The use Command

MongoDB **use DATABASE\_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows −

use DATABASE\_NAME

Example

If you want to use a database with name **<mydb>**, then **use DATABASE** statement would be as follows −

>use mydb

switched to db mydb

To check your currently selected database, use the command **db**

>db

mydb

If you want to check your databases list, use the command **show dbs**.

>**show dbs**

local 0.78125GB

test 0.23012GB

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

>db.movie.insert({"name":"DDLJ"})

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

**In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.**

## The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

### Syntax

Basic syntax of **dropDatabase()** command is as follows −

db.dropDatabase()

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

### Example

First, check the list of available databases by using the command, **show dbs**.

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

>

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows −

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>

Now check list of databases.

>show dbs

local 0.78125GB

test 0.23012GB

>

The createCollection() Method

MongoDB **db.createCollection(name, options)** is used to create collection.

Syntax

Basic syntax of **createCollection()** command is as follows −

db.createCollection(name, options)

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Name | String | Name of the collection to be created |
| Options | Document | (Optional) Specify options about memory size and indexing |

Options parameter is optional, so you need to specify only the name of the collection. Following is the list of options you can use −

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| capped | Boolean | (Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.** |
| autoIndexId | Boolean | (Optional) If true, automatically create index on \_id field.s Default value is false. |
| size | number | (Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.** |
| max | number | (Optional) Specifies the maximum number of documents allowed in the capped collection. |

While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

Examples

Basic syntax of **createCollection()** method without options is as follows −

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

You can check the created collection by using the command **show collections**.

>show collections

mycollection

system.indexes

The following example shows the syntax of **createCollection()** method with few important options −

> db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } ){

"ok" : 0,

"errmsg" : "BSON field 'create.autoIndexID' is an unknown field.",

"code" : 40415,

"codeName" : "Location40415"

}

>

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

>db.tutorialspoint.insert({"name" : "tutorialspoint"}),

WriteResult({ "nInserted" : 1 })

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

## The drop() Method

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

### Syntax

Basic syntax of **drop()** command is as follows −

db.COLLECTION\_NAME.drop()

### Example

First, check the available collections into your database **mydb**.

>use mydb

switched to db mydb

>show collections

mycol

mycollection

system.indexes

tutorialspoint

>

Now drop the collection with the name **mycollection**.

>db.mycollection.drop()

true

>

Again check the list of collections into database.

>show collections

mycol

system.indexes

tutorialspoint

>

drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

MongoDB supports many datatypes. Some of them are −

* **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
* **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
* **Boolean** − This type is used to store a boolean (true/ false) value.
* **Double** − This type is used to store floating point values.
* **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
* **Arrays** − This type is used to store arrays or list or multiple values into one key.
* **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
* **Object** − This datatype is used for embedded documents.
* **Null** − This type is used to store a Null value.
* **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
* **Date**− This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
* **Object ID** − This datatype is used to store the document’s ID.
* **Binary data** − This datatype is used to store binary data.
* **Code** − This datatype is used to store JavaScript code into the document.
* **Regular expression** − This datatype is used to store regular expression.

## The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

### Syntax

The basic syntax of **insert()** command is as follows −

>db.COLLECTION\_NAME.insert(document)

### Example

> db.users.insert({

... \_id : ObjectId("507f191e810c19729de860ea"),

... title: "MongoDB Overview",

... description: "MongoDB is no sql database",

... by: "tutorials point",

... url: "http://www.tutorialspoint.com",

... tags: ['mongodb', 'database', 'NoSQL'],

... likes: 100

... })

WriteResult({ "nInserted" : 1 })

>

Here **mycol** is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the \_id parameter, then MongoDB assigns a unique ObjectId for this document.

\_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

\_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

You can also pass an array of documents into the insert() method as shown below:.

> db.createCollection("post")

> db.post.insert([

{

title: "MongoDB Overview",

description: "MongoDB is no SQL database",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 100

},

{

title: "NoSQL Database",

description: "NoSQL database doesn't have tables",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 20,

comments: [

{

user:"user1",

message: "My first comment",

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

BulkWriteResult({

"writeErrors" : [ ],

"writeConcernErrors" : [ ],

"nInserted" : 2,

"nUpserted" : 0,

"nMatched" : 0,

"nModified" : 0,

"nRemoved" : 0,

"upserted" : [ ]

})

>

To insert the document you can use **db.post.save(document)** also. If you don't specify **\_id** in the document then **save()** method will work same as **insert()** method. If you specify \_id then it will replace whole data of document containing \_id as specified in save() method.

## The insertOne() method

If you need to insert only one document into a collection you can use this method.

### Syntax

The basic syntax of insert() command is as follows −

>db.COLLECTION\_NAME.insertOne(document)

### Example

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

> db.createCollection("empDetails")

{ "ok" : 1 }

> db.empDetails.insertOne(

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

})

{

"acknowledged" : true,

"insertedId" : ObjectId("5dd62b4070fb13eec3963bea")

}

>

## The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

### Example

Following example inserts three different documents into the empDetails collection using the insertMany() method.

