MongoDB

Learning MongoDB

PRAVEEN NAIR

Introduction to MongoDB

MongoDB is a document database.

MongoDB is a non-relational, non-tabular database.

Relational data is stored differently.

Instead of having multiple tables all the related data are stored together.

In MongoDB, tables are called collections.

MongoDB can be installed locally or in cloud called MongoDB Atlas

Mongosh or Compass can be used to query MongoDB

Advantages of MongoDB

Flexibility: MongoDB is schema-less, meaning you don't need to design a schema for the database.

Scalability: MongoDB can be horizontally scaled by distributing data across multiple servers, a process called sharding.

Performance: MongoDB is fast at inserting or updating large numbers of records. It also supports geospatial efficiently.

MongoDB Installation

https://www.mongodb.com/try/download/community

Choose MSI

Connect to local mongodb

Install Mongosh (https://www.mongodb.com/try/download/shell)

Type mongosh –version

Type mongosh to get prompt

show dbs

use myproj to create or access new db

db.dropDatabase(“dbname”) to delete database (or db.dropDatabase())

show collections

db.createCollection(“employees")

db.createCollection(“employees“,{capped:true,size:100,max:100}) //max 100 employees, size max 100 bytes. Delets oldest document

db.emploees.drop() to delete collection

db.restaurant.renameCollection('restaurants’) //rename collection

Case sensitive

Inserting Data

```
db.employees.insertOne({
  name: "John Smith",
  email: "john@gmail.com",
  department: "IT",
  salary: 1456,
  location: ["FL", "OH"],
  date: Date()
})
```

```
db.employees.find()
```

Inserting Multiple Data

```
db.employees.insertMany([{
  name: "Mike Joseph",
  email: "mike@gmail.com",
  department: "IT",
  salary: 2456,
  location: ["FL", "TX"],
  date: Date()
},
{ name: "Cathy G",
  email: "cathy@gmail.com",
  department: "IT",
  salary: 3456,
  location: ["AZ", "TX"],
  date: Date()
}])
```

Data type

String

Integers

Double (decimal)

Boolean

Date() (new Date())

Null

Arrays []

Nested documents {}

Find Data

returns first 20, then type it for more documents

```
db.employees.find()  //returns first 20, then type it for more documents
db.employees.find().skip(2)
db.employees.findOne()
db.users.find().sort({name:1})  //sorting -1 for reverse
 db.users.find().limit(1)  //returns 1 document sort by object id
db.users.find().sort({name:1}).limit(3)
db.employees.find( {department: "IT"} )
db.users.find({name:"Cathy",pass:"1234"})  //two condition
db.employees.find({}, {_id: 0, salary: 1, date: 1})  //cannot give 0
db.users.find({},{_id:false,name:true}) //cannot give false
db.employees.find({}, {_id: 0, salary: 0, date: 1})//either use 0 or 1, can’t use both
 db.users.find({'address.city':'Gwenborough’})  //query nested documents
db.users.find({address.geo.lat:'-37.3159’})
db.employees.find({'location':'TX’})  //where location : [‘FL’,’TX’]
db.users.find().count()
db.employees.find({},{"dept":"$department",email:1,salary:1})  //dept is alias
```

Update Document

```
db.users.find({'address.city':'Gwenborough’})  //query nested documents
db.users.find({address.geo.lat:'-37.3159’})
db.employees.find({'location':'TX’})  //where location : [‘FL’,’TX’]
```

Query Operators

```
db.employees.find({department:{$eq:'HR'}})

db.users.find({email:{$ne:'cathy@gmail.com'}})

db.employees.find({salary:{$gt:3000}})

db.employees.find({salary:{$gte:3000}})

db.employees.find({salary:{$gte:3000,$lt:5000}})

db.employees.find({salary:{$gt:1000},department:{$eq:'HR'}})

db.employees.find({salary:{$gt:2000},department:{$in:['HR','IT']}})

db.employees.find({salary:{$gt:2000},department:{$nin:['HR','IT']}})

db.employees.find({$or:[{salary:{$gt:2000}},{department:{$eq:'HR'}}]})

db.employees.find({$and:[{salary:{$gt:2000}},{department:{$eq:'HR'}}]})

db.employees.find({$nor:[{salary:{$gt:2000}},{department:{$eq:'HR'}}]})    //like and but both should be false

db.employees.find({department:{$not:{$eq:'HR'}}})

db.users.find({email1:{$exists:false}})
```

