# WEEK 10

# DBMS LAB

## What is NoSQL Database?

- A dynamic change in the nature of data - i.e., nowadays data are in structured, semi-structured, nonstructured as well as polymorphic in type.
- The variety of applications and the type of data feed into them for analysis has now become more diverse and distributed and is approaching cloud-oriented.
- Also, modern applications and services are serving tens of thousands of users in diverse geo-locations, having diverse time zones. So data integrity needs to be there at all the time.

Data residing in multiple virtual servers and other cloud storage (remote-based) in the cloud infrastructure can be easily analyzed using the NoSQL database management techniques and largely when the data set is in a non-structured manner. So, it can be said that the NoSQL database is intended to overcome the diversity of data, increase performance, modeling of data, scalability, and distribution, which is usually encountered in the Relational Databases.

### Structured Data Vs. Unstructured Data

Whereas unstructured data are haphazard data formats (such as document files, image files, video files, icons, etc.) where structured data can be pulled out or mine from unstructured data, but this process usually takes a lot of time. Modern-day data generated from different applications, services, or sources are a combination of structured and unstructured both. So, you will need something to store such data to make your application work properly. NoSQL based languages and scripts can help in this regard.

### Types of Database in NoSQL

Here are some of the common database types that come under NoSQL:

1. Document type databases: Here, the key gets paired with a compound data structure, i.e., document. MongoDB is an example of such type.
2. Key-Value stores: Here, each unstructured data is stored with a key for recognizing it.
3. Graph stores: In this type of database, data is stored mostly for networked data. It helps to relate data based on some existing data.
4. Wide-column stores: This type of data stores large data sets Cassandra (Used by Facebook), HBase are examples of such type.

# What is MongoDB?

**MongoDB** is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s.

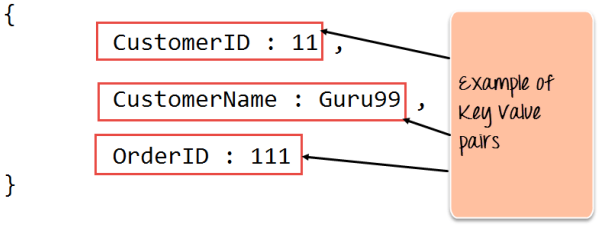
# Key Components of MongoDB Architecture

Below are a few of the common terms used in MongoDB

1. **_id** – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

| _Id | CustomerID | CustomerName | OrderID |
|---|---|---|---|
| 563479cc8a8a4246bd27d784 | 11 | Guru99 | 111 |
| 563479cc7a8a4246bd47d784 | 22 | Trevor Smith | 222 |
| 563479cc9a8a4246bd57d784 | 33 | Nicole | 333 |

2. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.
3. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
4. **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
5. **Document** – A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
6. **Field** – A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.
   The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair's defined in the document.



7. JSON – This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the _id field and a normal collection field. The _id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

Here is a table showing the relation between the terminologies used in RDBMS and MongoDB:

| RDBMS | MongoDB |
|---|---|
| Database | Database |
| Table | Collection |
| Tuple or Row | Document |
| Column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary key / Default key |
| Mysqld / Oracle | mongod |

# Installation of MangoDB

Install MongoDB Community Edition on Windows — MongoDB Manual

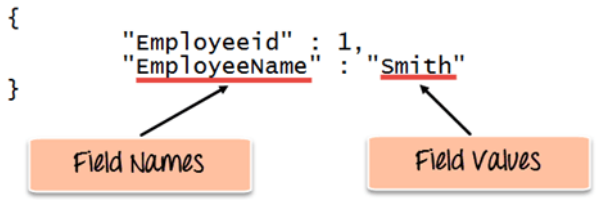
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

Install MongoDB Community Edition on Linux — MongoDB Manual

https://docs.mongodb.com/manual/administration/install-on-linux/

# How to Create Database & Collection in MongoDB

In MongoDB, the first basic step is to have a database and collection in place. The database is used to store all the collections, and the collection in turn is used to store all the documents. The documents in turn will contain the relevant Field Name and Field values.




The Field Names of the document are “Employeeid” and “EmployeeName” and the Field values are “1” and “Smith’ respectively. A bunch of documents would then make up a collection in MongoDB.

