1. Connect with going to location then start mongod.exe --dbpath ""

2. 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.

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

4. To check your currently selected database, use the command db.

5. db.movie.insert({"name":"tutorials point"})

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

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

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

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

>show collections

mycollection

system.indexes

9. 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" : 1 }

>

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

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

>db.COLLECTION\_NAME.insert(document)

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

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

14. **Conditions:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS Equivalent** |
| Equality | {<key>:<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 |

15. In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition. Following is the basic syntax of **AND** −

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

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(7df78ad8902c),

"title": "MongoDB Overview",

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

"by": "tutorials point",

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

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

"likes": "100"

}

> db.contacts.find({$and:[{"first\_name":"Abhishek"},{"last\_name":"Saxena"}]}).pretty()

{

"\_id" : ObjectId("5a3bdc5fc9f15d134473b9b8"),

"first\_name" : "Abhishek",

"last\_name" : "Saxena",

"phone" : "9540904552",

"\_\_v" : 0

}

16. 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"

}

>

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

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

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

>db.mycol.find()

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

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

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

>

18. 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})

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

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

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

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()

>db.mycol.find()

>

20. The basic syntax of **find()** method with projection is as follows −

>db.COLLECTION\_NAME.find({},{KEY:1})

> db.contacts.find({}, {"first\_name":1, "\_id":0}).pretty()

{ "first\_name" : "Bruce" }

{ "first\_name" : "Abhishek" }

21. The basic syntax of **limit()** method is as follows −

>db.COLLECTION\_NAME.find().limit(NUMBER)

>db.mycol.find({},{"title":1,\_id:0}).limit(2)

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

>

> db.contacts.find({}, {"first\_name":true, "\_id":false}).limit(1).pretty()

{ "first\_name" : "Bruce" }

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

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

> db.contacts.find({}, {"first\_name":true, "\_id":false}).limit(2).skip(1).pretty()

{ "first\_name" : "Abhishek" }

22. Sort

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

>db.COLLECTION\_NAME.find().sort({KEY:1})

>db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

{"title":"Tutorials Point Overview"}

{"title":"NoSQL Overview"}

{"title":"MongoDB Overview"}

> db.contacts.find({}, {"first\_name":true, "\_id":false}).limit(2).sort({"first\_name":1}).pretty()

{ "first\_name" : "Abhishek" }

{ "first\_name" : "Bruce" }

> db.contacts.find({}, {"first\_name":true, "\_id":false}).limit(2).sort({"first\_name":-1}).pretty()

{ "first\_name" : "Bruce" }

{ "first\_name" : "Abhishek" }

23. Indexing

The basic syntax of **ensureIndex()** method is as follows().

>db.COLLECTION\_NAME.ensureIndex({KEY:1})

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

### Example

>db.mycol.ensureIndex({"title":1})

>

In **ensureIndex()** method you can pass multiple fields, to create index on

multiple fields.

>db.mycol.ensureIndex({"title":1,"description":-1})

24. Aggregation

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

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

### Example

In the collection you have the following data −

{

\_id: ObjectId(7df78ad8902c)

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by\_user: 'tutorials point',

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

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

likes: 100

},

{

\_id: ObjectId(7df78ad8902d)

title: 'NoSQL Overview',

description: 'No sql database is very fast',

by\_user: 'tutorials point',

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

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

likes: 10

},

{

\_id: ObjectId(7df78ad8902e)

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Neo4j',

url: 'http://www.neo4j.com',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

},

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following **aggregate()**method −

> db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

{

"result" : [

{

"\_id" : "tutorials point",

"num\_tutorial" : 2

},

{

"\_id" : "Neo4j",

"num\_tutorial" : 1

}

],

"ok" : 1

}

25. Text Indexing

We will create a text index on post\_text field so that we can search inside our posts' text −

>db.posts.ensureIndex({post\_text:"text"})

Now that we have created the text index on post\_text field, we will search for all the posts having the word **tutorialspoint** in their text.

>db.posts.find({$text:{$search:"tutorialspoint"}})

To delete an existing text index, first find the name of index using the following query −

>db.posts.getIndexes()

After getting the name of your index from above query, run the following command. Here, **post\_text\_text** is the name of the index.

>db.posts.dropIndex("post\_text\_text")

## 26. Using regex Expression

The following regex query searches for all the posts containing string **tutorialspoint** in it −

>db.posts.find({post\_text:{$regex:"tutorialspoint"}})

The same query can also be written as −

>db.posts.find({post\_text:/tutorialspoint/})

To make the search case insensitive, we use the **$options** parameter with value **$i**. The following command will look for strings having the word **tutorialspoint**, irrespective of smaller or capital case −

>db.posts.find({post\_text:{$regex:"tutorialspoint",$options:"$i"}})

We can also use the concept of regex on array field. This is particularly very important when we implement the functionality of tags. So, if you want to search for all the posts having tags beginning from the word tutorial (either tutorial or tutorials or tutorialpoint or tutorialphp), you can use the following code −

>db.posts.find({tags:{$regex:"tutorial"}})

## Optimizing Regular Expression Queries

* If the document fields are **indexed**, the query will use make use of indexed values to match the regular expression. This makes the search very fast as compared to the regular expression scanning the whole collection.
* If the regular expression is a **prefix expression**, all the matches are meant to start with a certain string characters. For e.g., if the regex expression is **^tut**, then the query has to search for only those strings that begin with **tut**.

Examples:

>db.products.findAndModify({

query:{\_id:2,product\_available:{$gt:0}},

update:{

$inc:{product\_available:-1},

$push:{product\_bought\_by:{customer:"rob",date:"9-Jan-2014"}}

}

})