Update Document

```
db.employees.updateOne({email:'cathy@gmail.com'},{$set:{department:'HR'}})

db.employees.updateOne(
  { email: "ria@gmail.com" },
  {
    $set:
      {
        name: "Ria K",
        email: "ria@gmail.com",
        department: "HR",
        salary: 5000,
        location: ["FL", "LA"],
        date: Date()
      }
  },
  { upsert: true }
)

db.employees.updateMany({}, { $set: { date: Date() } })
```

Delete Document

```
db.employees.deleteOne({email:'ria@gmail.com'})
db.employees.deleteMany({email:'ria@gmail.com'})
```

Query Operators - 2

```
db.employees.find(
   {department:{$in:["HR","Admin"]}}
)


db.employees.find(
   {department:{$nin:["HR","Admin"]}}
)
```

Update Operators(fields)

```
db.employees.updateOne({email:'cathy@gmail.com'},{$set:{email:'cathy@hotmail.com'}})

db.employees.updateMany({},{$set:{points:0}})    -- new field

db.employees.updateMany({},{$inc:{points:70}})

db.employees.updateMany({},{$rename:{points:'score'}})

db.employees.updateMany({},{$unset:{score:""}})  //deletes the field
```

Summary - CRUD

```
db.users.find({filter},{projection})
db.users.insertOne({document})
db.users.insertMany([{document},{document}]
db.users.deleteMany({filter})
db.users.updateMany({filter},{$set:{flag:false}})
db.users.updateMany({filter},{$unset:{flag:""}})
db.users.updateMany({filter},{$inc:{score:20}})  //increment by 20
db.users.updateMany({filter},{$rename:{flag:"indicator"}})
db.users.find({$and:[{},{}]})
```

Update Operators (arrays)

db.employees.updateOne({email:'cathy@hotmail.com'},{$addToSet:{location:'FL’}}) //duplicates won’t be added, use push instead

db.employees.updateOne({email:'cathy@hotmail.com'},{$pop:{location:1}}) –try -1

db.employees.updateMany({email:'cathy@hotmail.com'},{$pull:{points:{$gt:1}}})

db.employees.updateMany({email:'cathy@hotmail.com'},{$push:{points:5}})

Indexes (improves search but slows insert, update)

```
db.users.find({email:'cathy@gmail.com'}).explain("executionStats")
 totalDocsExamined: 13,

db.users.createIndex({email:1})   //ascending
totalDocsExamined: 3,

db.users.getIndexes()

db.users.createIndex({'email':1},{unique:true})

 db.users.dropIndex("email_1")
```

Misc – skip and limit

```
db.employees.find().skip(2)
db.employees.find().skip(2).limit(1)
```

Used for pagination

Aggregation pipeline

```
db.employees.aggregate([
  {pipeline1 or stage 1 },
  {pipeline2 or stage 2},
])
```

Aggregation - $match

```
db.employees.aggregate([
  {
    $match: {}   //stage 1
  },
  {
    $group: { _id: "$department", total: { $sum: "$salary" } }  //stage 2
  },
{
    $sort: { “department": -1 }
  },
])
```

Aggregation - $match

```
db.employees.aggregate([
 {
   $match: { salary: { $gt: 1000 } }   //state 1
 },
 {
   $group: { _id: "$department", total: { $sum: "$salary" } }  //stage 2
 }
])
```

