

MongoDB - Overview

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```
{
  _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,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide **_id** while inserting the document. If you don't provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

MongoDB - Environment

Let us now see how to install MongoDB on Windows.

Install MongoDB On Windows

To install MongoDB on Windows, first download the latest release of MongoDB from <https://www.mongodb.org/downloads>. Make sure you get correct version of MongoDB depending upon your Windows version. To get your Windows version, open command prompt and execute the following command.

```
C:\>wmic os get osarchitecture
OSArchitecture
64-bit
C:\>
```

32-bit versions of MongoDB only support databases smaller than 2GB and suitable only for testing and evaluation purposes.

Now extract your downloaded file to c:\ drive or any other location. Make sure the name of the extracted folder is mongodb-win32-i386-[version] or mongodb-win32-x86_64-[version]. Here [version] is the version of MongoDB download.

Next, open the command prompt and run the following command.

```
C:\>move mongodb-win64-* mongodb
1 dir(s) moved.
C:\>
```

In case you have extracted the MongoDB at different location, then go to that path by using command **cd FOOLDER/DIR** and now run the above given process.

MongoDB requires a data folder to store its files. The default location for the MongoDB data directory is c:\data\db. So you need to create this folder using the Command Prompt. Execute the following command sequence.

```
C:\>md data
C:\>md data\db
```

If you have to install the MongoDB at a different location, then you need to specify an alternate path for **\data\db** by setting the path **dbpath** in **mongod.exe**. For the same, issue the following commands.

In the command prompt, navigate to the bin directory present in the MongoDB installation folder. Suppose my installation folder is **D:\set up\mongodb**

```
C:\Users\XYZ>d:
D:\>cd "set up"
D:\set up>cd mongodb
D:\set up\mongodb>cd bin
D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set up\mongodb\data"
```

This will show **waiting for connections** message on the console output, which indicates that the mongod.exe process is running successfully.

Now to run the MongoDB, you need to open another command prompt and issue the following command.

```
D:\set up\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.6
connecting to: test
>db.test.save( { a: 1 } )
>db.test.find()
{ "_id" : ObjectId(5879b0f65a56a454), "a" : 1 }
>
```

This will show that MongoDB is installed and run successfully. Next time when you run MongoDB, you need to issue only commands.

```
D:\set up\mongodb\bin>mongod.exe --dbpath "d:\set up\mongodb\data"
D:\set up\mongodb\bin>mongo.exe
```

Install MongoDB on Ubuntu

Run the following command to import the MongoDB public GPG key –

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
```

Create a /etc/apt/sources.list.d/mongodb.list file using the following command.

```
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen'
| sudo tee /etc/apt/sources.list.d/mongodb.list
```

Now issue the following command to update the repository –

```
sudo apt-get update
```

Next install the MongoDB by using the following command –

```
apt-get install mongodb-10gen = 2.2.3
```

In the above installation, 2.2.3 is currently released MongoDB version. Make sure to install the latest version always. Now MongoDB is installed successfully.

Start MongoDB

```
sudo service mongod start
```

Stop MongoDB

```
sudo service mongod stop
```

Restart MongoDB

```
sudo service mongod restart
```

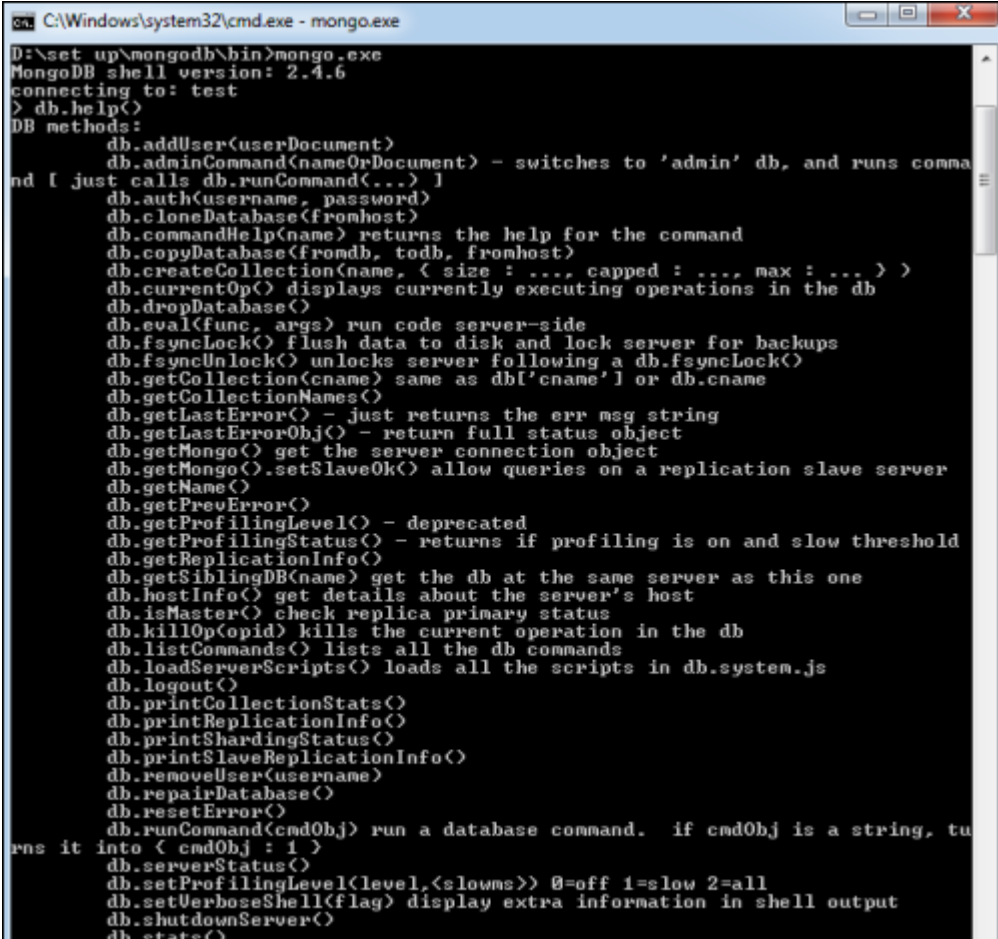
To use MongoDB run the following command.

