# MongoDB Crash Course

<https://youtu.be/ofme2o29ngU>

(I referred this link, ARC)

<https://www.youtube.com/watch?v=Ya0H-7A5cE4&list=PLp50dWW_m40UWFSV6PTgYzciZJIxgHy7Q>

<https://youtu.be/ExcRbA7fy_A>

Installing MongoDB:  
<https://youtu.be/59Yhxwdgi7I>

**22/11/22 MongoDB VisualStudio Commands**

**Create database**

test> use suresh

switched to db suresh

suresh> db

Suresh

**Drop database**

suresh> db

Suresh

Suresh>db.dropDatabase()

**Create Collections**

suresh> db.createCollection("Training")

{ ok: 1 }

suresh> db.createCollection("Blog")

{ ok: 1 }

suresh> db.createCollection("Learning")

{ ok: 1 }

suresh> show dbs

admin 40.00 KiB

config 108.00 KiB

local 184.00 KiB

suresh 24.00 KiB

suresh> show collections

Blog

Learning

Training

**Drop Collection**

suresh> db

Suresh

Suresh>db.Blog.drop()

suresh> show collections

Learning

Training

**Inserting Document**

suresh> db.Learning.insert({"name":"suresh","role":"Trainer"})

DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.

{

acknowledged: true,

insertedIds: { '0': ObjectId("637c89eadfd3ba418e8d41c0") }

}

**Inserting one Document**

suresh> db.Learning.insertOne({"name":"vanitha","role":"UI/UXTrainer"})

{

acknowledged: true,

insertedId: ObjectId("637c8a40dfd3ba418e8d41c1")

}

**Inserting Many Documents**

suresh> db.Learning.insertMany([{"name":"vasanth","role":"Back-End Trainer"},{"name":"Gawtham","role":"ReactNative Trainer"}])

{

acknowledged: true,

insertedIds: {

'0': ObjectId("637c8bb4dfd3ba418e8d41c2"),

'1': ObjectId("637c8bb4dfd3ba418e8d41c3")

}

}

suresh> db.Training.insertMany([{"id":"1","sub":"Html/css","isActive":"true"},

... {"id":"2","sub":"JS","isActive":"false"},

... {"id":"3","sub":"MongoDB","isActive":"true"}])

{

acknowledged: true,

insertedIds: {

'0': ObjectId("637c931adfd3ba418e8d41c4"),

'1': ObjectId("637c931adfd3ba418e8d41c5"),

'2': ObjectId("637c931adfd3ba418e8d41c6")

}

}

**Updating one Document**

suresh> db.Training.updateOne(

... {"id":"1"},

... {

... $set: {"sub":"HTML/CSS"}

... }

... )

{

acknowledged: true,

insertedId: null,

matchedCount: 1,

modifiedCount: 1,

upsertedCount: 0

}

**Updating Many Document**

suresh> db.Training.updateMany(

... {"isActive":"true"},

... {

... $set: {"timings":"2.30 to 3.30"}

... }

... )

{

acknowledged: true,

insertedId: null,

matchedCount: 2,

modifiedCount: 2,

upsertedCount: 0

}

**Read data from Collections using find methods:**

**Find()**

suresh> db.Learning.find()

[

{

\_id: ObjectId("637c89eadfd3ba418e8d41c0"),

name: 'suresh',

role: 'Trainer'

},

{

\_id: ObjectId("637c8a40dfd3ba418e8d41c1"),

name: 'vanitha',

role: 'UI/UXTrainer'

},

{

\_id: ObjectId("637c8bb4dfd3ba418e8d41c2"),

name: 'vasanth',

role: 'Back-End Trainer'

},

{

\_id: ObjectId("637c8bb4dfd3ba418e8d41c3"),

name: 'Gawtham',

role: 'ReactNative Trainer'

}

]

**findOne()**

suresh> db.Learning.findOne()

{

\_id: ObjectId("637c89eadfd3ba418e8d41c0"),

name: 'suresh',

role: 'Trainer'

}

**find({queries}) with queries/condition**

suresh> db.Learning.find({"name":"vasanth"})

[

{

\_id: ObjectId("637c8bb4dfd3ba418e8d41c2"),

name: 'vasanth',

role: 'Back-End Trainer'

}

]

**find({queries}) with multiple queries/condition**

suresh> db.Learning.find({"name":"suresh","role":"Trainer"})

[

{

\_id: ObjectId("637c89eadfd3ba418e8d41c0"),

name: 'suresh',

role: 'Trainer'

}

]

