* MongoDB Datatypes
  + String
  + Integer
  + Boolean
  + Double
  + Min / Max keys - This type is used to compare a value against the lowest and highest BSON(BinaryJSON) elements
  + Arrays
  + Timestamp
  + Object - This datatype is used for embedded documents
  + Null
  + Symbol - it's generally reserved for languages that use a specific symbol type
  + Date
  + Object ID - This datatype is used to store the document’s ID
  + Binary Data
  + Code - This datatype is used to store JavaScript code into the document
  + Regular Expression
* **insert data into MongoDB collection** - db.COLLECTION\_NAME.insert(document)
  + 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.

>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

})

* + 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 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.COLLECTION\_NAME.save(document)** also
* **query data from MongoDB collection** - 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.
  + findOne() method, that returns only one document.

>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"

}

>

* **RDBMS Where Clause Equivalents in MongoDB**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 - if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

## Example

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

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "http://www.tutorialspoint.com",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

## OR in MongoDB

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

## Example

>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"

}

>

## AND and OR together

>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 into a collection - update() and save()

## 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

>db.COLLECTION\_NAME.update(SELECTION\_CRITERIA, UPDATED\_DATA)

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

## 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})

## remove a document from the collection - db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA, justOne)

>db.mycol.remove({'title':'MongoDB 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()

## Projection

## Projection means selecting only the necessary data rather than selecting whole of the data of a document

## 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

>db.mycol.find({},{"title":1,\_id:0})

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

{"title":"Tutorials Point Overview"}

>

## Limit Records

## limit() method accepts one number type argument, which is the number of documents that you want to be displayed

>db.mycol.find({},{"title":1,\_id:0}).limit(2)

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

>

## Skip Records - skip the number of documents

## The default value in skip() method is 0.

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

{"title":"NoSQL Overview"}

>

## Sort

## sort() method accepts a document containing a list of fields along with their sorting order.

## 1 is used for ascending order while -1 is used for descending order.

## If you don't specify the sorting preference, then sort() method will display the documents in ascending order.

>db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

{"title":"Tutorials Point Overview"}

{"title":"NoSQL Overview"}

{"title":"MongoDB Overview"}

>

## Indexing

## 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 the index

## To create an index you need to use ensureIndex() method

>db.mycol.ensureIndex({"title":1})

## ensureIndex() method you can pass multiple fields, to create index on multiple fields.

>db.mycol.ensureIndex({"title":1,"description":-1})

## Aggregation and Pipeline

## Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result

## There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline)

## Create Dump

>mongodump

## Restore backup

>mongorestore

## Replication - MongoDB achieves replication by the use of replica set. A replica set is a group of mongod instances that host the same data set.

* + Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
  + In a replica set, one node is primary node and remaining nodes are secondary.
  + All data replicates from primary to secondary node.
  + At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
  + After the recovery of failed node, it again join the replica set and works as a secondary node
* Sharding
  + Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth.
  + As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.