```
mongo
```

This will connect you to running MongoDB instance.

MongoDB Help

To get a list of commands, type **db.help()** in MongoDB client. This will give you a list of commands as shown in the following screenshot.

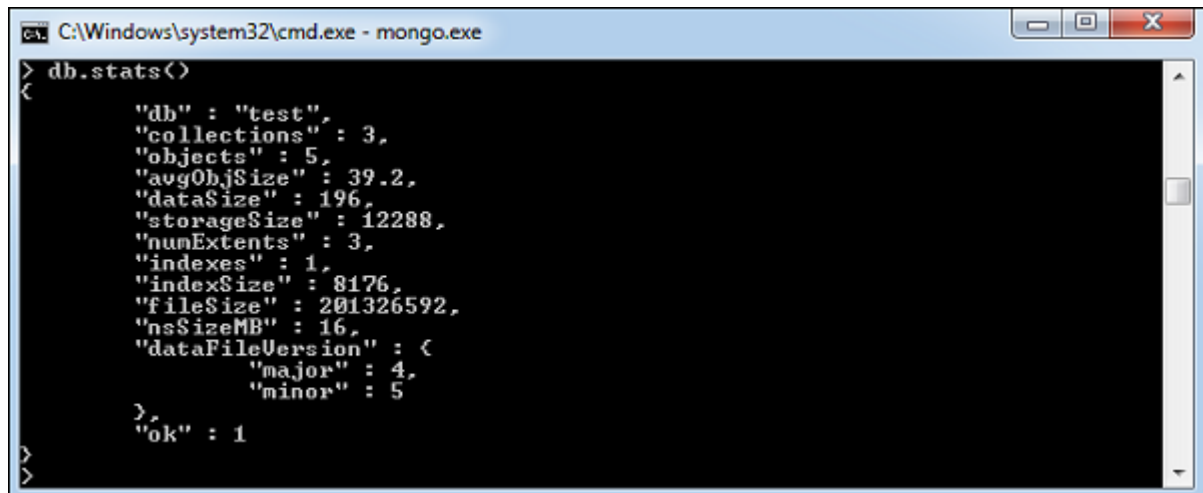
A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe - mongo.exe". The window shows the MongoDB shell version 2.4.6 connecting to a test database. The user has entered the command `> db.help()`, and the shell displays a list of DB methods. The methods listed include `db.addUser`, `db.adminCommand`, `db.auth`, `db.cloneDatabase`, `db.commandHelp`, `db.copyDatabase`, `db.createCollection`, `db.currentOp`, `db.dropDatabase`, `db.eval`, `db.fsyncLock`, `db.fsyncUnlock`, `db.getCollection`, `db.getCollectionNames`, `db.getLastError`, `db.getLastErrorObj`, `db.getMongo`, `db.getMongo.setSlaveOk`, `db.getName`, `db.getPrevError`, `db.getProfilingLevel`, `db.getProfilingStatus`, `db.getReplicationInfo`, `db.getSiblingDB`, `db.hostInfo`, `db.isMaster`, `db.killOp`, `db.listCommands`, `db.loadServerScripts`, `db.logout`, `db.printCollectionStats`, `db.printReplicationInfo`, `db.printShardingStatus`, `db.printSlaveReplicationInfo`, `db.removeUser`, `db.repairDatabase`, `db.resetError`, `db.runCommand`, `db.serverStatus`, `db.setProfilingLevel`, `db.setVerboseShell`, `db.shutdownServer`, and `db.stats`. The output is truncated at the bottom with "runs it into < cmdObj : 1 >".