**findOneAndReplace({query},{replace})**

suresh> db.Learning.findOneAndReplace({"role":"ReactNative Trainer"},{"role":"Frontend Developer","Exp":"2years"})

{

\_id: ObjectId("637c8bb4dfd3ba418e8d41c3"),

name: 'Gawtham',

role: 'ReactNative Trainer'

}

**Document update like this:**

{

  "\_id": {

    "$oid": "637c8bb4dfd3ba418e8d41c3"

  },

  "role": "Frontend Developer",

  "Exp": "2years"

}

**findOneAndDelete({query})**

suresh> db.Learning.findOneAndDelete({"name":"gawtham"})

{

\_id: ObjectId("637cfd38dfd3ba418e8d41c7"),

name: 'gawtham',

role: 'React Native Developer'

}

**Document update like this:**

[

  {

    "\_id": "637c89eadfd3ba418e8d41c0",

    "name": "suresh",

    "role": "Trainer"

  },

  {

    "\_id": "637c8a40dfd3ba418e8d41c1",

    "name": "vanitha",

    "role": "UI/UXTrainer"

  },

  {

    "\_id": "637c8bb4dfd3ba418e8d41c2",

    "name": "vasanth",

    "role": "Back-End Trainer"

  }

]

**deleteOne({query})**

suresh> db.Training.deleteOne({"id":"1"})

{ acknowledged: true, deletedCount: 1 }

**Document update like this:**

[

  {

    "\_id": "637c931adfd3ba418e8d41c4",

    "id": "1",

    "sub": "HTML/CSS",

    "isActive": "true",

    "timings": "2.30 to 3.30"

  },

  {

    "\_id": "637c931adfd3ba418e8d41c5",

    "id": "2",

    "sub": "JS",

    "isActive": "false"

  },

  {

    "\_id": "637c931adfd3ba418e8d41c6",

    "id": "3",

    "sub": "MongoDB",

    "isActive": "true",

    "timings": "2.30 to 3.30"

  }

]

Output:

[

  {

    "\_id": "637c931adfd3ba418e8d41c5",

    "id": "2",

    "sub": "JS",

    "isActive": "false"

  },

  {

    "\_id": "637c931adfd3ba418e8d41c6",

    "id": "3",

    "sub": "MongoDB",

    "isActive": "true",

    "timings": "2.30 to 3.30"

  }

]

# 23/11/22 commands workedout:

**Find() with greaterthan and equalto , greaterthan , lessthan , lessthan equalto**

db.Training.find({"rating":{$gte:"3"}})

db.Training.find({"rating":{$gt:"3"}})

suresh> db.Training.find({"rating":{$lt:"4"}})

[

{

\_id: ObjectId("637c931adfd3ba418e8d41c6"),

id: '3',

sub: 'MongoDB',

isActive: 'true',

timings: '2.30 to 3.30',

rating: '3'

},

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3'

}

]

**Multiple Queries:**

suresh> db.Training.find({"rating":{$lt:"4"},"timings":"5.30 to 6.30"})

[

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3'

}

]

**$and operation**

suresh> db.Training.find({$and:[{"isActive":"true"},{"rating":{$gte:"4"}}]})

[

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4'

}

]

**$Or operation**

suresh> db.Training.find({$or:[{"isActive":"true"},{"rating":{$gte:"4"}}]})

[

{

\_id: ObjectId("637c931adfd3ba418e8d41c5"),

id: '2',

sub: 'JS',

isActive: 'false',

rating: '5'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c6"),

id: '3',

sub: 'MongoDB',

isActive: 'true',

timings: '2.30 to 3.30',

rating: '3'

},

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4'

},

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3'

}

]

**Projection Concepts**

**Particular Field display - but it displays with id by default**

db.Training.find({},{"rating":1})  **( 1->means display rating)**

[

{ \_id: ObjectId("637c931adfd3ba418e8d41c5"), rating: '5' },

{ \_id: ObjectId("637c931adfd3ba418e8d41c6"), rating: '3' },

{ \_id: ObjectId("637db6be28807e6520c43838"), rating: '4' },

{ \_id: ObjectId("637db82c28807e6520c4383a"), rating: '3' }

]

**Particular Field only display without id:**

suresh> db.Training.find({},{"rating":1,"\_id":0})

[ { rating: '5' }, { rating: '3' }, { rating: '4' }, { rating: '3' } ]

**Particular Field only display with Condition:**

suresh> db.Training.find({"rating":{$gt:"3"}},{"rating":1,"\_id":0,"sub":1})