Aggregation - $group

An aggregation pipeline return results for groups of documents. For example, return the total, average, maximum, and minimum values.

```
db.employees.aggregate([
  {
    $group: {
      _id: "$department",
      Total: { $sum: "$salary" },
      Hightest: { $max: "$salary" },
      Lowest: { $min: "$salary" },
      Average: { $avg: "$salary" },
    },
  },
]);
```

Aggregation - $limit

```
db.employees.aggregate([
  { $group: { _id: "$department", Total: { $sum: "$salary" } } },
  { $limit: 1 },
]);
```

Aggregation - $project

```
db.employees.aggregate([
  {
    $project: {
      "name": 1,
      "email": 1,
      "salary": 1
    }
  },
  {
    $limit: 2
  }
])
```

$project – remove field

```
db.employees.aggregate([{ $project: { _id: 0, name: 0 } }]);
```

$project – rename & add calc

```
db.employees.aggregate([
  {
    $project: {
      empname: "$name",
      email: 1,
      salary: 1,
      AnnualSalary: { $multiply: [12, "$salary"] },
    },
  },
]);
```

Aggregation - $sort

```
db.employees.aggregate([
  {
    $sort: { "name": -1 }
  },
  {
    $project: {
      "name": 1,
      "email": 1,
      "salary":1
    }
  },
  {
    $limit: 5
  }
])
```

Aggregation - $addFields, $cond

```
{ $cond: [ <boolean-expression>, <true-case>, <false-case> ] }
...................................
db.employees.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      salary: 1,
      grade: { $cond: [{ $gte: ["$salary", 2000] }, "Grade A", "Grade B"] },
    },
  },
]);
```

Aggregation - $addFields -$cond-if

```
{ $cond: { if: <boolean-expression>, then: <true-case>, else: <false-case> } }
..........................
db.employees.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      salary: 1,
      grade: {
        $cond: {
          if: { $gte: ["$salary", 2000] },
          then: "Grade A",
          else: "Grade B",
        },
      },
    },
  },
]);
```

Aggregation - $lookup prep

```
db.createCollection("orders")


db.orders.insertOne({'empid':ObjectId('65fc6dd2198f1b870853d26e'),'date':
Date(),'orderValue':5000})
```

Aggregation - $lookup – orders to emp

```
db.orders.aggregate([
  {
    $lookup: {
      from: "employees",
      localField: "empid",
      foreignField: "_id",
      as: "employee_details",
    },
  },
  {
    $limit: 1
  }
])
```

Aggregation - $lookup – emp to orders

```
db.employees.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "empid",
      as: "Orders",
    },
  },
]);
```

Aggregation - $out (creates ratingbydep collection)

```
db.employees.aggregate([
    {
      $project: {
        name: 1,
        department: 1,
        rating:{$convert:{input:"$rating",to:"int"}}
      },
    },
    { $group: { _id: "$department", avg: { $avg: "$rating" } } },
    {$out:"ratingByDep"}
  ]);
```

Views

```
db.createView(
  "activeUsers",
  "users",
  [
    { $match: { isActive: true } },
  ]
)

db.activeUsers.find()
db.activeUsers.drop()
```

Backup and Restore - Tool

Download MSI version using below link:

https://www.mongodb.com/try/download/database-tools

Click on the downloaded file and install

Setup environment variables to add path

C:\Program Files\MongoDB\Tools\100\bin

Backup Steps

//backup of a particular database

mongodump -d mydb -o d:/bck //d means data

//backup of a particular collection

mongodump -d mydb -c employees -o d:/bck //c means collection

//backup of all the databases

mongodump -o d:/bck //o means output

Restore Steps

```
//to restore a particular database
mongorestore -d mydb d:/bck/mydb

//to restore a particular collection
mongorestore -d mydb -c employees d:\bck\mydb\employees.bson

//to restore all the databases
mongorestore --dir d:\bck\

//creates a new database and then restores
mongorestore -d mydbnew -c employees d:\bck\mydb\employees.bson

//creates a new collection and then restores
mongorestore -d mydbnew -c employees d:\bck\mydb\employees.bson
```