## Creating a database using “use” command

Creating a database in MongoDB is as simple as issuing the “**use**” command. The **“use”** command is used to create a database in MongoDB. If the database does not exist a new one will be created.

### Syntax

```
use DATABASE_NAME
```

### Example

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

```
> use week10
switched to db week10
```

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

```
> db
week10
>
```

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

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

Your created database (week10) is not present in list. To display database, you need to insert at least one document into it.

```
> db.student.insert({"name":"Anamika"})
WriteResult({ "nInserted" : 1 })
> show dbs
EmployeeDB  0.000GB
admin       0.000GB
config      0.000GB
local       0.000GB
week10      0.000GB
>
```

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

## Drop a database using db.dropDatabase() command.

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

### Syntax

```
db.dropDatabase()
```

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

Example:

```
> show dbs
EmployeeDB  0.000GB
admin       0.000GB
config      0.000GB
local       0.000GB
week10      0.000GB
```

If you want to delete new database **<week10>**, then **dropDatabase()** command would be as follows

```
> use week10
switched to db week10
> db.dropDatabase()
{ "ok" : 1 }
> show dbs
EmployeeDB  0.000GB
admin       0.000GB
config      0.000GB
local       0.000GB
```

## Create a collection using db.createCollection(name, options).

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

### Syntax:

```
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("Week10_NoSql")
{ "ok" : 1 }
>
```

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

```
> show collections
Week10_NoSql
>
```

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.week10.insert({"name" : "Week10_NoSql"})
```

```
> show collections
Week10_NoSql
```

Drop a collection using **db.collection.drop()**.

Syntax

```
db.COLLECTION_NAME.drop()
```

Example

```
> use week10
switched to db week10
> db.week10_NoSQL_Collection.insert({"name" : "Anamika"})
WriteResult({ "nInserted" : 1 })
> show collections
week10_NoSQL_Collection
> db.week10_NoSQL_Collection.drop()
true
> show collections
>
```

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

## 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 – 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.

Examples:

Integer:

```
db.TestCollection.insert({"Integer example": 62})
```

**Boolean:**

```
db.TestCollection.insert({"Nationality Indian": true})
```

Double:

```
db.TestCollection.insert({"double data type": 3.1415})
```

**String:**

```
db.TestCollection.insert({"string data type" : "This is a sample message."})
```

**Arrays:**

```
var degrees = ["BCA", "BS", "MCA"]
db.TestCollection.insert({" Array Example" : " Here is an example of
                          array"," Qualification" : degrees})
```

**Object:**

Object is implemented for embedded documents.

```
var embeddedObject = {"English" : 94, "ComputerSc." : 96, "Maths" :
                      80, "GeneralSc." : 85}
db.TestCollection.insert({"Object data type" : "This is Object",
                      "Marks" : embeddedObject})
```

NULL:

```
db.TestCollection.insert({" EmailID ": null}
```

**DATE:**

```
var date=new Date()
var date2=ISODate()
var month=date2.getMonth()
db.TestCollection.insert({"Date":date, "Date2":date2,
"Month":month})
```

## Insert document in MongoDB collection

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

### Syntax

```
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.nitt.edu.in", tags: ['mongodb', 'database', 'NoSQL'], likes: 100 })
```

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")
```

```
> show collections
Week10_NoSql
post
users
```

```
> db.post.insert([
... {
... title: "MongoDB Overview",
... description: "MongoDB is no SQL database",
... by: "DBMS LAB",
... url: "http://www.nitt.edu.in/cse",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 100
... },
... {
... title: "NoSQL Database",
... description: "NoSQL database doesn't have tables",
... by: "DbMS CSE",
... url: "http://www.nitt.edu.in",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 20,
... comments: [
... {
... user:"user1",
... message: "My first comment",
... dateCreated: new Date(2013,11,10,2,35),
... like: 0
... }
... ]
... }
... ])
```

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

Syntax:

```
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 }
> show collections
Week10_NoSql
empDetails
post
users
> db.empDetails.insertOne(
... {
... First_Name: "Radhika",
... Last_Name: "Sharma",
... Date_Of_Birth: "1995-09-26",
... e_mail: "radhika_sharma.123@gmail.com",
... phone: "9848022338"
... })
```

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

Example

```
> 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"
... }
... ]
... )
```

## How to query document from MongoDB collection

To query data from MongoDB collection using MongoDB's **find()** method.