**( 1->means display rating, 0->means don’t display \_id field)**

[ { sub: 'JS', rating: '5' }, { sub: 'SQL', rating: '4' } ]

**( This above three concepts are nothing but PROJECTION)**

**Aggregation(pipeline, option)**

**Use $sort**

suresh> var pipeline=[

... {$sort:{"name":-1}} **(-1 means desending order, 1- means ascending order)**

... ]

suresh> db.Training.aggregate(pipeline)

[

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4',

name: 'Vasanth'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c5"),

id: '2',

sub: 'JS',

isActive: 'false',

rating: '5',

name: 'Vanitha'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c6"),

id: '3',

sub: 'MongoDB',

isActive: 'true',

timings: '2.30 to 3.30',

rating: '3',

name: 'Suresh'

},

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3',

name: 'Gawtham'

}

]

**Use $sort and $limit**

suresh> var pipeline=[

... {$sort:{"name":-1}},

... {$limit:2} **(it displays only 2 documents out of 4)**

... ]

suresh> db.Training.aggregate(pipeline)

[

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4',

name: 'Vasanth'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c5"),

id: '2',

sub: 'JS',

isActive: 'false',

rating: '5',

name: 'Vanitha'

}

]

# 24/11/22 commands workedout:

**Limit()-> only first two documents**

suresh> db.Training.find().limit(2)

[

{

\_id: ObjectId("637c931adfd3ba418e8d41c5"),

id: '2',

sub: 'JS',

isActive: 'false',

rating: '5',

name: 'Vanitha'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c6"),

id: '3',

sub: 'MongoDB',

isActive: 'true',

timings: '2.30 to 3.30',

rating: '3',

name: 'Suresh'

}

]

**Skip()-> skip first two documents**

suresh> db.Training.find().skip(2)

[

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4',

name: 'Vasanth'

},

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3',

name: 'Gawtham'

}

]

**Sort() 1-> means ascending order, -1 desending order**

suresh> db.Training.find().sort({name:1})

[

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3',

name: 'Gawtham'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c6"),

id: '3',

sub: 'MongoDB',

isActive: 'true',

timings: '2.30 to 3.30',

rating: '3',

name: 'Suresh'

},

{

\_id: ObjectId("637c931adfd3ba418e8d41c5"),

id: '2',

sub: 'JS',

isActive: 'false',

rating: '5',

name: 'Vanitha'

},

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4',

name: 'Vasanth'

}

]

**Backup Mongo Database tool( Mongodump)**

**Index:**

db.Training.find({name:"Suresh"}).explain('executionStats')- Total document Examined-4

db.Training.createIndex({name:1})

db.Training.getIndex()

db.Training.find({name:"Suresh"}).explain('executionStats')Total document Examined-1

db.Training.dropIndex({name:1})

db.Training.find({name:"Suresh"}).explain('executionStats')

 db.Training.createIndex({name:1})

 db.Training.getIndex()

db.Training.find({name:"Suresh"}).explain('executionStats')

db.Training.dropIndex({name:1})

**(so timing is so quick compare than normal finding data)**

**Nested Documents**

db.Learning.insertOne({ name: "vasanth", role: "trainer", lang: ["angular", "vue"], trainee: [{ name: "suresh", role: "fullstack" }, { name: "manoj", role: "react" }, { name: "gowtham", role: "native" }] })

 "\_id": "637f6252887f850e5705abc0",

    "name": "vasanth",

    "role": "trainer",

    "lang": [

      "angular",

      "vue"

    ],

    "trainee": [

      {

        "name": "suresh",

        "role": "fullstack"

      },

      {

        "name": "manoj",

        "role": "react"

      },

      {

        "name": "gowtham",

        "role": "native"

      }

    ]

  }

]

**Find data in nested documents (array of object data)**

suresh> **db.Learning.find({trainee:{name:"manoj",role:"react"}})**

[

{

\_id: ObjectId("637f6252887f850e5705abc0"),

name: 'vasanth',

role: 'trainer',

lang: [ 'angular', 'vue' ],

trainee: [

{ name: 'suresh', role: 'fullstack' },

{ name: 'manoj', role: 'react' },

{ name: 'gowtham', role: 'native' }

]

}

]

**Find data in nested documents (array )**

suresh> db.Learning.find({lang:["angular","vue"]})

[

{

\_id: ObjectId("637f6252887f850e5705abc0"),

name: 'vasanth',

role: 'trainer',

lang: [ 'angular', 'vue' ],

trainee: [

{ name: 'suresh', role: 'fullstack' },

{ name: 'manoj', role: 'react' },

{ name: 'gowtham', role: 'native' }

]