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5dd631f270fb13eec3963bed"),

ObjectId("5dd631f270fb13eec3963bee"),

ObjectId("5dd631f270fb13eec3963bef")

]

}

>

## The find() Method

To query data from MongoDB collection, you need to use MongoDB's **find()** method.

### Syntax

The basic syntax of **find()** method is as follows −

>db.COLLECTION\_NAME.find()

**find()** method will display all the documents in a non-structured way.

### Example

Assume we have created a collection named mycol as −

> use sampleDB

switched to db sampleDB

> db.createCollection("mycol")

{ "ok" : 1 }

>

And inserted 3 documents in it using the insert() method as shown below −

> db.mycol.insert([

{

title: "MongoDB Overview",

description: "MongoDB is no SQL database",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 100

},

{

title: "NoSQL Database",

description: "NoSQL database doesn't have tables",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 20,

comments: [

{

user:"user1",

message: "My first comment",

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

Following method retrieves all the documents in the collection −

> db.mycol.find()

{ "\_id" : ObjectId("5dd4e2cc0821d3b44607534c"), "title" : "MongoDB Overview", "description" : "MongoDB is no SQL database", "by" : "tutorials point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }

{ "\_id" : ObjectId("5dd4e2cc0821d3b44607534d"), "title" : "NoSQL Database", "description" : "NoSQL database doesn't have tables", "by" : "tutorials point", "url" : "http://www.tutorialspoint.com", "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 20, "comments" : [ { "user" : "user1", "message" : "My first comment", "dateCreated" : ISODate("2013-12-09T21:05:00Z"), "like" : 0 } ] }

>

## The pretty() Method

To display the results in a formatted way, you can use pretty() method.

### Syntax

>db.COLLECTION\_NAME.find().pretty()

### Example

Following example retrieves all the documents from the collection named mycol and arranges them in an easy-to-read format.

> db.mycol.find().pretty()

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534c"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534d"),

"title" : "NoSQL Database",

"description" : "NoSQL database doesn't have tables",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 20,

"comments" : [

{

"user" : "user1",

"message" : "My first comment",

"dateCreated" : ISODate("2013-12-09T21:05:00Z"),

"like" : 0

}

]

}

## The findOne() method

Apart from the find() method, there is **findOne()** method, that returns only one document.

### Syntax

>db.COLLECTIONNAME.findOne()

### Example

Following example retrieves the document with title MongoDB Overview.

> db.mycol.findOne({title: "MongoDB Overview"})

{

"\_id" : ObjectId("5dd6542170fb13eec3963bf0"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

## RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS Equivalent** |
| Equality | {<key>:{$eg;<value>}} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |
| Values in an array | {<key>:{$in:[<value1>, <value2>,……<valueN>]}} | db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty() | Where name matches any of the value in :["Raj", "Ram", "Raghu"] |
| Values not in an array | {<key>:{$nin:<value>}} | db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty() | Where name values is not in the array :["Ramu", "Raghav"] or, doesn’t exist at all |

## AND in MongoDB

### Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND −

>db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })

### Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

> db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id" : ObjectId("5dd4e2cc0821d3b44607534c"),

"title" : "MongoDB Overview",

"description" : "MongoDB is no SQL database",

"by" : "tutorials point",

"url" : "http://www.tutorialspoint.com",

"tags" : [

"mongodb",

"database",

"NoSQL"

],

"likes" : 100

}

>

For the above given example, equivalent where clause will be **' where by = 'tutorials point' AND title = 'MongoDB Overview' '**. You can pass any number of key, value pairs in find clause.

## OR in MongoDB

### Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

## Using AND and OR Together

### Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},

{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

>

## NOR in MongoDB

### Syntax

To query documents based on the NOT condition, you need to use $not keyword. Following is the basic syntax of **NOT** −

>db.COLLECTION\_NAME.find(

{

$not: [

{key1: value1}, {key2:value2}

]

}

)

### Example

Assume we have inserted 3 documents in the collection **empDetails** as shown below −

db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Age: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Age: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Age: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

Following example will retrieve the document(s) whose first name is not "Radhika" and last name is not "Christopher"

> db.empDetails.find(

{

$nor:[

40

{"First\_Name": "Radhika"},

{"Last\_Name": "Christopher"}

]

}

).pretty()

{

"\_id" : ObjectId("5dd631f270fb13eec3963bef"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Age" : "24",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

## NOT in MongoDB

### Syntax

To query documents based on the NOT condition, you need to use $not keyword following is the basic syntax of **NOT** −

>db.COLLECTION\_NAME.find(

{

$NOT: [

{key1: value1}, {key2:value2}

]

}

).pretty()

### Example

Following example will retrieve the document(s) whose age is not greater than 25

> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )

{

"\_id" : ObjectId("5dd6636870fb13eec3963bf7"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Age" : "24",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

## MongoDB Update() Method

The update() method updates the values in the existing document.

