MongoDB session

* Install Mongodb from

<https://www.mongodb.com/download-center>

Click download – select community server 🡪 windows 🡪download msi

* Run mongodb.msi file to install Mongodb
* Observe the installation in Program files
  + C:\Program Files\MongoDB\Server\3.x\bin
* Observe utilities here for mongo
  + Mongod, mongo, mongoimport, mongoexport, mongoldap, mongodump, mongooplog, mongoperf, mongorestore, mongostat
  + Set the path to environment variable (optional)
    - Open file explorer🡪 this PC 🡪 Right Click 🡪 properties 🡪 Advanced System settings 🡪 Environment variables 🡪 select Path 🡪 Edit 🡪

… ; C:\Program Files\MongoDB\Server\3.4\bin 🡪 ok 🡪 ok

* Create folder c:\data\db
* ..\> mongod help
* ..\> mongod

…….

Waiting for connection on port 27017

Customising prompt:

cmdCount = 1;

host = db.serverStatus().host;

prompt = **function**() {

**return** db +"@"+host+cmdCount+"$ ";

}

The prompt would then resemble the following:

test@myHost1$

Exit shell:

To exit the shell, type quit() or use the <Ctrl-C> shortcut.

Writing code in script.js and executing in mongo:

For Mongo shell help

>help

To display the db’s

>db

test

> show dbs

**Local 0.000GB**

**Database help()**

>db.help()

**Shows the complete information of current db**

**> db.stats()**

{

"db" : "test",

"collections" : 0,

"objects" : 0,

"avgObjSize" : 0,

"dataSize" : 0,

"storageSize" : 0,

"numExtents" : 0,

"indexes" : 0,

"indexSize" : 0,

"fileSize" : 0,

‘ "ok" : 1

}

**To create a database:**

>use customerdb

Switched to customerdb

>show dbs

Not showing as atleast one document must be added to show

>db.createCollection("customers")

{ "ok" : 1 }

>show collections

customers

>db.customers.insert({ fname:”murthy”,lname:”srirama”})

WriteResult({inserted:1})

To Query :

>db.customers.find() 🡪 Unformatted Output

>db.customers.find().pretty() 🡪 Formatted output

To drop database:

>db.dropDatabase()

{“dropped”:”mydb”,”ok”:1}

Check Now:

>show dbs

* To see the list of collections in the current database, use the show collections command:

**>show collections**

For help on collections

>db.collection.help()

**Data types:**

myObj =

{ "bool":true,"string":"sriram","number":122,"nullvalue":null,

"marks":[10,20,30,40]

}

> db.bar.insert(myObj)

WriteResult({ "nInserted" : 1 })

> db.bar.find().pretty()

**Configuring editor - Sublime Text 3.0:**

**Linking Sublime text with Mongo Shell:**

Open Sublime text 🡪 tools 🡪 Build System 🡪 new Build System

Write below code:

{

"selector": "source.js",

"shell":true,

"cmd":["mongo","<","$file"]

}

// save --> name --> mongo

// click tools --> build system --> mongo (Observe tick mark for mongo)

Add a new file source.js , tools🡪 build system 🡪 mongo

**Write code:**

var string="hello from sublime";

var x=Math.random();

print(string)

print(x)

----------------------------------------------

save source.js then Ctrl+b to build

// Press ctrl+b to build and see the output below through mongoshell

**Customising Mongo shell with .mongorc.js:**

(for writing multi lines code as only one command can be written in mongo shell )

Open C:\Users\DSR Murthy\.mongorc.js file which is already created by default and customise it: write below code:

EDITOR ="c:\\PROGRA~1\\SUBLIM~1\\sublime\_text.exe"

Save and exit

>var myobj={}

>edit myobj

{ }

Now write code in editor:

{ name :"murthy", job:"engineer"}

Save and (close) exit (note: Until we exit , shell prompt is locked)

> myobj

{{ name :"murthy", job:"engineer"}

>edit myobj

myDB 3$ (observe how prompt changes to current DB with which you are working.