### Syntax:

```
db.COLLECTION_NAME.find()
```

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

Example

create a collection named mycol as

```
> use sampleDB
switched to db sampleDB
> db.createCollection("mycol")
```

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: "DBMS LAB",
... url: "http://www.nitt.edu.in/cse.com",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 100
... },
... {
... title: "NoSQL Database",
... description: "NoSQL database doesn't have tables",
... by: "CSE",
... url: "http://www.nitt.edu.in/V semester.com",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 20,
... comments: [
... {
... user:"user1",
... message: "My first comment",
... dateCreated: new Date(2013,11,10,2,35),
... like: 0
... }
... ]
... }
... ])
```

retrieves all the documents in the collection –

```
db.mycol.find()
```

```
> db.mycol.find()
{ "_id" : ObjectId("6178d1118de0911dfdbde074"), "title" : "MongoDB Overview", "description" : "MongoDB is no SQL database", "by" : "DBMS LAB", "url" : "http://www.nitt.edu.in/cse.com", "tags"
: [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
{ "_id" : ObjectId("6178d1118de0911dfdbde075"), "title" : "NoSQL Database", "description" : "NoSQL database doesn't have tables", "by" : "CSE", "url" : "http://www.nitt.edu.in/V semester.com",
 "tags" : [ "mongodb", "database", "NoSQL" ], "likes" : 20, "comments" : [ { "user" : "user1", "message" : "My first comment", "dateCreated" : ISODate("2013-12-09T21:05:00Z"), "like" : 0 } ] }
```

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

```
db.COLLECTION_NAME.find().pretty()
```

**findOne()** method, that returns only one document.

```
db.COLLECTIONNAME.findOne()
```

## 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>:{$eg;<value>}} | db.mycol.find({"by":"DBMS LAB"}).pretty() | where by = 'DBMS LAB' |
| Less Than | {<key>:{$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

To query documents based on the AND condition

### Syntax:

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

### Example

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

```
{
        "_id" : ObjectId("6178d1118de0911dfdbde074"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "by" : "DBMS LAB",
        "url" : "http://www.nitt.edu.in/cse.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

## OR in MongoDB

### Syntax

To query documents based on the OR condition

```
db.mycol.find({$or:[{key1: value1},{key2:value2}]}).pretty()
```

### Example

```
> db.mycol.find({$or:[{"by":"No LAB"},{"title": "MongoDB Overview"}]}).pretty()
```

```
{
        "_id" : ObjectId("6178d1118de0911dfdbde074"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "by" : "DBMS LAB",
        "url" : "http://www.nitt.edu.in/cse.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 'DBMS LAB'.

```
db.mycol.find({"likes": {$gt:10}, $or: [{"by": "DBMS LAB"},
  {"title": "SQL Overview"}]}).pretty()
```

```
> db.mycol.find({"likes": {$gt:10}, $or: [{"by": "DBMS LAB"},
...    {"title": "SQL Overview"}]}).pretty()
{
        "_id" : ObjectId("6178d1118de0911dfdbde074"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "by" : "DBMS LAB",
        "url" : "http://www.nitt.edu.in/cse.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

## NOR in MongoDB

To query documents based on the NOT condition

### Syntax

```
db.COLLECTION_NAME.find({$not:[key1: value1}, {key2:value2}]})
```

### Example

```
> 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"
... }
... ]
... )
```

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

```
> db.empDetails.find({$nor:[{"First_Name": "Radhika"},{"Last_Name": "Christopher"}]}).pretty()
```

```
{
        "_id" : ObjectId("6178d6378de0911dfdbde078"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Age" : "24",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

## NOT in MongoDB

To query documents based on the NOT condition

### Syntax

```
db.COLLECTION_NAME.find({$NOT:[{key1: value1}, {key2:value2}]}).pretty()
```

### Example

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

```
> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } ).pretty()
{
        "_id" : ObjectId("6178d6378de0911dfdbde078"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Age" : "24",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

## Use of Update Operation in MongoDB

The update operation in a database is used to change or update values in a document. MongoDB makes use of the `update()` method for updating the documents within a MongoDB collection. For updating any specific documents, a new criterion can be incorporated with the update statement, which will only update the selected documents.