### Syntax

The basic syntax of **update()** method is as follows −

>db.COLLECTION\_NAME.update(SELECTION\_CRITERIA, UPDATED\_DATA)

### Example

Consider the mycol collection has the following data.

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

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

>db.mycol.find()

{ "\_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}

{ "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

## MongoDB Save() Method

The **save()** method replaces the existing document with the new document passed in the save() method.

### Syntax

The basic syntax of MongoDB **save()** method is shown below −

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

### Example

Following example will replace the document with the \_id '5983548781331adf45ec5'.

>db.mycol.save(

{

"\_id" : ObjectId("507f191e810c19729de860ea"),

"title":"Tutorials Point New Topic",

"by":"Tutorials Point"

}

)

WriteResult({

"nMatched" : 0,

"nUpserted" : 1,

"nModified" : 0,

"\_id" : ObjectId("507f191e810c19729de860ea")

})

>db.mycol.find()

{ "\_id" : ObjectId("507f191e810c19729de860e6"), "title":"Tutorials Point New Topic",

"by":"Tutorials Point"}

{ "\_id" : ObjectId("507f191e810c19729de860e6"), "title":"NoSQL Overview"}

{ "\_id" : ObjectId("507f191e810c19729de860e6"), "title":"Tutorials Point Overview"}

>

## MongoDB findOneAndUpdate() method

The **findOneAndUpdate()** method updates the values in the existing document.

### Syntax

The basic syntax of **findOneAndUpdate()** method is as follows −

>db.COLLECTION\_NAME.findOneAndUpdate(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

### Example

Assume we have created a collection named empDetails and inserted three documents in it as shown below −

> db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Age: "26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Age: "27",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Age: "24",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

Following example updates the age and email values of the document with name 'Radhika'.

> db.empDetails.findOneAndUpdate(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

{

"\_id" : ObjectId("5dd6636870fb13eec3963bf5"),

"First\_Name" : "Radhika",

"Last\_Name" : "Sharma",

"Age" : "30",

"e\_mail" : "radhika\_newemail@gmail.com",

"phone" : "9000012345"

}

## MongoDB updateOne() method

This methods updates a single document which matches the given filter.

### Syntax

The basic syntax of updateOne() method is as follows −

>db.COLLECTION\_NAME.updateOne(<filter>, <update>)

### Example

> db.empDetails.updateOne(

{First\_Name: 'Radhika'},

{ $set: { Age: '30',e\_mail: 'radhika\_newemail@gmail.com'}}

)

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }

>

## MongoDB updateMany() method

The updateMany() method updates all the documents that matches the given filter.

### Syntax

The basic syntax of updateMany() method is as follows −

>db.COLLECTION\_NAME.update(<filter>, <update>)

### Example

> db.empDetails.updateMany(

{Age:{ $gt: "25" }},

{ $set: { Age: '00'}}

)

{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }

You can see the updated values if you retrieve the contents of the document using the find method as shown below −

> db.empDetails.find()

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf5"), "First\_Name" : "Radhika", "Last\_Name" : "Sharma", "Age" : "00", "e\_mail" : "radhika\_newemail@gmail.com", "phone" : "9000012345" }

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf6"), "First\_Name" : "Rachel", "Last\_Name" : "Christopher", "Age" : "00", "e\_mail" : "Rachel\_Christopher.123@gmail.com", "phone" : "9000054321" }

{ "\_id" : ObjectId("5dd6636870fb13eec3963bf7"), "First\_Name" : "Fathima", "Last\_Name" : "Sheik", "Age" : "24", "e\_mail" : "Fathima\_Sheik.123@gmail.com", "phone" : "9000054321" }

>

## The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

* **deletion criteria** − (Optional) deletion criteria according to documents will be removed.
* **justOne** − (Optional) if set to true or 1, then remove only one document.

### Syntax

Basic syntax of **remove()** method is as follows −

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

### Example

Consider the mycol collection has the following data.

{\_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},

{\_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},

{\_id : ObjectId("507f191e810c19729de860e3"), title: "Tutorials Point Overview"}

Following example will remove all the documents whose title is 'MongoDB Overview'.

>db.mycol.remove({'title':'MongoDB Overview'})

WriteResult({"nRemoved" : 1})

> db.mycol.find()

{"\_id" : ObjectId("507f191e810c19729de860e2"), "title" : "NoSQL Overview" }

{"\_id" : ObjectId("507f191e810c19729de860e3"), "title" : "Tutorials Point Overview" }

## Remove Only One

If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

## Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

> db.mycol.remove({})

WriteResult({ "nRemoved" : 2 })

> db.mycol.find()

>