MongoDB - Create Database

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":“robo2"})
```

```
>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 - Drop 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 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 - Create Collection

The createCollection() Method

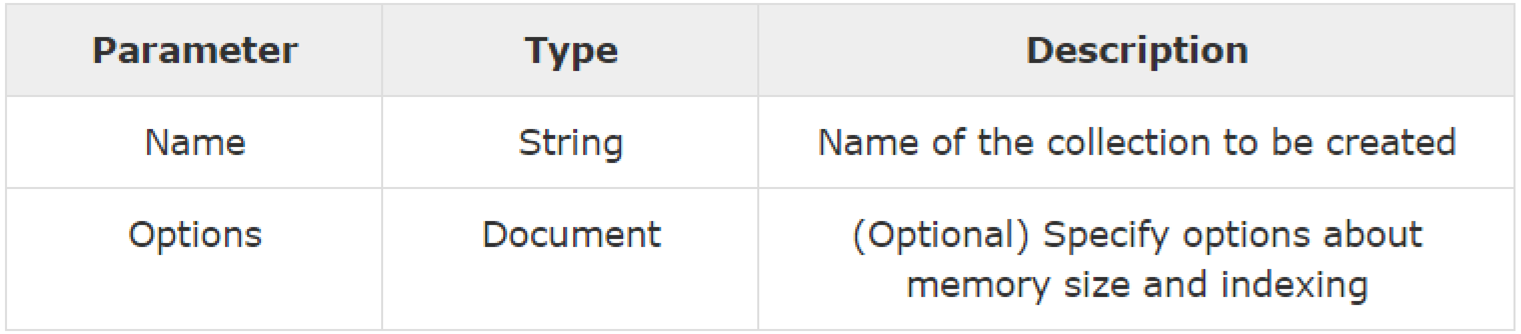
MongoDB db.createCollection(name, options) is used to create collection.

Syntax

Basic syntax of **createCollection()** command is as follows −

```
db.createCollection(name, options)
```

