# MONGODB - QUICK GUIDE

# 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.

## Sample document

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
      }
   ]
}
```

# INSTALL MONGODB ON WINDOWS

To install the MongoDB on windows, first doownload the latest release of MongoDB from http://www.mongodb.org/downloads

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

Now open command prompt and run the following command

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

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

In command prompt navigate to the bin directory present into 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 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
```

## CREATE DATABASE

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 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.

## DROP DATABASE

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:

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 }
>
```

## CREATE COLLECTION

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

## Syntax:

Basic syntax of **createCollection** method 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
```

## List of options

| Field | Type | Description |
|---|---|---|
| capped | Boolean | *Optional* If true, enables a capped collection. Capped collection is a collection fixed size collecction 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 **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.

## Syntax :

```
>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
>
```

# DROP COLLECTION

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:

Below given example will drop the collection with the name **mycollection**

```
>use mydb
switched to db mydb
>db.mycollection.drop()
true
>
```

# INSERT DOCUMENT

To insert data into MongoDB collection, you need to use MongoDB's **insert** 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
      }
   ]
}
])
```

## QUERY DOCUMENT

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. 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* ":" *tutorialspoint* " .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** $by = ' tutorialspoint ' ORtitle = ' MongoDBOverview '$**'**

```
>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"
}
>
```

## UPDATE DOCUMENT

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

### MongoDB Update method

### Syntax:

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

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

### 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 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

**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"}
>
```

## DELETE DOCUMENT

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()
>
```

## MONGODB PROJECTION

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.

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 filed 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

## LIMIT DOCUMENTS

### MongoDB Limit Method

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

### Syntax:

Basic syntax of **limit** method is as follows

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

### 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 only 2 documents while quering the document.

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

If you don't specify 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 used to skip number of documents.

### Syntax:

Basic syntax of **skip** method is as follows

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

### Example:

Following example will only display only second document.

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

Please note default value in **skip** method is 0

# SORTING DOCUMENTS

To sort documents in MongoDB, you need to use **sort** method. **sort** method accepts a document containing list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

### Syntax:

Basic syntax of **sort** method is as follows

```
>db.COLLECTION_NAME.find().sort({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 documents sorted by title in descending order.

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

Please note if you don't specify the sorting preference, then **sort** method will display documents in ascending order.

# MONGODB INDEXING

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require the mongod to process a large volume of data.

Indexes are special data structures, that store a small portion of the data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in index.

To create an index you need to use ensureIndex method of mongodb.

## Syntax:

Basic syntax of **ensureIndex** method is as follows

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

Here key is the name of filed 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})
>
```

**ensureIndex** method also accepts list of options *whichareoptional*, whose list is given below:

| Parameter | Type | Description |
|---|---|---|
| background | Boolean | Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**. |
| unique | Boolean | Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**. |
| name | string | The name of the index. If unspecified, MongoDB generates an |

| | | |
|---|---|---|
| | | index name by concatenating the names of the indexed fields and the sort order. |
| dropDups | Boolean | Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**. |
| sparse | Boolean | If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations *particularlysorts*. The default value is **false**. |
| expireAfterSeconds | integer | Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection. |
| v | index version | The index version number. The default index version depends on the version of mongod running when creating the index. |
| weights | document | The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score. |
| default_language | string | For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **english**. |
| language_override | string | For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language. |

## MONGODB AGGREGATION

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In sql count $*$ and with group by is an equivalent of mongodb aggregation. For the aggregation in mongodb you should use **aggregate** method.

### Syntax:

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 that how many tutorials are written by each user then you will use **aggregate** method as shown below:

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "tutorials point",
         "num_tutorial" : 2
      },
      {
         "_id" : "tutorials point",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
>
```

Sql equivalent query for the above use case will be **select by_user, count $*$ from mycol group by by_user**

In the above example we have grouped documents by field **by_user** and on each occurance of by_user previous value of sum is incremented. There is a list available aggregation expressions.