}

]

# 25/11/22 commands workedout:

# $in, $nin: with Array

suresh> db.Training.find({rating:{$in:["4","5"]}})

[

{

\_id: ObjectId("637c931adfd3ba418e8d41c5"),

id: '2',

sub: 'JS',

isActive: 'false',

rating: '5',

name: 'Vanitha'

},

{

\_id: ObjectId("637db6be28807e6520c43838"),

id: '4',

sub: 'SQL',

isActive: 'true',

timings: '3.30 to 4.30',

rating: '4',

name: 'Vasanth'

}

]

**$nin:**

suresh> db.Training.find({rating:{$nin:["4","5"]}})

[

{

\_id: ObjectId("637c931adfd3ba418e8d41c6"),

id: '3',

sub: 'MongoDB',

isActive: 'true',

timings: '2.30 to 3.30',

rating: '3',

name: 'Suresh'

},

{

\_id: ObjectId("637db82c28807e6520c4383a"),

id: '5',

sub: 'React',

isActive: 'true',

timings: '5.30 to 6.30',

rating: '3',

name: 'Gawtham'

}

]

**02/12/22 MongoDB VisualStudio Commands (aggregation concepts)**

db.purchase\_ordersdetails.insertMany(

     [

          {product: "toothbrush", total: 30, customer: "Suresh"},

          {product: "guitar", total: 250, customer: "Manoj"},

          {product: "milk", total: 25, customer: "Suresh"},

          {product: "pizza", total: 50, customer: "Vasanth"},

          {product: "toothbrush", total: 30, customer: "Vasanth"},

          {product: "pizza", total: 50, customer: "Gowtham"},

          {product: "toothbrush", total: 30, customer: "Suresh"},

     ]

)

//how many toothbrush were sold

db.purchase\_ordersdetails.count({product:"toothbrush"})

db.purchase\_orders.countDocuments({product:"toothbrush"}) //both are same return value

//output:

3

//how many list of products sold

db.purchase\_ordersdetails.distinct('product')  //using distinct, we can avoid duplication too

//output:

[ 'guitar', 'milk', 'pizza', 'toothbrush' ]

//how much of moneny spent by each customer  using:group concept

db.purchase\_ordersdetails.aggregate(

     [

          {$group:{\_id:'$customer',total:{$sum:'$total'}} }

     ]

)

//output:

[

     { \_id: 'Vasanth', total: 80 },

     { \_id: 'Gowtham', total: 50 },

     { \_id: 'Suresh', total: 85 },

     { \_id: 'Manoj', total: 250 }

   ]

//how much of moneny spent for each product

db.purchase\_ordersdetails.aggregate(

     [

          {$group:{\_id:'$product',total:{$sum:'$total'}} }

     ]

)

//output:

[

     { \_id: 'milk', total: 25 },

     { \_id: 'pizza', total: 100 },

     { \_id: 'toothbrush', total: 90 },

     { \_id: 'guitar', total: 250 }

   ]

//above same concept with sort for product

   db.purchase\_ordersdetails.aggregate(

     [

          {$group:{\_id:'$product',total:{$sum:'$total'}} },

          {$sort:{total:-1}}

     ]

)

//output:

[

     { \_id: 'guitar', total: 250 },

     { \_id: 'pizza', total: 100 },

     { \_id: 'toothbrush', total: 90 },

     { \_id: 'milk', total: 25 }

   ]

//above same concept with sort for customer

db.purchase\_ordersdetails.aggregate(

     [

          {$group:{\_id:'$customer',total:{$sum:'$total'}} },

          {$sort:{total:-1}}

     ]

)

//output:

[

     { \_id: 'Manoj', total: 250 },

     { \_id: 'Suresh', total: 85 },

     { \_id: 'Vasanth', total: 80 },

     { \_id: 'Gowtham', total: 50 }

   ]

//match concept (nothing but filter) only suresh and vasanth customers detail

db.purchase\_ordersdetails.aggregate([

     {$match:{customer:{$in:['Suresh' , 'Vasanth']}}},

     {$group:{\_id:'$customer',total:{$sum:'$total'}}},

     {$sort:{total:-1}}

])

//output:

[ { \_id: 'Suresh', total: 85 }, { \_id: 'Vasanth', total: 80 } ]

//eg:2

db.purchase\_ordersdetails.aggregate([

     {$match:{total:{$lte:30}}},

     {$group:{\_id:'$customer',total:{$sum:'$total'}}},

     {$sort:{total:1}}

])

//output:

[ { \_id: 'Vasanth', total: 30 }, { \_id: 'Suresh', total: 85 } ]

//eg:3

db.purchase\_ordersdetails.aggregate([

     {$match:{customer:'Suresh'}},

     {$group:{\_id:'$product',total:{$sum:'$total'}}},

     {$sort:{total:1}}

])

//output:

[ { \_id: 'milk', total: 25 }, { \_id: 'toothbrush', total: 60 } ]

**Aggregation:**

(I referred this link too(Mike Dane))

<https://www.mikedane.com/databases/mongodb/aggregation/>

<https://www.youtube.com/watch?v=Kk6Er0c7srU> (aggregation)

**Mongoose:**

<https://www.youtube.com/watch?v=eYVGoXPq2RA&t=315s> (mongoose)

**I reffered these below videos for Mongoose Transaction**

<https://www.youtube.com/watch?v=spdyW-ANgJw&t=328s> (Transaction)

<https://www.youtube.com/watch?v=fQPB5TPqcGM&t=144s> (Transaction)

**Replicaset commands (Docker):**

**Docker install and version check:**

docker -v

docker images

docker network create mongoNet

**Create objects for 27017:**

docker run -d -p 27020:27017 --net mongoNet --name r0 mongo:latest --replSet suresh

docker run -d -p 27021:27017 --net mongoNet --name r1 mongo:latest --replSet suresh

docker run -d -p 27022:27017 --net mongoNet --name r2 mongo:latest --replSet suresh

docker run -d -p 27023:27017 --net mongoNet --name r3 mongo:latest --replSet suresh

docker ps

docker exec -it r0 bash

Mongosh

**Open next terminal:**

Ipconfig(IPv4 Address. . . . . . . . . . . : 192.168.29.56) get this IP address

**Come back to mongosh terminal and type:**

**Config statement:**

config={"\_id":"suresh", "members":[{\_id:0,host:"192.168.29.56:27020"},{\_id:1,host:"192.168.29.56:27021"},{\_id:2,host:"192.168.29.56:27022"},{\_id:3,host:"192.168.29.56:27023"}]}

rs.initiate(config)

Get result

{ok:1}

Finally we need to change .env file

MONGODB\_LOCAL\_URI=mongodb://localhost:27026/mongoose-DB

**Multiple collection Join with complex aggregation:**

**<https://www.youtube.com/watch?v=6Machxvfo0U>**

**User sign-up & login:**

**<https://www.youtube.com/watch?v=HGgyd1bYWsE>**

**Replicaset commands:**

**video ref:**

<https://www.youtube.com/watch?v=bJo7nr9xdrQ>

**Steps:**

**standalone mongodb server off in services .**

C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\Program Files\MongoDB\Server\6.0\data" --logpath "C:\Program Files\MongoDB\Server\6.0\log\mongod.log" --port 27017 --storageEngine=wiredTiger --journal --replSet suresh

C:\WINDOWS\system32>mongosh --port 27017

>rs.initiate()

Secondary>**click enter (automatically convert into primary)**

Primary>

primary>rs.status()

Primary>show dbs;

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

**Reconfig concept in replicaset:**

rsconf={\_id:"suresh",members:[{\_id:0,host:"localhost:27017"}]}

**rs.reconfig(config, { force: true } )**

rs.initiate(rsconf)

rs.status()

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

**If you want secondary nodes, you can follow next steps:**

**Third step:** create data1 and data2 for configure Secondary servers (check in video link)

**In Administrator command prompt**

C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data3\db" --logpath "C:\data3\log\mongod.log" --port 27041 --storageEngine=wiredTiger --journal --replSet suresh

C:\Program Files\MongoDB\Server\6.0\bin>mongod --dbpath "C:\data4\db" --logpath "C:\data4\log\mongod.log" --port 27042 --storageEngine=wiredTiger --journal --replSet suresh

**Fourth step:**

C:\WINDOWS\system32>mongosh --port 27017

test> show dbs;

rsconf={\_id:"suresh",members:[{\_id:0,host:"localhost:27017"}]}

Or **rsconf**={\_id:"suresh",members:[{\_id:0,host:"localhost:27017"},{\_id:1,host:”localhost:27041”},{\_id:2,host:”localhost:27042”}]}

**rs.initiate(rsconf)**

**rs.status()**

**Open new terminal in administrator**: :mongosh --port 27041

rs.SecondaryOk() **(instead of slaveOk())**

**Open new terminal in administrator**: :mongosh --port 27042

rs.SecondaryOk() **(instead of slaveOk())**