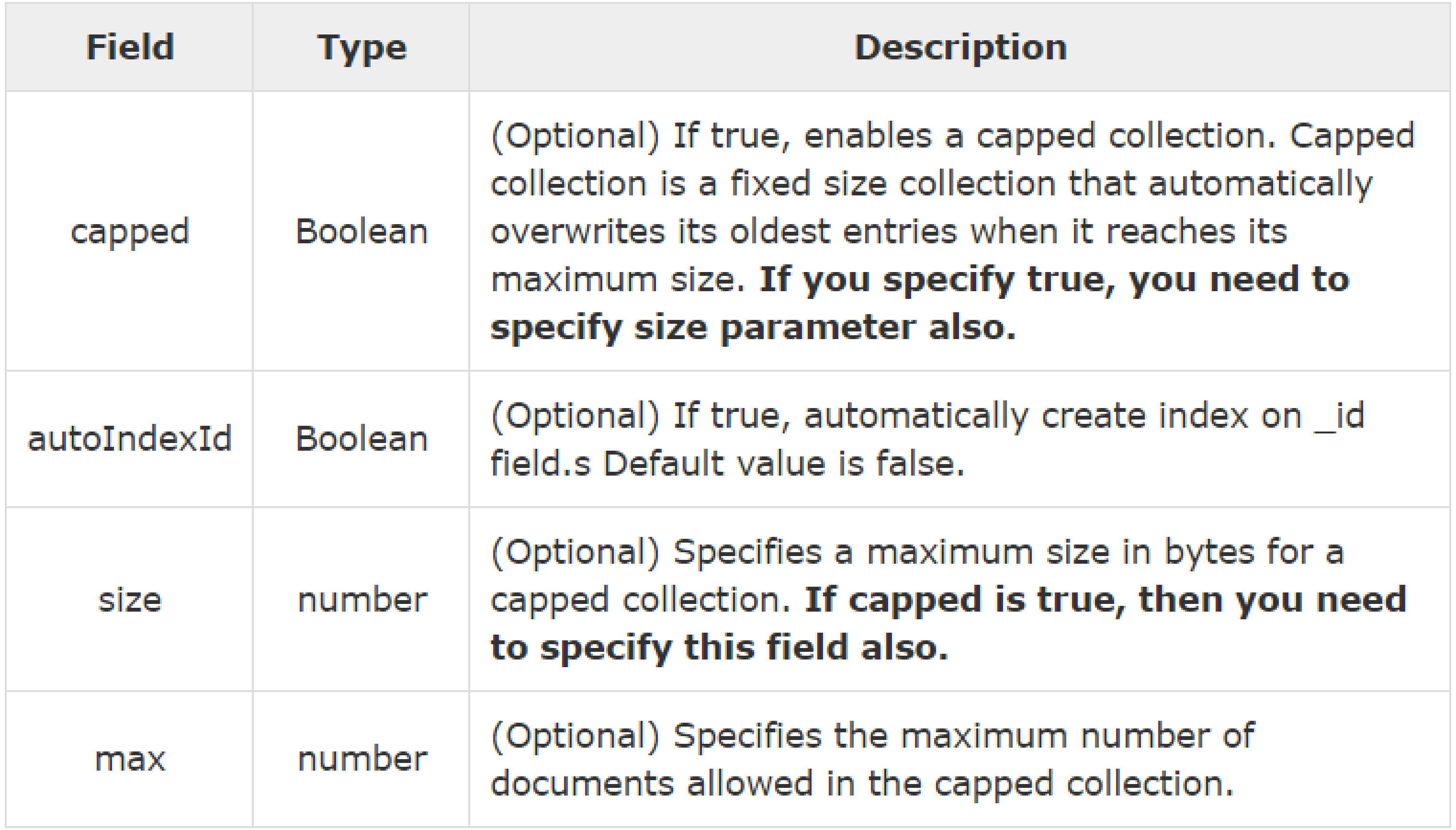
- ✓ In the command, name is name of collection to be created.
- ✓ Options is a document and is used to specify configuration of collection.



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 }
>
```

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

```
>db.tutorial.insert({"name" : “MongoDB"})

>show collections
mycol
mycollection
tutorials
>
```

MongoDB - Drop Collection

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
tutorials
>
```

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
tutorials
>
```

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

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 − 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 - Insert Document

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',
   url: 'http://www.facebook.com/kanagarajg',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
})
```

Note:
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.

```
>db.post.insert([
  {
    title: 'MongoDB Overview',
    description: 'MongoDB is no sql database',
    by: 'tutorials',
    url: ' 'http://www.facebook.com/kanagarajg ',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
  },

  {
    title: 'NoSQL Database',
    description: "NoSQL database doesn't have tables",
    by: 'tutorials',
    url: ' 'http://www.facebook.com/kanagarajg ',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 20,
    comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2013,11,10,2,35),
         like: 0
      }
    ]
  }
])
```

Note:

- 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 - Query Document

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()
```

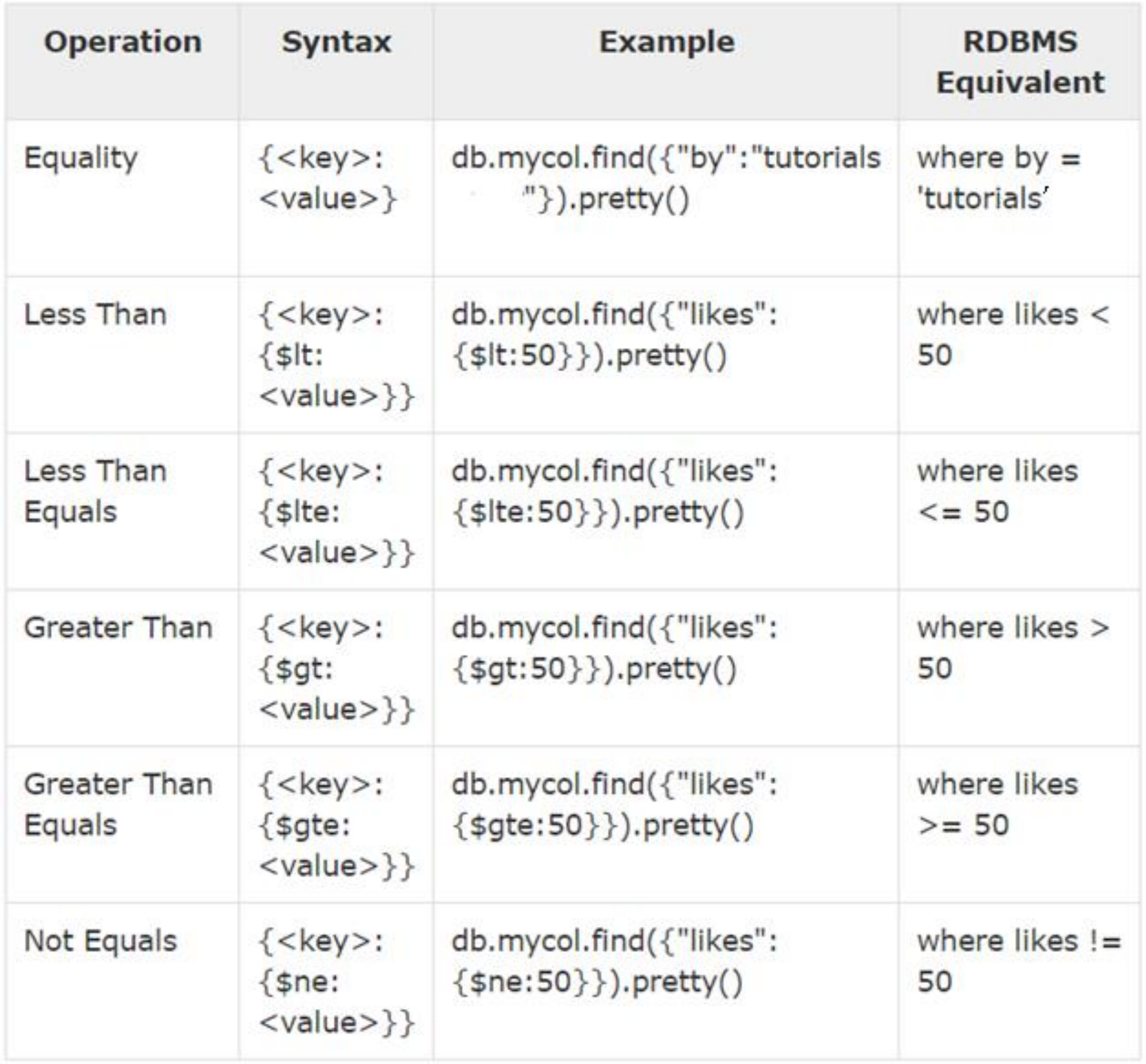
```
>db.mycol.find().pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials",
   "url": "http://www.facebook.com/kanagarajg ",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

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

```
> db.post.findOne()
{
        "_id" : ObjectId("5d3169dd1f5f2793a8ffb613"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no sql database",
        "by" : "tutorials",
        "url" : "http://www.facebook.com/kanagrajg",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

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"}).pretty()	where by = 'tutorials'
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' 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",
   "url": " http://www.facebook.com/kanagarajg ",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

For the above given example, equivalent where clause will be **' where by = 'tutorials' 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' 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.facebook.com/kanagarajg ",
   "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' 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.facebook.com/kanagarajg ",
   "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.

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

MongoDB - Delete Document

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(DELLETION_CRITTERIA)
```

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 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 Overview"}
>
```

Remove Only One

The **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(DELLETION_CRITTERIA)
```

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 - 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 - Limit Records

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 - Sort Records

The sort() Method

- To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a 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

The 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 the 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 the documents in ascending order.