| Expression | Description | Example |
|---|---|---|
| $sum | Sums up the defined value from all documents in the collection. | db.mycol.aggregate $[\$group: {}_{i}d: "\$by_{u}ser", num_{t}utorial: \$sum: "\$likes"]$ |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate $[\$group: {}_{i}d: "\$by_{u}ser", num_{t}utorial: \$avg: "\$likes"]$ |
| $min | Gets the minimum of the corresponding values from all documents in the collection. | db.mycol.aggregate $[\$group: {}_{i}d: "\$by_{u}ser", num_{t}utorial: \$min: "\$likes"]$ |
| $max | Gets the maximum of the corresponding values from all documents in the collection. | db.mycol.aggregate $[\$group: {}_{i}d: "\$by_{u}ser", num_{t}utorial: \$max: "\$likes"]$ |
| $push | Inserts the value to an array in the resulting document. | db.mycol.aggregate $[\$group: {}_{i}d: "\$by_{u}ser", url: \$push: "\$url"]$ |
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate $[\$group: {}_{i}d: "\$by_{u}ser", url: \$addToSet: "\$url"]$ |
| $first | Gets the first document from the | db.mycol.aggregate |

| | | |
|---|---|---|
| | source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | $[\$group:{}_id:"\$by_user", first_url:\$first:"\$url"]$ |
| $last | Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate $[\$group:{}_id:"\$by_user", last_url:\$last:"\$url"]$ |

# MONGODB REPLICATION

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers, replication protects a database from the loss of a single server. Replication also allows you to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

## Why Replication?

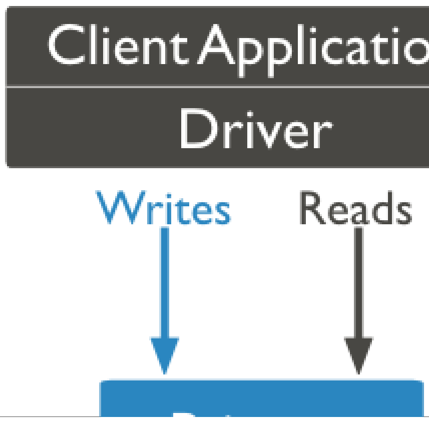
- To keep your data safe
- High $24 * 7$ availability of data
- Disaster Recovery
- No downtime for maintenance *likebackups, indexrebuilds, compaction*
- Read scaling *extracopiestoreadfrom*
- Replica set is transparent to the application

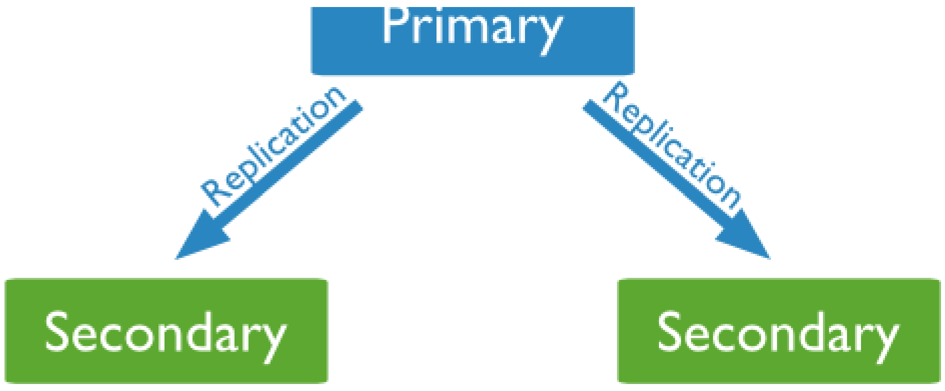
## How replication works in MongoDB

MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set. In a replica one node is primary node that receives all write operations. All other instances, secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

1. Replica set is a group of two or more nodes *generallyminimum3nodesarerequired*.
2. In a replica set one node is primary node and remaining nodes are secondary.
3. All data replicates from primary to secondary node.
4. At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
5. After the recovery of failed node, it again join the replica set and works as a secondary node.

A typical diagram of mongodb replication is shown in which client application always interact with primary node and primary node then replicate the data to the secondary nodes.