MongoDB – Regex

```
db.employees.find({name:{$regex:’Cathy’}}) //consists Cathy
db.employees.find({name:{$regex:“cathy",$options:"i"}}) // case insensitive
 db.employees.find({name:{$regex:"^C"}}) // starts with C
db.employees.find({name:{$regex:“y$"}}) //ends with y
```

Mongodb cluster

Replica Set

Replica of data is created

Sharded cluster

Parts of data is stored in different machine,..used in very large database

Mongodb Replication - 1

Create a folder mongo-replica and sub folders data1 data2 and data3

Open command prompt and start running servers on separate tabs

```
mongod -replSet rs1 -logpath "d:\mongo-replica\data1\1.log" --dbpath
"d:\mongo-replica\data1" --port 27018
```

```
mongod -replSet rs1 -logpath "d:\mongo-replica\data2\2.log" --dbpath
"d:\mongo-replica\data2" --port 27019
```

```
mongod -replSet rs1 -logpath "d:\mongo-replica\data3\3.log" --dbpath
"d:\mongo-replica\data3" --port 27020
```

Mongodb Replication - 2

Follow these instructions to configure replica set:

```
mongosh - -port 27018
```

```
rs.initiate({_id:"rs1",members:[{_id:0,host:"127.0.0.1:27018"},{_id:1,host:"127.0.0.1:27019"},{_id:2,host:"127.0.0.1:27020"}]})
```

```
rs.config()  //to check the config
```

```
rs.status()
```

Mongodb Replication - 3

Use mongosh command with the following connection string and the primary server will automatically get connected:

```
mongosh
"mongodb://localhost:27018,localhost:27019,localhost:27020/?replicaSet=r
s1"
```

```
show dbs
```

```
use mytestdb
```

```
db.createCollection("customers")
```

```
db.customers.insertOne({name:"John"})
```

Mongodb Replication - 4

Check secondary servers. Check both the servers if data is replicated

```
mongosh --port 270xx
Secondary will start, can read but cannot write
db.getMongo().setReadPref("secondary")  //or rs.secondaryOk()
use mytestdb
db.customers.find() – will work now
```

```
mongosh --port 270xx
Secondary will start, can read but cannot write
db.getMongo().setReadPref("secondary")  //or rs.secondaryOk()
use mytestdb
db.customers.find() – will work now
```

Mongodb Replication - 5

Shutdown primary server and the primary will be automatically changed to one of the other two servers

Go to primary 270xx
Use admin
db.shutdownServer()

Now go to secondary servers 270xx or 270xx, and type show dbs…you would notice that one of the servers will be changed to primary automatically

----

Open new tab and start previous primary 270xx again

```
mongod -replSet rs1 -logpath d:\mongo-replica\data1\1.log --dbpath d:\mongo-replica\data1\ --port 270xx
```

Open another tab and run mongosh. You will observe that it is now a secondary server.

```
mongosh --port 270xx
```

Sharding

shard: a small piece or part

Sharding is a Horizontal Scaling method that distributes data across multiple machines compared to vertical scaling where capacity of single server is increased to the maximum.

Sharding - 1

Create folder dbshards and then create sub folders: conf, rconf, s1, s1r, s2, s2r

Start Config servers on separate tabs of command prompt

```
mongod --configsvr --port 27018 --replSet cf --dbpath d:\dbshards\conf
```

```
mongod --configsvr --port 27019 --replSet cf --dbpath d:\dbshards\rconf
```

Open new tab and Initiate replica set for config servers

```
mongosh --port 27018
```

```
rs.initiate({_id:'cf',members:[{_id:0,host:'localhost:27018'},{_id:1,host:'localhos
t:27019'}]})
```

Sharding - 2

Start Shard1 servers on separate tabs of command prompt

```
mongod --shardsvr --port 27020 --replSet rs1 --dbpath d:\dbshards\s1
```

```
mongod --shardsvr --port 27021 --replSet rs1 --dbpath d:\dbshards\s1r
```