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.

The aggregate() Method

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
},
```

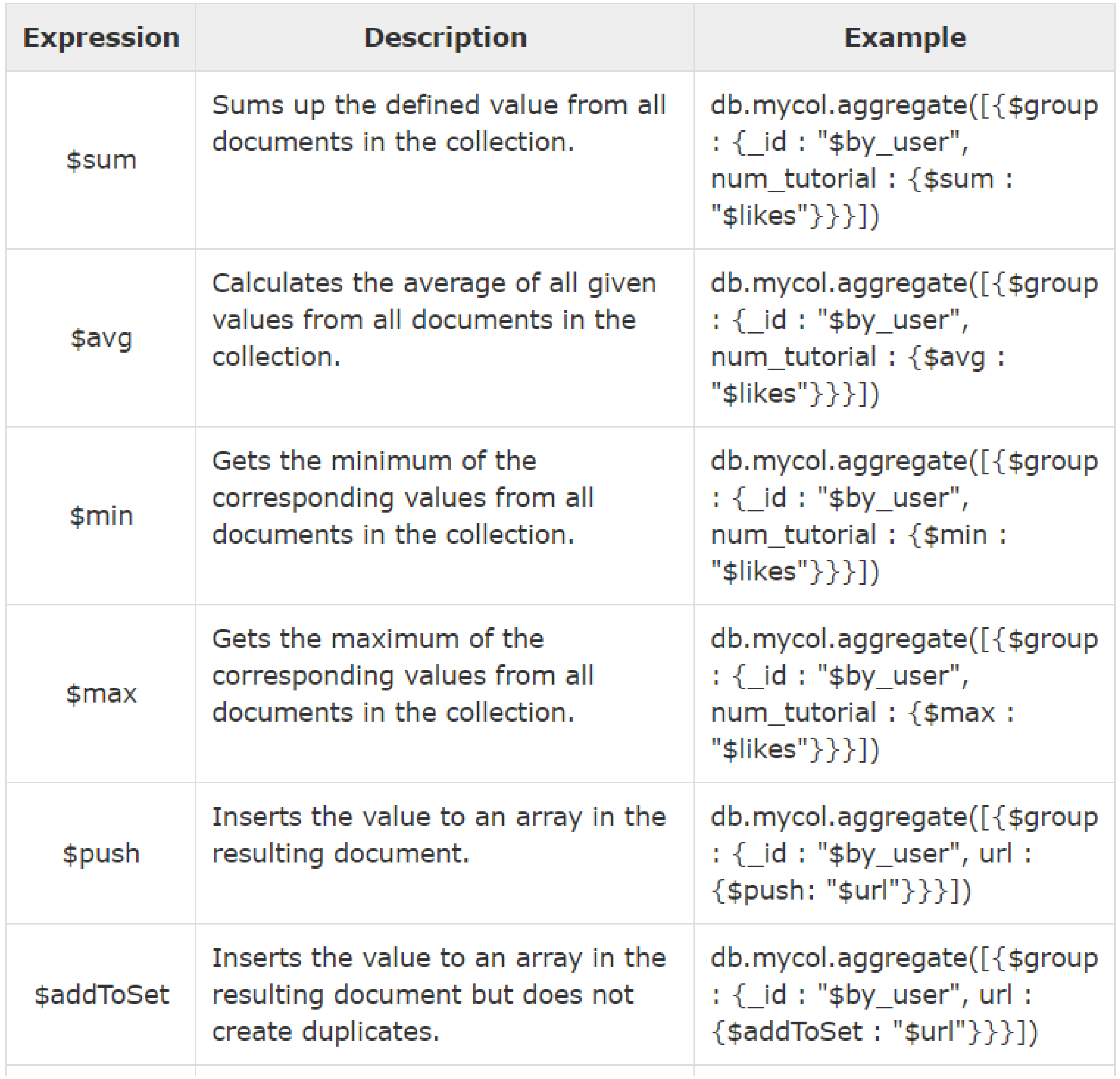
Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method –