### Syntax

```
db.collection.update(
  <query>,
  <update>,
  {
    upsert: <boolean>,
    multi: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    arrayFilters: [ <filterdocument1>, ... ]
  }
)
```

### Example

First, let's select a record to update:

```
> db.musicians.find({ _id: 4 }).pretty()
{
        "_id" : 4,
        "name" : "Steve Morse",
        "instrument" : "Guitar",
        "born" : 1954
}
```

update the list of instrument played by this person, by making use of the `$set` operator for updating a single field.

```
> db.musicians.find({ _id: 4 }).pretty()
{
        "_id" : 4,
        "name" : "Steve Morse",
        "instrument" : "Guitar",
        "born" : 1954
}
> db.musicians.update(
...
... {_id: 4},
... {
...   $set: { instrument: ["Vocals", "Violin", "Octapad"] }
... }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.musicians.find({ _id: 4 }).pretty()
{
        "_id" : 4,
        "name" : "Steve Morse",
        "instrument" : [
                "Vocals",
                "Violin",
                "Octapad"
        ],
        "born" : 1954
}
```

Characteristics of Update in MongoDB

- In case your field does not subsist in the current document, the $set operator will insert a new field with the specified value, until and unless it violates the type constraint.
- MongoDB users can also make use of { multi: true } for updating multiple documents which will meet the query criteria.
- Making use of { upsert: true } for creating a new document is also possible as no document goes with the query.

## Deleting Documents in MongoDB

MongoDB allows you to delete a document or documents collectively using its one of the three methods. Three methods provided by MongoDB for deleting documents are:

1. db.collection.deleteOne()
2. db.collection.remove()
3. db.collection.deleteMany()

### db.collection.deleteOne()

This method is used to delete only a single document, even when more than one document matches with the criteria. Here is an example of using this db.collection.deleteOne() method for deleting the single document. To perform this process here, we have created a database and saved all the data separately.

Example

```
> db.programmers.insert(
... [
...   { name: "James Gosling" },
...   { name: "Dennis Ritchie" },
...   { name: "Bjarne Stroustrup" }
... ]
... )
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> db.programmers.find()
{ "_id" : ObjectId("61791304e891885292ce26d5"), "name" : "James Gosling" }
{ "_id" : ObjectId("61791304e891885292ce26d6"), "name" : "Dennis Ritchie" }
{ "_id" : ObjectId("61791304e891885292ce26d7"), "name" : "Bjarne Stroustrup" }
> db.programmers.deleteOne( { name: { $in: [ "Dennis Ritchie", "Bjarne Stroustrup"] } } )
{ "acknowledged" : true, "deletedCount" : 1 }
> db.programmers.find()
{ "_id" : ObjectId("61791304e891885292ce26d5"), "name" : "James Gosling" }
{ "_id" : ObjectId("61791304e891885292ce26d7"), "name" : "Bjarne Stroustrup" }
```

Executing this statement, you will notice that, although two documents match the criteria, only one document gets deleted.

## db.collection.remove()

It is possible for you to delete all your existing documents in a collection by simply excluding the filter criteria that is mentioned in the parenthesis () and specifying the document names within it. Now, to delete all documents from the programmers' collection, you have to write the query like this:

Example

```
> db.programmers.remove( {} )
WriteResult({ "nRemoved" : 2 })
```

But, when the filtering is done for removing elements, the db.collection.remove() method will document which matches the specified criteria.
Here, we delete all documents where the artist name is "James Gosling".

```
> db.programmers.remove( { name: "James Gosling" } )
WriteResult({ "nRemoved" : 0 })
> db.programmers.find()
>
```

## db.collection.deleteMany()

MongoDB allows you to delete multiple documents using the db.collection.deleteMany() method. This method deletes all your documents whichever match its criteria mentioned in the parameter. To check its implementation of db.collection.deleteMany() method, you can use the method the same way as done previously:

```
> db.programmers.deleteMany( { name: { $in: [ "Dennis Ritchie", "Bjarne Stroustrup" ] } } )
```

## Aggregation

In MongoDB, aggregation can be defined as the operation that is used for processing various types of data in the collection, which returns a calculated result. The concept of aggregation mainly clusters out your data from multiple different documents which are then used and operates in lots of ways (on

these clustered data) to return a combined result which can bring new information to the existing database. You can relate aggregation to that of the `count(*)` along with the `'group by'` used in SQL since both are equivalent in terms of the working.