Open new tab and Initiate replica set for shard1 servers

```
mongosh --port 27020
```

```
rs.initiate({_id:'rs1',members:[{_id:0,host:'localhost:27020'},{_id:1,host:'localhost:27021'}]})
```

Sharding - 3

Start Shard2 servers on separate tabs of command prompt

```
mongod --shardsvr --port 27022 --replSet rs2 --dbpath d:\dbshards\s2
```

```
mongod --shardsvr --port 27023 --replSet rs2 --dbpath d:\dbshards\s2r
```

Open new tab and Initiate replica set for shard2 servers

```
mongosh --port 27022
```

```
rs.initiate({_id:'rs2',members:[{_id:0,host:'localhost:27022'},{_id:1,host:'localh
ost:27023'}]})
```

Sharding - 4

Start Mongo Routing Service on separate tab of command prompt

```
mongos  --configdb cf/localhost:27018,localhost:27019 --port 27050
```

Sharding - 5

Now connect to 27050 and add shards

```
mongosh --port 27050
sh.addShard("rs1/localhost:27020,localhost:27021")
sh.addShard("rs2/localhost:27022,localhost:27023")
sh.status()
use mydatabase
sh.enableSharding("mydatabase")
sh.shardCollection("mydatabase.customers", { _id: 1 })
sh.status()
sh.getShardedDataDistribution() //run this after executing below nodejs scripts
```

Sharding - Insert dummy data

```
import { MongoClient } from "mongodb";
const uri = "mongodb://127.0.0.1:27050/“
const client = new MongoClient(uri);
async function insertTestData() {
  try {
    await client.connect();
    const db = client.db("mydatabase");
    const collection =   db.collection("customers");
    // const res = await collection.countDocuments()
    // console.log(res)
    const bulk = [];
    for (let i = 0; i < 90000; i++) {
      bulk.push({
        userId: i,
        name: `User${i}`,
        email: `user${i}@test.com`,
        createdAt: new Date(),
      });
    }
    const result = await collection.insertMany(bulk);
    console.log(`Inserted ${result.insertedCount} documents.`);
  } catch (err) {
    console.error("Error inserting data:", err);
  } finally {
    await client.close();
  }
}
insertTestData();
```

Sharding - Verify Shard servers

```
mongosh --port 27020
show dbs
//if mydatabase exists then run below two commands
use mydatabase
db.customers.countDocuments()
```

```
mongosh --port 27022
show dbs
use mydatabase
db.customers.countDocuments()
```

Note:Keep running the nodejs script and you will observe that mydatabase appears on both the servers.

Open mongo routing service and check the distribution

```
mongosh --port 27050
sh.status()
sh.getShardedDataDistribution()
```

Over a period of time orphanDocument will become 0. It gets created if documents gets created in wrong shard. Observe numOwnedDocuments on both the shards

To verify secondary servers run following command:

```
db.getMongo().setReadPref("secondary")  //or rs.secondaryOk(
```

User Management - 1

```
use admin
```

```
db.createUser({
  user: "admin",
  pwd: "admin",
  roles: [ { role: "root", db: "admin" } ]
})
```

add following settings in mongod.conf available in program files / mongodb

```
security:
  authorization: enabled
```

go to services and restart mongodb server

User Management - 2

mongosh will still connect but you can't run any command so try with following options:

```
mongosh --username admin --authenticationDatabase admin  //for prompt
```

```
mongosh --username admin --password admin --authenticationDatabase admin
//without prompt
```

connect using mongodb compass using following connection string

```
mongodb://admin:admin@localhost:27017/
mongodb://admin:admin@localhost:27017/?authSource=admin
mongodb://admin:admin@localhost:27017/mydb?authSource=admin
```

User Management - 3

```
use mydb

db.createUser({
  user: "user1",
  pwd: "1234",
  roles: [
    { role: "read", db: "mydb" }
  ]
})

db.getUsers()

mongosh --username user1 --authenticationDatabase mydb

db.dropUser("user1")
```

Thank You

- PRAVEEN NAIR