## Replica set features

- A cluster of N nodess
- Anyone node can be primary
- All write operations goes to primary
- Automatic failover
- Automatic Recovery
- Consensus election of primary

## Set up a replica set

In this tutorial we will convert standalone mongod instance to a replica set. To convert to replica set follow the below given steps:

- Shutdown already running mongodb server.
- Now start the mongodb server by specifying **--replSet** option. Basic syntax of **--replSet** is given below:

  ```
  mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet
  "REPLICA_SET_INSTANCE_NAME"
  ```

  ### Example

  ```
  mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
  ```

  It will start a mongod instance with the name rs0, on port 27017

- Now start the command prompt and connect to this mongod instance.
- In mongo client issue the command **rs.initiate** to initiate a new replica set.
- To check the replica set configuration issue the command **rs.conf**.
- To check the status of replica sete issue the command **rs.status**.

# MONGODB CREATE BACKUP

## Dump MongoDB Data

To create backup of database in mongodb you should use **mongodump** command. This command will dump all data of your server into dump directory. There are many options available by which you can limit the amount of data or create backup of your remote server.

## Syntax:

Basic syntax of **mongodump** command is as follows

```
>mongodump
```

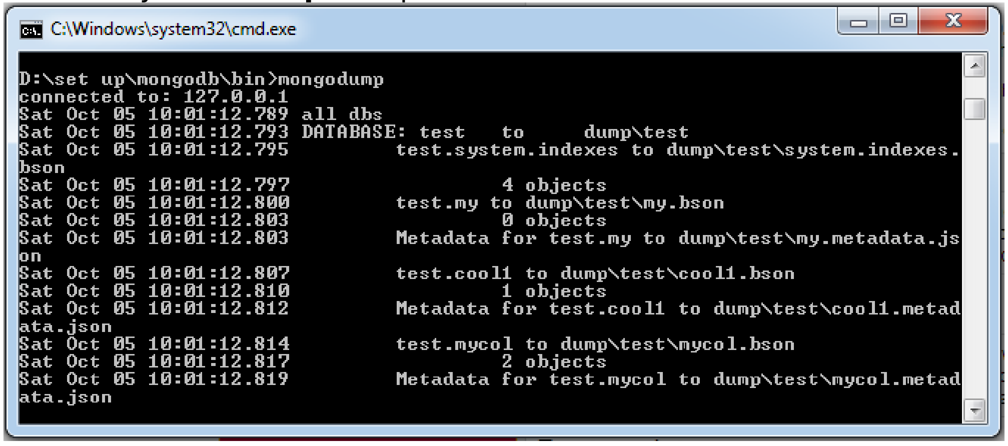
## Example

Start your mongod server. Assuming that your mongod server is running on localhost and port 27017. Now open a command prompt and go to bin directory of your mongodb instance and type the command **mongodump**

Consider the mycol collectioin has following data.

```
>mongodump
```

The command will connect to the server running at **127.0.0.1** and port **27017** and back all data of the server to directory **/bin/dump/**. Output of the command is shown below:



```
D:\set up\mongodb\bin>mongodump
connected to: 127.0.0.1
Sat Oct 05 10:01:12.789 all dbs
Sat Oct 05 10:01:12.793 DATABASE: test   to     dump\test
Sat Oct 05 10:01:12.795         test.system.indexes to dump\test\system.indexes.
bson
Sat Oct 05 10:01:12.797                  4 objects
Sat Oct 05 10:01:12.800         test.my to dump\test\my.bson
Sat Oct 05 10:01:12.803                  0 objects
Sat Oct 05 10:01:12.803         Metadata for test.my to dump\test\my.metadata.js
on
Sat Oct 05 10:01:12.807         test.cool1 to dump\test\cool1.bson
Sat Oct 05 10:01:12.810                  1 objects
Sat Oct 05 10:01:12.812         Metadata for test.cool1 to dump\test\cool1.metad
ata.json
Sat Oct 05 10:01:12.814         test.mycol to dump\test\mycol.bson
Sat Oct 05 10:01:12.817                  2 objects
Sat Oct 05 10:01:12.819         Metadata for test.mycol to dump\test\mycol.metad
ata.json
```

There are a list of available options that can be used with the **mongodump** command.