MongoDB offers three different ways of performing aggregation:

- The aggregation pipeline.
- The map-reduce function.
- Single purpose aggregation methods.

MongoDB's aggregate function will cluster out the records in the form of a collection which can be then employed for providing operations like total number(sum), mean, minimum and maximum, etc. from the aggregated group of data extracted.

For performing such an aggregate function, the aggregate() method is used.

Syntax:
`db.collection_name.aggregate(aggregate_operation)`

a collection named `writers`, which has the following data:

```
> db.writers.find().pretty()
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0d"),
        "title" : "Networking",
        "description" : "Networking Essentials",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0e"),
        "title" : "Game Engineering",
        "description" : "Game Engineering and Development",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0f"),
        "title" : "PHP",
        "description" : "PHP as a General-Purpose",
        "author" : "Gautam"
}
>
```

db.writers.aggregate([

{$group : {_id : "$author", TotalBooksWritten : {$sum : 1}}}

])

```
> db.writers.aggregate([{$group : {_id : "$author", TotalBooksWritten : {$sum : 1}}}])
{ "_id" : "Gautam", "TotalBooksWritten" : 1 }
{ "_id" : "Gaurav Mandes", "TotalBooksWritten" : 2 }
>
```

| Expression | Description |
|---|---|
| $sum | adds up the definite values of every document of a collection. |
| $avg | computes the average values of every document of a collection. |
| $min | finds and returns the minimum of all values from within a collection. |
| $max | finds and returns the maximum of all values from within a collection. |
| $push | feeds in the values to an array in the associated document. |
| $first | fetches out the first document. |
| $last | fetches out the last document. |
| $addToSet | feeds in the values to an array without duplication. |

## Sorting

For sorting your MongoDB documents, you need to make use of the `sort()` method. This method will accept a document that has a list of fields and the order for sorting. For indicating the sorting order, you have to set the value `1 or -1` with the specific entity based on which the ordering will be set and displayed. One indicates organizing data in ascending order while -1 indicates organizing in descending order.

Syntax:

db.collection_name.find().sort({FieldName1: sort order 1 or -1, FieldName2: sort order})

Example:

Consider a collection that is having the following data:

{ "_id" : ObjectId(5983548781331abf45ec5), "topic":"Cyber Security"}

{ "_id" : ObjectId(5565548781331aef45ec6), " topic ":"Digital Privacy"}

{ "_id" : ObjectId(5983549391331abf45ec7), " topic ":"Application Security Engineering"}

Suppose I want to get data only from the topic field from all the documents in `ascending order`, then it will be executed like this:

Example:

db.techSubjects.find({}, {"topic":1, _id:0}).sort({"topic":1})

```
> db.techSubjects.find({}, {"topic":1, _id:0}).sort({"topic":1})
{ "topic" : "Application Security Engineering" }
{ "topic" : "Cyber Security" }
{ "topic" : "Digital Privacy" }
>
```

To display the topic field of all the techSubjects in descending order:

Example:

```
> db.techSubjects.find({}, {"topic":1, _id:0}).sort({"topic":-1})
{ "topic" : "Digital Privacy" }
{ "topic" : "Cyber Security" }
{ "topic" : "Application Security Engineering" }
>
```

## Indexing

The concept of indexing is crucial for any database, and so for MongoDB also. Databases having indexes make queries performance more efficient. When you have a collection with thousands of documents and no indexing is done, your query will keep on finding certain documents sequentially. This would require more time to find the documents. But if your documents have indexes, MongoDB would restrict make it specific the number of documents to be searched within your collection.

In the MongoDB to create an index, the ensureIndex() method is used.

Syntax:

db.collection_name.ensureIndex( {KEY:1, KEY:2} )

Here in the above syntax, the key is a field name that you want to make an index and it will be 1 and -1 to make it on ascending or descending order.

Suppose we have the field name PhoneNo in the user's collection and you want to index it, then you have to type the syntax in this way:

Example:

db.Users.ensureIndex({"PhoneNo":1})

MongoDB ensureIndex method is Deprecated since version 3.0.0 and db.collection.ensureIndex() is now an alias for db.collection.createIndex().

Syntax:

db.collection_name.createIndex( {KEY:1, KEY:2} )

Example:

db.Users.createIndex({"PhoneNo":1})