```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{
   "result" : [
      {
         "_id" : "tutorials point",
         "num_tutorial" : 2
      },
      {
         "_id" : "Neo4j",
         "num_tutorial" : 1
      }
   ],
   "ok" : 1
}
>
```

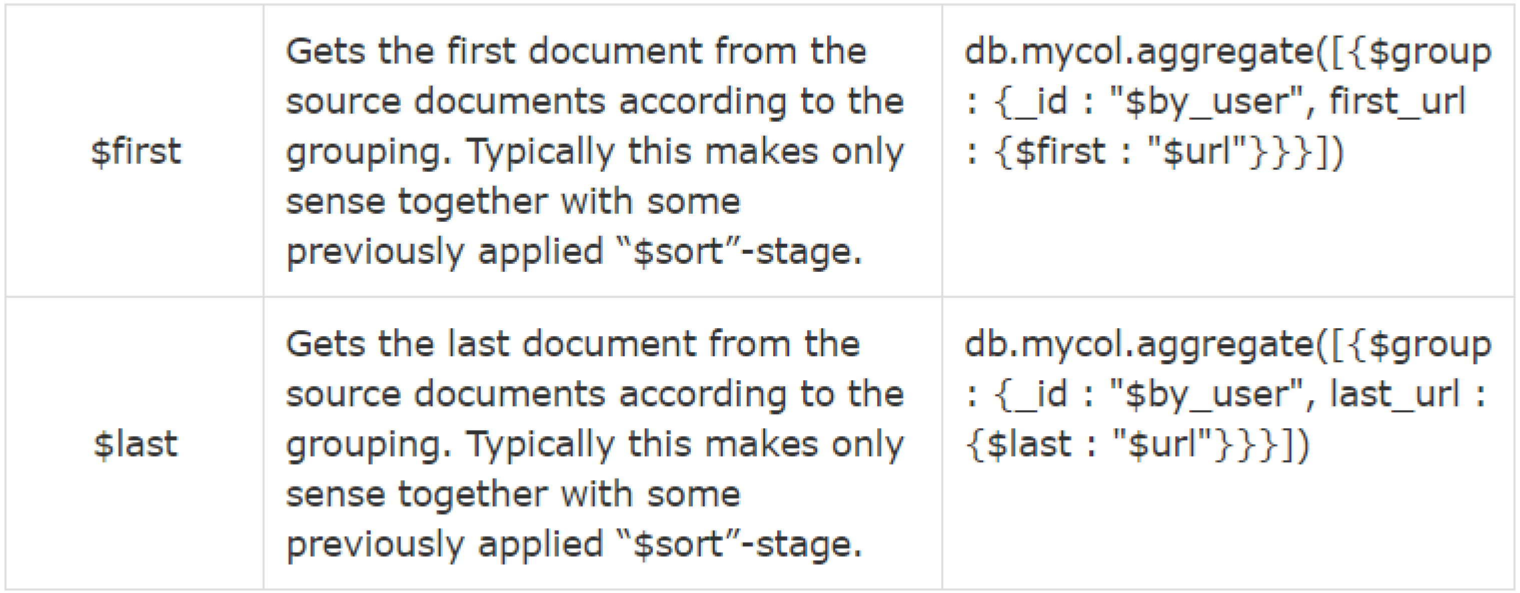
Sql equivalent query for the above use case will be ***select by_user, count(*) from mycol group by by_user****.*

Herewe have grouped documents by field by_user and on each occurrence of by_user previous value of sum is incremented.

list of available aggregation expressions:



Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])



$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", 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"}}}])