Inserting Multiple records via Sublime text

Add new file : source.js and write below code:

// inserting multiple objects

var obj1={

name:"Kiran",

mailid:"kiran@gmail.com"

};

var obj2={

name:"Mallika",

mailid:"Mallika@gmail.com"

};

use customersdb

db.customers.insert([obj1,obj2]);

db.customers.find().pretty();

Press CTRL+B (build the system) and observe below output:

MongoDB shell version v3.4.7

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.4.7

switched to db myDB

BulkWriteResult({

"writeErrors" : [ ],

"writeConcernErrors" : [ ],

"nInserted" : 2,

"nUpserted" : 0,

"nMatched" : 0,

"nModified" : 0,

"nRemoved" : 0,

"upserted" : [ ]

})

{

"\_id" : ObjectId("59e7043315ded3dd2eb6cf7b"),

"name" : "Kiran",

"mailid" : "kiran@gmail.com"

}

{

"\_id" : ObjectId("59e7043315ded3dd2eb6cf7c"),

"name" : "Mallika",

"mailid" : "Mallika@gmail.com"

}

Bye

Finished 0.05sec

MongoDB 3.4 new Features:

insertOne(),

insertMany(),

updateOne(),

updateMany(),

deleteOne(),

deleteMany()

* [db.collection.update()](https://docs.mongodb.com/manual/reference/method/db.collection.update/#db.collection.update)
* [db.collection.updateOne()](https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne)
* [db.collection.updateMany()](https://docs.mongodb.com/manual/reference/method/db.collection.updateMany/#db.collection.updateMany)
* [db.collection.findAndModify()](https://docs.mongodb.com/manual/reference/method/db.collection.findAndModify/#db.collection.findAndModify)
* [db.collection.findOneAndUpdate()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndUpdate/#db.collection.findOneAndUpdate)
* [db.collection.findOneAndReplace()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndReplace/#db.collection.findOneAndReplace)
* [db.collection.save()](https://docs.mongodb.com/manual/reference/method/db.collection.save/#db.collection.save).
* [db.collection.bulkWrite()](https://docs.mongodb.com/manual/reference/method/db.collection.bulkWrite/#db.collection.bulkWrite).

db.inventory.insertMany([

{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },

{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },

{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },

{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },

{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }

]);

db.inventory.find( { status: { $in: [ "A", "D" ] } } )

db.collection.insertOne( { \_id: 10, calc: NumberLong("2090845886852") } )

db.collection.updateOne( { \_id: 10 },

{ $set: { calc: NumberLong("2555555000000") } } )

db.collection.updateOne( { \_id: 10 },

{ $inc: { calc: NumberLong(5) } } )

>db.inventory.find( { item : { $type: 10 } } )

> db.inventory.find( { item : { $exists: **false** } } )

>db.inventory.updateOne(

{ item: "paper" },

{

$set: { "size.uom": "cm", status: "P" },

$currentDate: { lastModified: **true** }

}

)

Update Many: in mongodb 3.2 onwards

db.inventory.updateMany(

{ "qty": { $lt: 50 } },

{

$set: { "size.uom": "in", status: "P" },

$currentDate: { lastModified: **true** }

}

)

**Replace : 3.4**

db.inventory.replaceOne(

{ item: "paper" },

{ item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }

)

* New features of Mongodb: 3.4
* [db.collection.findOneAndReplace()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndReplace/#db.collection.findOneAndReplace).
* [db.collection.findOneAndUpdate()](https://docs.mongodb.com/manual/reference/method/db.collection.findOneAndUpdate/#db.collection.findOneAndUpdate).
* [db.collection.findAndModify()](https://docs.mongodb.com/manual/reference/method/db.collection.findAndModify/#db.collection.findAndModify).
* [db.collection.save()](https://docs.mongodb.com/manual/reference/method/db.collection.save/#db.collection.save).
* [db.collection.bulkWrite()](https://docs.mongodb.com/manual/reference/method/db.collection.bulkWrite/#db.collection.bulkWrite)

Retrieve the document to verify:

db.collection.findOne( { \_id: 10 } )

Delete: in 3.4

* [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/#db.collection.deleteMany)
* [db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/#db.collection.deleteOne)

Delete all documents:

db.inventory.deleteOne( { status: "D" } )

> db.inventory.deleteMany({})

Remove all documents from the inventory collection where the status field equals "A":

db.inventory.deleteMany({ status : "A" })

Bulk Write:

The following [bulkWrite()](https://docs.mongodb.com/manual/reference/method/db.collection.bulkWrite/" \l "db.collection.bulkWrite" \o "db.collection.bulkWrite()) performs multiple operations on the collection:

**try** {

db.characters.bulkWrite(

[

{ insertOne :

{

"document" :

{

"\_id" : 4, "char" : "Dithras", "class" : "barbarian", "lvl" : 4

}

}

},

{ insertOne :

{

"document" :

{

"\_id" : 5, "char" : "Taeln", "class" : "fighter", "lvl" : 3

}

}

},

{ updateOne :

{

"filter" : { "char" : "Eldon" },

"update" : { $set : { "status" : "Critical Injury" } }

}

},

{ deleteOne :

{ "filter" : { "char" : "Brisbane"} }

},

{ replaceOne :