```
C:\Windows\system32\cmd.exe - mongo.exe
D:\set up\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.6
connecting to: test
> db.help()
DB methods:
  db.addUser(userDocument)
  db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [ just calls db.runCommand(...) ]
  db.auth(username, password)
  db.cloneDatabase(fromhost)
  db.commandHelp(name) returns the help for the command
  db.copyDatabase(fromdb, todb, fromhost)
  db.createCollection(name, { size : ..., capped : ..., max : ... } )
  db.currentOp() displays currently executing operations in the db
  db.dropDatabase()
  db.eval(func, args) run code server-side
  db.fsyncLock() flush data to disk and lock server for backups
  db.fsyncUnlock() unlocks server following a db.fsyncLock()
  db.getCollection(cname) same as db['cname'] or db.cname
  db.getCollectionNames()
  db.getLastError() - just returns the err msg string
  db.getLastErrorObj() - return full status object
  db.getMongo() get the server connection object
  db.getMongo().setSlaveOk() allow queries on a replication slave server
  db.getName()
  db.getPrevError()
  db.getProfilingLevel() - deprecated
  db.getProfilingStatus() - returns if profiling is on and slow threshold
  db.getReplicationInfo()
  db.getSiblingDB(name) get the db at the same server as this one
  db.hostInfo() get details about the server's host
  db.isMaster() check replica primary status
  db.killOp(opid) kills the current operation in the db
  db.listCommands() lists all the db commands
  db.loadServerScripts() loads all the scripts in db.system.js
  db.logout()
  db.printCollectionStats()
  db.printReplicationInfo()
  db.printShardingStatus()
  db.printSlaveReplicationInfo()
  db.removeUser(username)
  db.repairDatabase()
  db.resetError()
  db.runCommand(cmdObj) run a database command.  if cmdObj is a string, turns it into { cmdObj : 1 }
  db.serverStatus()
  db.setProfilingLevel(level,<slowness>) 0=off 1=slow 2=all
  db.setVerboseShell(flag) display extra information in shell output
  db.shutdownServer()
  db.stats()
```

```
> db.version() current version of the server
```

MongoDB Statistics

To get stats about MongoDB server, type the command **db.stats()** in MongoDB client. This will show the database name, number of collection and documents in the database. Output of the command is shown in the following screenshot.



```
C:\Windows\system32\cmd.exe - mongo.exe
> db.stats()
{
  "db" : "test",
  "collections" : 3,
  "objects" : 5,
  "avgObjSize" : 39.2,
  "dataSize" : 196,
  "storageSize" : 12288,
  "numExtents" : 3,
  "indexes" : 1,
  "indexSize" : 8176,
  "fileSize" : 201326592,
  "nsSizeMB" : 16,
  "dataFileVersion" : {
    "major" : 4,
    "minor" : 5
  },
  "ok" : 1
}
```

MongoDB - Create Database

In this chapter, we will see how to create a database in MongoDB.

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":"tutorials point"})  
>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.

MongoDB - Create Collection

In this chapter, we will see how to create a collection using MongoDB.

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" : 1 }
>
```

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

```
>db.tutorialspoint.insert({"name" : "tutorialspoint"})
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

MongoDB - Datatypes

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** – timestamp. 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.

MongoDB - Delete Document

In this chapter, we will learn how to delete a document using MongoDB.

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(DELETION_CRITERIA)
```

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 remove all the documents whose title is 'MongoDB Overview'.

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "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({})  
>db.mycol.find()  
>
```

MongoDB - Drop Collection

In this chapter, we will see how to drop a collection using MongoDB.

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 - Drop Database

In this chapter, we will see how to drop a database using MongoDB command.

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
>
```

MongoDB - Insert Document

In this chapter, we will learn how to insert document in MongoDB collection.

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.mycol.insert({
  _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
})
```

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)
```

To insert multiple documents in a single query, you can pass an array of documents in **insert()** command.

Example

```
>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
      }
    ]
  }
]
})

```

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.

MongoDB - Limit Records

In this chapter, we will learn how to limit records using MongoDB.

The Limit() Method

To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of **limit()** method is as follows –

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Example

Consider the collection mycol 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 display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

If you don't specify the number argument in **limit()** method then it will display all documents from the collection.

MongoDB Skip() Method

Apart from **limit()** method, there is one more method **skip()** which also accepts number type argument and is used to skip the number of documents.

Syntax

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

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Example

Following example will display only the second document.

```
>db.mycol.find({}, {"title":1, _id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

Please note, the default value in **skip()** method is 0.

MongoDB - Projection

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

The find() Method

MongoDB's **find()** method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute **find()** method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

Syntax

The basic syntax of **find()** method with projection is as follows –

```
>db.COLLECTION_NAME.find({}, {KEY:1})
```

Example

Consider the collection mycol 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 display the title of the document while querying the document.

```
>db.mycol.find({}, {"title":1, _id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

Please note **_id** field is always displayed while executing **find()** method, if you don't want this field, then you need to set it as 0.

MongoDB - Query Document

In this chapter, we will learn how to query document from MongoDB collection.

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.

The pretty() Method

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

Syntax

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

Example

```
>db.mycol.find().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"
}
>
```

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

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>:<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

AND in MongoDB

Syntax

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"
}
```

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"  
}  
>
```

MongoDB - Update Document

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'}}
>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(5983548781331adf45ec5), "title":"Tutorials Point New Topic"  
    "by":"Tutorials Point"  
  }  
)  
>db.mycol.find()  
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic",  
  "by":"Tutorials Point"}  
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}  
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}  
>
```