**What is MongoDB?**

* MongoDB is an open-source document database, and a leading NoSQL database.
* MongoDB is written in c++
* MongoDB is a cross-platform, document oriented database that provides high performance, high availability, and easy scalability.
* MongoDB works on the concept of collection and document.
* A single MongoDB server typically has multiple databases.

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) |
| **Database Server and Client** | |
| Mysql/Oracle | mongod |
| mysql/sqlplus | mongo |

**Advantages of MongoDB over RDBMS**

* Schema less : MongoDB is document database in which one collection holds different 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

Starting up MongoDB

In command prompt navigate to the bin directory present into the mongodb installation folder. Suppose my installation folder is **C:\”Program Files”\MongoDB\Server\3.2**

C:\Program Files\mongodb\Server\3.2>cd bin

C:\Program Files\mongodb\Server\3.2\bin>mongod.exe --dbpath "C:\ 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.

C:\Program Files\mongodb\Server\3.2\bin>mongo.exe

MongoDB shell version: 2.4.6

connecting to: test

This will show that mongodb is installed and run successfully. Next time when you run mongodb you need to issue only commands.

MongoDB Statistics

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.

**Create Database**

**1. Create Database:**

>use DATABASE\_NAME

EX: >use mydatabase

To check current database type:

>db

To check Database List:

>show dbs

**O/p:** local

As of now, the database “mydatabase” is not in this list. It is because we have no inserted anything in the database.

db.sometable.insert({“name”:”Peeyush Taori”})

show dbs

**Drop Database**

**Drop Database:**

* + db.dropDatabase()

Drops the current database which we are using.

**MongoDB - Create Collection**

**CreateCollection() Method:**

* + 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 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.

use test

db.createCollection(“firstCollection”)

show collections

db.createCollection("collectionWithOpts",{capped:true, autoIndexID:true, max:10000})

**Drop collection**

**Drop() method**

* + db.COLLECTION.drop()
* Used to drop A collection from the database
* drop() method will return true, if the selected collection is dropped successfully otherwise it will return false

**db.firstCollection.drop()**

**Data Types**

MongoDB supports many datatypes whose list is given below:

* **String** : This is 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** : ctimestamp. 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 binay data.
* **Code** : This datatype is used to store javascript code into document.
* **Regular expression** : This datatype is used to store regular expression

**Insert Document**

**Insert() Method**

* + db.COLLECTION\_NAME.insert(document)
    - used to insert and save document.
    - 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.

**db.firstCollection.insert({**

**\_id: "ObjectId(7df78ad8902c)",**

**title: 'MongoDB',**

**description: 'MongoDB is no sql database',**

**by: 'GL',**

**url: 'http://www.GLacademy.com',**

**tags: ['mongodb', 'database', 'NoSQL'],**

**likes: 100**

**})**

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:

4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer

To insert multiple documents, you pass array of documents

db.post.insert([

{

title: 'MongoDB',

description: 'MongoDB is no sql database',

by: 'GL',

url: 'http://www.GLacademy.com',

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

likes: 100

},

{

title: 'NoSQL Database',

description: "NoSQL database doesn't have tables",

by: 'GL',

url: 'http://www.GLacademy.com',

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

likes: 20,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2016,09,09,2,35),

like: 0

}

]

}

])

**QUERY DOCUMENT**

**Find() method**

* + db.COLECTION\_NAME.find()
    - Used to query data from the MongoDB collection

**Pretty() method**

* + db.firstCollection.find().pretty()
    - used to display the results in a formatted way.

db.post.findOne()

## 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.post.find({"by":"GL"}).pretty() | where by = 'GL' |
| Less Than | {<key>:{$lt:<value>}} | db.post.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.post.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.post.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.post.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.post.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

**AND in MongoDB**

* + **db.mycol.find({key1:value1, key2:value2}).pretty()**
    - In the find() method if you pass multiple keys by separating them by ',' then MongoDB treats it AND condition.

db.post.find({"by":"GL","title": "MongoDB"}).pretty()

**OR in MongoDB**

* To query documents based on the OR condition, you need to use **$or** keyword.

## db.post.find({$or:[{"by":"GL"},{"title": "MongoDB"}]}).pretty()

## db.post.find({"likes": {$gt:10}, $or: [{"by": "GL"},{"title": "MongoDB"}]}).pretty()

## Update() method

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

## 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.

## db.post.update({'title':'MongoDB'},{$set:{'title':'Updated MongoDB'}})

## db.post.find()

## db.post.update({'title':'MongoDB'},{$set:{'title':'Updated MongoDB'}},{multi:true})

## Save() Method

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

## The save() method replaces the existing document with the new.

## .document passed in save() method

## 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 just One 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.

**REMOVE()**

* + db.COLLECTION\_NAME.remove(DELETION\_CRITERIA)
  + db.post.remove({'title':'Updated MongoDB'})

**REMOVE ONLY ONE**

* + DB.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)
  + db.post.remove({'title':'Updated MongoDB'},1)

**REMOVE ALL DOCUMENTS**

* + db.firstCollection.remove({})
  + db.firstCollection.find

**MongoDB- Projection**

**Find() Method**

MongoDB's **find()** method, explained in [MongoDB Query Document](http://www.tutorialspoint.com/mongodb/mongodb_query_document.htm) 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.

>db.COLLECTION\_NAME.find({},{KEY:1})

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

**MongoDB- Limit Records**

**Limit() Method**

MongoDB's **find()** method, explained in [MongoDB Query Document](http://www.tutorialspoint.com/mongodb/mongodb_query_document.htm) 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.

* + db.COLLECTION\_NAME.find().limit(NUMBER)

**db.post.find({},{"title":1,\_id:0}).limit(5)**

**Sort() Method**

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.

* + db.COLLECTION\_NAME.find().sort({KEY:1})
  + **db.post.find({},{"title":1,\_id:0}).sort({"title":-1})**

**Indexing**

**EnsureIndex() method**

* + db.COLLECTION\_NAME.ensureIndex({KEY:1})
    - Here key is the name of field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.
    - db.post.ensureIndex({"title":1})

## 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.

## Aggregate() Method

## db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

## Importing data from files

## JSON files

## mongoimport --db test --collection restaurant --drop --file "D:\Peeyush\dataset.json"

## db.restaurant.find( { "borough": "Manhattan" } )

## db.restaurant.find( { "address.zipcode": "10075" } )

## CSV Files

## mongoimport --db test --collection loandata --type csv --drop --file "D:\Peeyush\loan\_data.csv" –headerline

## Export

## mongoexport --db test --collection restaurant --out "D:\Peeyush\restaurant.json"