{

"filter" : { "char" : "Meldane" },

"replacement" : { "char" : "Tanys", "class" : "oracle", "lvl" : 4 }

}

}

]

);

}

**catch** (e) {

print(e);

}

> db.customers.drop() // drop table

> db.dropDatabase() // drop database

**Queries in MongoDB:**

Insert of Array of Documents:

>use empdb;

>var myemp=[

{ Empid:2,EmpName:"Mallika"},

{ Empid:3,"EmpName":"Kavitha"},

{ "Empid":4,"EmpName":"Lalitha"},

];

>db.employee.insert(myemp)

>db.employee.find().pretty()

Printing in JSON format:

>db.employee.find().forEach(printjson)

Creating our own id:

>db.employee.insert({\_id:10, "EmpName" : "Kiran"})

>db.employee.find().pretty()

>db.employee.find({EmpName : "Kavitha"}).forEach(printjson);

Cursor:

>var myCursor = db.employee.find( { Empid : { $gt:2 }});

while(myCursor.hasNext()){

print(tojson(myCursor.next()));

}

Now write code in script.js and save it then run from shell:

Open any editor and write code: script.js

db=connect('127.0.0.1:27017/myDB');

function insertData(myObject){

db.names.insert(myObject);

}

function getData(){

var cursor=db.names.find();

while(cursor.hasNext()){

printjson(cursor.next())

}

}

>load("script.js")

>insertData({name:"sriram"})

>insertData({name:"raju",”age”:30})

> getData()

>db.names.find().pretty()

{ "\_id" : ObjectId("59e637ca8ea26b5d035efde5"), "name" : "sriram"}

Limit no. of documents:

>db.employee.find().limit(2).forEach(printjson);

Sort

>db.employee.find().sort({Empid:-1}).forEach(printjson)

Aggregation:

>db.employee.count()

Deleting doc:

>db.employee.remove({Empid:2})

Update:

>db.employee.update({"Empid" : 1},

{$set: { "EmpName" : "Srirama murthy"}});

Multi-update:

>db.employee.update

(

{

Empid : 1

},

{

$set :

{

"EmpName" : "Dhulipala Murthy"

"Empid" : 22

}

}

)

Selectors:

>use blogdb

>db.blog.insert(

{

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: ‘murthy',

url: 'http://murthyblog.com',

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

likes: 100

})

//create atleast 4 documents (Records)

>db. blog.find().pretty()

>db.blog.findOne({\_id:123})

>db.blog.find().pretty().sort({user:1})

>db.blog.find({"by":“murthy","title": "MongoDB Overview"}).pretty() // and logic

>db.blog.find({$or:[{"by":“murthy"},{"title": "MongoDB Overview"}]}).pretty() // or logic

>db.blog.find({"likes": {$gt:10}, $or: [{"by": “murthy"},{"title": "MongoDB Overview"}]}).pretty()

Udpate:

>db.blog.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB '}})

>db.blog.find()

**Delete**

> db.blog.remove({title : "My Blog Post"})

Limit records :

> db.blog.find().limit(2)

Skip record:

> db.blog.find().limit(1).skip(1)

Index

> db.blog.ensureIndex({“user":1})

Aggregation

db.blog.aggregate([{$group : {\_id : "$by\_user", likes : {$sum : 1}}}])

Create users collection to apply selectors, criteria,aggragation , modifiers

-------------------------------------------------------------------------------------------------

>use trainingdb

>db.users.insert({

username:’murthy’,

age:50,

registered:true,

email:’murthy@gmail.com’

address:{

street:’Tarnaka’,

city:’Hyderabad’,

State:’AP’,

zip:500013

}

})

 Insert couple of records in users collection:

> db.users.find({"age" : 27})

> db.users.find({"username" : “murthy"})

> db.users.find({"username" : “murthy", "age" : 27})

**Query Criteria:**

"$lt", "$lte", "$gt", and "$gte" are all comparison operators, corresponding to <,

<=, >, and >=, respectively.

> db.users.find({"age" : {"$gte" : 18, "$lte" : 30}}) (and)

> start = **new** Date("01/01/2017")

* db.users.find({"registered" : {"$lt" : start}})

> db.users.find({"user\_id" : {"$in" : [12345, 14323,12432]}) (or query)

> db.users.find({"$or" : [{“username" : “murthy}, {“age" : **27**}]}) ( $or )

> db.users.find({"id\_num" : {"$not" : {"$mod" : [5, 1]}}}) ($not)

>db.users.find({“usernamename" : /murthy/i}) (Regular exp)

Embedded document query:

> db.users.find({"address.zip":500013})

> db.users.find().limit(3)

> db.users.find().skip(3)

> db.users.find().sort({username : 1, age : -1})

Aggregation:

Aggregations operations process data records and return computed results.

