# Mongodb

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.

Below given 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) |

Below given 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
    }
  ]
}
```

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB there is no concept of relationship

# Advantages of MongoDB over RDBMS

- Schema less: MongoDB is document database in which one collection holds different documents. Number of fields, content and size of the document can be differ from one document to another.
- Structure of a single object is clear
- No complex joins
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL
- Tuning
- Ease of scale-out: MongoDB is easy to scale
- Conversion / mapping of application objects to database objects not needed
- Uses internal memory for storing the (windowed) working set, enabling faster access of data

# Why should use MongoDB

- Document Oriented Storage : Data is stored in the form of JSON style documents
- Index on any attribute
- Replication & High Availability
- Auto-Sharding
- Rich Queries
- Fast In-Place Updates
- Professional Support By MongoDB

# Where should use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure

- User Data Management
- Data Hub

**_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you didn't provide then MongoDB provide 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.

Data Modelling in Mongo

Data in MongoDB has a flexible schema. 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.

# Some considerations while designing schema in MongoDB

- Design your schema according to user requirements.
- Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
- Duplicate the data (but limited) because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases.
- Do complex aggregation in the schema

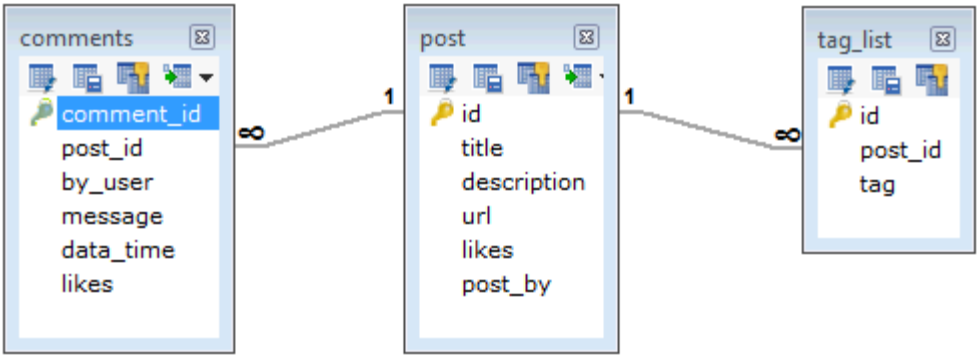
# Example

Suppose a client needs a database design for his blog website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.

- Every post has the unique title, description and url.
- Every post can have one or more tags.
- Every post has the name of its publisher and total number of likes.
- Every Post have comments given by users along with their name, message, data-time and likes.

- On each post there can be zero or more comments.

In RDBMS schema design for above requirements will have minimum three tables.




While in MongoDB schema design will have one collection post and has the following structure:

```
{
  _id: POST_ID
  title: TITLE_OF_POST,
  description: POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    },
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
```

```
    }
  ]
}
```

So while showing the data, in RDBMS you need to join three tables and in mongodb data will be shown from one collection only.

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:

## Example:

If you want to create 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, then 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 atleast 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.

# 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 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
```

```
>
```

## Create collection:

# 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 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 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 collection 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 fields Default value is false. |

| | | |
|---|---|---|
| size | number | (Optional) Specifies a maximum size in bytes for a capped collection. If **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
```

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
>
```

# 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

# The insert() Method

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

## Syntax

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 previous tutorial. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert document into it.

In the inserted document if we don't specify the _id parameter, then MongoDB assigns an 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 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.

# The find () Method

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

### Syntax

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 reruns 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 **AND** condition. Basic syntax of**AND** is shown below −

```
>db.mycol.find({key1:value1, key2:value2}).pretty()
```

## Example

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

```
>db.mycol.find({"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. Basic syntax of **OR** is shown below −

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

## Example

Below given 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

Below given example will show the documents that have likes greater than 100 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's **update()** and **save()** methods are used to update document into a collection. The update() method update 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 values in the existing document.

## Syntax

Basic syntax of **update()** method is as follows

```
>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

## Example

Consider the mycol collection has 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 single document, to update multiple you need to set a paramter '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 save() method

## Syntax

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 '5983548781331adf45ec7'

```
>db.mycol.save(
  {
    "_id" : ObjectId(5983548781331adf45ec7), "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"}
>
```

# The remove() Method

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

1. **deletion criteria :** (Optional) deletion criteria according to documents will be removed.

2. **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 collectioin has 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 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()
>
```

In mongodb projection meaning is selecting only 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 list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the field.

## Syntax:

Basic syntax of **find()** method with projection is as follows

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

## Example

Consider the collection myycol 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 quering 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

For further reading mongodb CRUD operations goto https://docs.mongodb.org/v3.0/applications/crud/