This command will backup only specified database at specified path

| Syntax | Description | Example |
|---|---|---|
| mongodump --host HOST_NAME --port PORT_NUMBER | This commmand will backup all databases of specified mongod instance. | mongodump --host tutorialspoint.com --port 27017 |
| mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY | | mongodump --dbpath /data/db/ --out /data/backup/ |
| mongodump --collection COLLECTION --db DB_NAME | This command will backup only specified collection of specified database. | mongodump --collection mycol --db test |

## Restore data

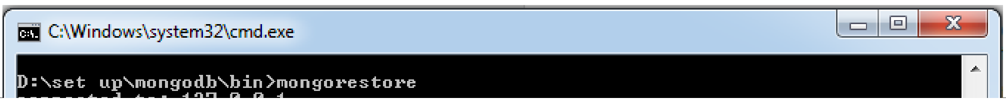
To restore backup data mongodb's **mongorestore** command is used. This command restore all of the data from the back up directory.

## Syntax

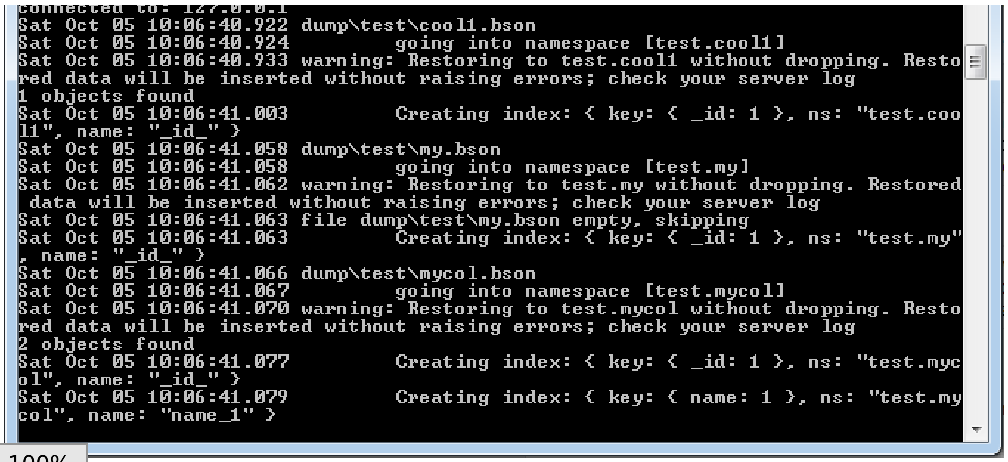
Basic syntax of <**mongorestore** command is

```
>mongorestore
```

Output of the command is shown below:



```
D:\set up\mongodb\bin>mongorestore
```



```
connected to: 127.0.0.1
Sat Oct 05 10:06:40.922 dump\test\cool1.bson
Sat Oct 05 10:06:40.924         going into namespace [test.cool1]
Sat Oct 05 10:06:40.933 warning: Restoring to test.cool1 without dropping. Resto
red data will be inserted without raising errors; check your server log
1 objects found
Sat Oct 05 10:06:41.003         Creating index: { key: { _id: 1 }, ns: "test.coo
l1", name: "_id_" }
Sat Oct 05 10:06:41.058 dump\test\my.bson
Sat Oct 05 10:06:41.058         going into namespace [test.my]
Sat Oct 05 10:06:41.062 warning: Restoring to test.my without dropping. Restored
 data will be inserted without raising errors; check your server log
Sat Oct 05 10:06:41.063 file dump\test\my.bson empty, skipping
Sat Oct 05 10:06:41.063         Creating index: { key: { _id: 1 }, ns: "test.my"
, name: "_id_" }
Sat Oct 05 10:06:41.066 dump\test\mycol.bson
Sat Oct 05 10:06:41.067         going into namespace [test.mycol]
Sat Oct 05 10:06:41.070 warning: Restoring to test.mycol without dropping. Resto
red data will be inserted without raising errors; check your server log
2 objects found
Sat Oct 05 10:06:41.077         Creating index: { key: { _id: 1 }, ns: "test.myc
ol", name: "_id_" }
Sat Oct 05 10:06:41.079         Creating index: { key: { name: 1 }, ns: "test.my
col", name: "name_1" }
```

Processing math: 100%