Aggregation operations group values from multiple documents and return single result. (like count(\*) in sql)

Operators: $sum, $limit,$sort,$group,$avg,$max,$min,$push,$first,$last

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

Add some more posts to test aggregate

{

\_id: ObjectId(7df78ad8902c)

title: 'Dotnet Overview',

description: 'Dotnet is MS technology',

by\_user: 'john',

email: 'john@hotmail.com',

tags: ['Dotnet', 'Web API', 'MVC'],

likes: 100

},

{

\_id: ObjectId(7df78ad8902d)

title: 'Java ',

description: 'Java is great Technology',

by\_user: 'murthy',

email: 'murthy@yahoo.com',

tags: ['java', 'spring’],

likes: 10

},

{

\_id: ObjectId(7df78ad8902e)

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Neo4j',

email: 'new4J@gmail.com',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

},

Try Aggregates:

Sql : **select by\_user, count(\*) from post group by by\_user**.

> db.post.aggregate([{$group : {\_id : "$by\_user", result : {$sum : 1}}}])

> db.post.aggregate([{$group : {\_id : "$by\_user", result : {$sum : "$likes"}}}])

> db.post.aggregate([{$group : {\_id : "$by\_user", result : {$avg : "$likes"}}}])

> db.post.aggregate([{$group : {\_id : "$by\_user", result: {$min : "$likes"}}}])

Field level aggregation

>db.post.aggregate([{$group : {\_id : "$by\_user", result : {$max : "$likes"}}}])

Pipeline Concept in Aggregation

Following are the possible stages in aggregation framework −

* **$project** − Used to select some specific fields from a collection.
* **$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 positions.

Modifiers in Mongo : $set,$inc, $decr,$push,$limit,$sort

use myDB;

var obj={

"name":"murthy",

"job":"engineer",

"salary":5000

};

db.people.insert(obj);

CTRL+B

> use myDB

> db.people.find().pretty()

> db.people.find({},{name:1}) // display only name field

>db.people.update({"name":"murthy"},{"$set":{"country":"IN"}})

> db.people.find().pretty()

//note :db.table.find({name:’murthy’},{country:1,\_id:0}).sort({age:1}).limit(10)

// in sql is :

//select country from table where name:’murthy’ orderby age desc limit 10

use myDB;

var obj={

"name":"Rama",

"job":"manager",

"salary":5000,

"languages":{

"web":["html","java","C++"],

"mobile":["ionic","android"]

}

};

db.people.insert(obj);

>db.people.find().pretty()

// update language (document in document update – Nested Document)

db.people.update({"languages.web":["html","java","C++"]}, {

"$set":{"languages.web":["I know nothing"]}

})

**$inc modifier:**

use myDB;

var obj={

"website":"http;\\amazon.com",

"hits":0

}

db.googleAnalytics.insert(obj);

db.googleAnalytics.find().pretty();

Ctrl+b

**Now update hit by incrementing with $inc modifier:**

use myDB;

db.googleAnalytics.update({

"website":"http://amazon.com"},

{"$inc":{

"hits":1

}

})

db.googleAnalytics.find().pretty();

ctrl+b

ctrl+b

ctrl+b

observe everytime $inc increments the hit

db.googleAnalytics.find().pretty();

Create a CSV file called data.csv and put the following data in it

Employeeid,EmployeeName

101,Murthy

202,Sriram

303,Rama

…bin/>mongoimport –db testdb --type csv --headerline –file data.csv

Exporting

…/bin> mongoexport –db testdb --collection data --type csv --fields empid,empname --out data.csv

Start the Mongodb server:

C\>mongod.exe --dbpath C:\data\db

Creating Database (if not there it will create new or use existing one)

>use Empdb

Creating collection :

>db.employee.insert({ “Empid”:1, “EmpName”:”Murthy”})

Mongo Security, Backup and Restore:

Create user and assign role:

>db.createUser ({user: “murthy”,pwd:”welcome”,

roles:[{role:”userAdminAnyDatabase”, db:”admin”} ]

})

To create a user who will manage single database , use useAdmin option:

>db.createUser({ user:”kavitha”,pwd:”welcome”,

roles:[ {role:”userAdmin” , db:”empdb”}]}

Manage Uses with roles:

Readrole : only users can read

Readwrite: read and write (CRUD is allowed)

>db.createUser({

user:”mallika”, password:”welcome”,

roles:[ {

role:”read”,db:”Marketing”,

role:”readWrite”,db:”Sales”

}]}

**Mongodb authentication with Kerberos**

**Instead of using username/password use certificates from Kerberos**

**Step 1)** Use x.509 Certificates to authenticate the client – A certificate is basically a trusted signature between the client and the MongoDB Server.

