MongoDB is a document-oriented database and each **document** has its own structure. Unlike a RDBMS in which each record must conform to the structure of its table, each **document** in MongoDB can have a different structure; you don't have to define a schema for **documents** before saving them in the database.

MongoDB groups document objects into **collections**. You can think of a collection as a table like you would create in a RDBMS, but the difference as I said before is that they won't force you to define a schema before you can store something.

With MongoDB, you can embed a **document** inside another one, which is really useful for cases where there is a one-to-one relationship. In a typical RDBMS you'd need to create two tables and link them together with a foreign key to achieve the same result. MongoDB doesn't support joins, which some people see as a con. But if you organize your data correctly then you'll find you don't need joins, which is a pro since you'll benefit from very high performance.

It's worth mentioning the aim of MongoDB and NoSQL isn't to kill off RDBMS. RDBMSs are still a very good solution for most of the development world's needs, but they do have their weaknesses, most noticeably the need to define a **rigid schema** for your data which is one problem NoSQL tries to solve.

cls => to clear mongodb command prompt screen

Reference: http://www.tutorialspoint.com/mongodb/index.htm

#0>.Important:

a>. Mongodb status| start | stop command
$ sudo service mongod (status| start | stop | restart)

b>.Mongodb configuration file location:
/etc/mongod.conf

Note: in file mongod.conf you can find location of logpath, dbpath , ...etc

c>. monodb data store directory: /var/lib/mongo

d>. For every database it will create two file.
like for database having name 'niit_tech', it will create two files i>. niit_tech.0, ii>. niit_tech.ns

#1>. Linux connect to mongodb server:

$ mongo 127.0.0.1:27017

#2>. 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.

```
$ use DATABASE_NAME
```

#3>. To check your currently selected database use the command db

```
$ db
```

#4>. if you want to check your databases list, then use the command show dbs.

```
$ show dbs
```

NOTE: Your created database (suppose 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
```

#5>. Drop Database:

```
$ db.dropDatabase()
```

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

#6>. Create tables/ collections for a particular db:

a>. $ use DATABASE_NAME
b>. $ 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

#7>. Show all collection of database:

```
$ show collections
```

#8>. Drop collection:

db.COLLECTION_NAME.drop()

NOTE:First, check the available collections into your database mydb

$ use mydb
switched to db mydb

$ show collections
mycol
mycollection
system.indexes
tutorialspoint

$db.mycollection.drop()

#9>.To get stats about mongodb server type the command db.stats() in mongodb client. This will show the database name, number of collection and documents in the database.

$ db.stats()

#10>. Create record/document in mongodb collection/table:

$ db.movie.insert({"fname":"girish","lname":"Thankur","village":"Morhar", "district":"Muzaffarpur" })

>> NOTE: here 'movie' is table/collection name.

NOTE: To insert multiple documents in single query, you can pass an array of documents in insert() command.

```
 $ db.movie.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
  }
  ]
}
```

NOTE:
a>. In the inserted document if we don't specify the _id parameter, then MongoDB assigns an unique ObjectId for this document.
b>. 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)

#11>. Query data from mongodb collection:

Syntax: **db.COLLECTION_NAME.find()**

NOTE:

a>. find() method will display all the documents in a non structured way.
b>. To display the results in a formatted way, you can use pretty() method.
$ db.mycol.find().pretty()

c>. Apart from find() method there is findOne() method, that retuns only one document.

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"}).pre	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 (&&) Operator on mongodb:

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-

```
$ 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"
}
```

NOTE:
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

To query documents based on the OR condition, you need to use **$or** keyword. Basic syntax of OR is shown below:
Syntax:

```
$ 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"
}
```

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"
}
```

#12>. MongoDB - Update Document

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.

>> The update() method updates values in the existing document.

Syntax: `db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)`

Example:

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

>> The save() method replaces the existing document with the new document passed in save() method

Syntax: `db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})`

```
 $ db.mycol.save(
  {
  "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic", "by":"Tutorials Point"
  }
)
```

#13>. MongoDB - 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:

```
$ db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Example:

```
$db.mycol.remove({'title':'MongoDB Overview'})
```

NOTE: 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)
```

NOTE: 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()
```

#14>. 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:

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

Following example will display the title of the document while querying the document.

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

NOTE: 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

#15>. MongoDB - Limiting Records

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:
$ db.COLLECTION_NAME.find().limit(NUMBER)
Example:
$ db.mycol.find({},{"title":1,_id:0}).limit(2)

NOTE: 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:

$ db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)

Example:

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

#16>. MongoDB - Sorting Records

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:
$ db.COLLECTION_NAME.find().sort({KEY:1})

Example:

Following example will display the documents sorted by title in descending order.

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

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

#17>. MongoDB - Indexing

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

Syntax:
$ db.COLLECTION_NAME.ensureIndex({KEY:1})

NOTE: 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:

a>. **$ db.mycol.ensureIndex({"title":1})**
b>. In ensureIndex() method you can pass multiple fields, to create index on multiple fields.
$ **db.mycol.ensureIndex({"title":1,"description":-1})**

NOTE: ensureIndex() method also accepts list of options (which are optional), 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 (particularly sorts). 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 vers	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.

#18>. 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.

Syntax:

$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" : "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

In the above example we have grouped documents by field by_user and on each occurrence 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 : {_id : "$by_user", num_tutorial : {$sum : "$likes"
$avg	Calculates the average of all given values from all documents in the collectior	db.mycol.aggregate ([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}
$min	Gets the minimum of the corresponding values from all documents in the collectior	db.mycol.aggregate ([{$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"}}}])

NOTE: Pipeline Concept

In UNIX command shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also support same concept in aggregation framework. There is a set of possible stages and each of those is taken a set of documents as an input and is producing a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn again be used for the next stage an so on.

Possible stages in aggregation framework are following:

$project: Used to select some specific fields from a collection.
$match: This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
$group: This does the actual aggregation as discussed above.
$sort: Sorts the documents.
$skip: With this it is possible to skip forward in the list of documents for a given amount of documents.
$limit: This limits the amount of documents to look at by the given number starting from the current position.s
$unwind: This is used to unwind document that are using arrays. when using an array the data is kind of pre-joinded and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

#19> MongoDB - Create Backup

NOTE: you don't have to login to mongodb shell, simply run following command on your shell

$ mongodump

NOTE: above command will take backup of all database of mongodb server

$ mongodump -db ankit_test_db

NOTE: Above command will take backup of **'ankit_test_db'** database

$ mongodump -collection movie -db ankit_test_db

NOTE: Above command will take backup of **'movie'** collection of **'ankit_test_db'** database

$ mongodump --host 127.0.0.1 --port 27017 -collection movie -db ankit_test_db

NOTE: Above command will take backup of **'movie'** collection of **'ankit_test_db'** database where **127.0.0.1** is host name and **27017** is port number.

$ mongodump --host 127.0.0.1 --port 27017 -collection movie -db ankit_test_db -out /home/ankitk/dbbackup/ankit

NOTE: Above command will take backup of **'movie'** collection of **'ankit_test_db'** database where **127.0.0.1** is host name ,**27017** is port number and '**/home/ankitk/dbbackup/ankit**' is the path where backup is stored.

#20>. MongoDB - Restore data

NOTE: goto directory where dump is sore and run following command:

$ mongorestore directory/database

Example:

$ mongorestore /home/ankitk/dbbackup/dump/ankit_test_db

NOTE: Above command will store **'ankit_test_db'** database base whose location is '**/home/ankitk/dbbackup/dump/**'