updates

MongoDB provides the update() command to update the documents of a collection. To update only the documents you want to update, you can add a criteria to the update statement so that only selected documents are updated.

The basic parameters in the command is a condition for which document needs to be updated, and the next is the modification which needs to be performed.

The following example shows how this can be done.

**Step 1)**Issue the update command

**Step 2)**Choose the condition which you want to use to decide which document needs to be updated. In our example, we want to update the document which has the Employee id 22.

**Step 3)** Use the set command to modify the Field Name

**Step 4)** Choose which Field Name you want to modify and enter the new value accordingly.

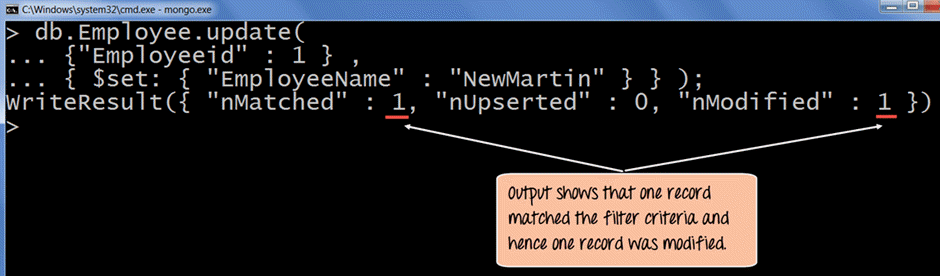
db.Employee.update(

{"Employeeid" : 1},

{$set: { "EmployeeName" : "NewMartin"}});

If the command is executed successfully, the following Output will be shown

**Output:**



The output clearly shows that one record matched the condition and hence the relevant field value was modified.

## Updating Multiple Values

To ensure that multiple/bulk documents are updated at the same time in MongoDB you need to use the multi option because otherwise by default only one document is modified at a time.

The following example shows how to update many documents.

In this example, we are going to first find the document which has the Employee id as “1” and change the Employee name from “Martin” to “NewMartin”

**Step 1)** Issue the update command

**Step 2)** Choose the condition which you want to use to decide which document needs to be updated. In our example, we want the document which has the Employee id of “1” to be updated.

**Step 3)** Choose which Field Name’s you want to modify and enter their new value accordingly.

db.Employee.update

(

{

Employeeid : 1

},

{

$set :

{

"EmployeeName" : "NewMartin",

"Employeeid" : 22

}

}

)

If the command is executed successfully and if you run the **“find”** command to search for the document with Employee id as 22 you will see the following Output will be shown

**Output:**