* 1. A valid certificate must be bought from a valid third party authority and install it on the MongoDB Server.
  2. The Client certificate must have the following properties (A single Certificate Authority (CA) must issue the certificates for both the client and the server . The Client certificates must contain the following fields – keyUsage and extendedKeyUsage.
  3. Each user who connects to the MongDB Server needs to have a separate certificate.

**Step 2)** Configure MongoDB with Kerberos Authentication on windows – Kerberos is an authentication mechanism used in large client-server environments.

It is a very secure mechanism wherein the password is only allowed if it is encrypted.

**Step 3)** Start the mongod.exe server process.

**Step 4)** Start the mongo.exe client process and connect to the MongoDB server.

**Step 5)** Add a user in MongoDB, which is basically a Kerberos principal name to the $external database.

The $external database is a special database which tells MongoDB to authenticate this user against a Kerberos system instead of its own internal system.

>use $external

>db.createUser({

user: "user1@example.NET",

roles:[ {role: "read" , db:"Marketing"} ]

}

**Step 6)** Start mongod.exe with Kerberos support by using the following command

mongod.exe –auth –setParameter authenticationMechanisms=GSSAPI

And then you can now connect with the Kerberos user and Kerberos authentication to the database.

Replica Set:

**Step 1)** Ensure that all mongod.exe instances which will be added to the replica set are installed on different servers. This is to ensure that even if one server goes down, the others will be available

**Step 2)** Ensure that all mongo.exe instances can connect to each other. From ServerA, issue the below 2 commands

mongo –host ServerB –port 27017

mongo –host ServerC –port 27017

**Step 3)** Start the first mongod.exe instance with the replSet option. This option provides a grouping for all servers which will be part of this replica set.

mongo –replSet "Replica1"

Where "Replica1" is the name of replica set.

**Step 4)** Now that the first server is added to the replica set, the next step is to initiate the replica set by issuing rs.initiate ()

**Step 5)** Verify the replica set by issuing the command rs.conf() to ensure the replica set up properly

Now add secondary using rs.add()

The secondary servers can be added to the replica set with rs.add command. This command takes in the name of the secondary servers and adds the servers to the replication set.

**Step 6)** Suppose if you have ServerA, ServerB, and ServerC, which are required to be part of your replica set and ServerA, is defined as the primary server in the replica set.

To add ServerB and ServerC to the replica set issue the commands

rs.add("ServerB")

rs.add("ServerC")

## Reconfiguring or Removing using rs.remove()

To remove a server from the configuration set, use the "rs.remove"

**Step 1)** First perform a shutdown of the instance which want to remove.

**Use**  db.shutdownserver command from the mongo shell.

**Step 2)** Connect to the primary server

**Step 3)** Use the rs.remove command to remove the required server from the replica set. So suppose if you have a replica set with ServerA, ServerB, and ServerC, and you want to remove ServerC from the replica set, issue the command

rs.remove("ServerC")

rs.status() command gives status of servers

Sharding:

Sharding is a concept in MongoDB, which splits large data sets into small data sets across multiple MongoDB instances.

Logically all the shards work as one collection.

The components of a Shard include

1. **A Shard** – MongoDB instance which holds the subset of the data. In production environments, all shards need to be part of replica sets.
2. **Config server** – This is a mongodb instance which holds metadata about the cluster, basically information about the various mongodb instances which will hold the shard data.
3. **A Router** – This is a mongodb instance which basically is responsible to re-directing the commands send by the client to the right servers.

**Step 1)** Create a separate database for the config server.

mkdir /data/configdb

**Step 2)** Start the mongodb instance in configuration mode. Suppose if we have a server named ServerD which would be our configuration server, we would need to run the below command to configure the server as a configuration server.

mongod –configdb ServerD: 27019

**Step 3)** Start the mongos instance by specifying the configuration server

mongos –configdb ServerD: 27019

**Step 4)** From the mongo shell connect to the mongo's instance

mongo –host ServerD –port 27017

**Step 5)** If Server A and Server B which needs to be added to the cluster

>sh.addShard("ServerA:27017")

>sh.addShard("ServerB:27017")

**Step 6)** Enable sharding for the database. So if we need to shard the Empdb database,

>sh.enableSharding(Empdb)

**Step 7)** Enable sharding for the collection. So if we need to shard the employee collection, issue the below command

sh.sharedCollection("db.employee" , { "Empid" : 1 , "EmpName" : 1})