MONGO-DB

use mydb

db.dropDatabase()

db - current working db

Display databases: show dbs or databases

Create Collection: db.createCollection(“Employee”)

Display Collections: show collections

Delete Collection: db.Employee.drop()

Use mydb

db.createCollection(“Employee”)

Insert Document in Collection: **insert()** **insertOne()** **insertMany()**

db.Employee.insert({EmpId: 101, Name: ‘Akash’, Age: 23, Dept: ‘IT’})

db.Employee.insert([

{EmpId: 102, Name: "Rahul", Age: 22, Dept:"Sales"},

{EmpId: 103, Name:"Karan",Age:23,Dept:"HR"}

])

Retrieve Documents:

find() -all **findOne() -retrieves 1st occurence** **find().limit(5)**

db.Employee.find() db.Employee.find().limit(5)

db.Employee.findOne()

db.Employee.find({dept: ‘Sales’}) **-- Prints Emps in SALES dept**

db.student.find({cgpa: {$gt:8, $tt:9}, {“_id”: 0} }) **-- without id**

db.Employee.find({Name:"Akash"},{"Name":1,"Age":1**,"_id":0**}) – id not displayed

db.students.find({}, { name: 1, age: 1, _id: 0 }) // Only return name and age

MONGODB-CURSOR LOOPS

var cur=db.student.find({cgpa: {$gt:8, $tt:9} })

cur. next()

cur.forEach(**printjson**) -- **prints in json format**

next() is a method which returns the current document.

forEach() iterates all documents.

printjson prints the document in JSON format.

var cur=db.student.find({cgpa: {$gt:8, $tt:9} })

cur. forEach(printjson) **--print all with cgpa between 8 and 9 in JSON format**

UPDATE documents: update() updateOne() updateMany()

db.Employee.update({'Name':'Rahul'},{$set: {'Name': 'BMW'}})

Increment Salary of all by 2000

db.Employee.updateMany({},{$inc: {salary: 2000}}) -- increments all by 2000

Increment Salary of all in Sales dept by 5000

db.Employee.updateMany({department: 'Sales'}, {$inc: {salary: 5000}})

$mul: Multiply with specific value

db.Employee.updateMany({department: 'IT'}, {$mul: {salary: 1.5}}) **--multiply by 1.5**

$min – Updates the values of the field to a specified new value if the new value is less than the current value of the field. Suppose salary= [1400, 2000, 2500, 3500]

db.Employee.UpdateMany({dept: ‘Sales’},{$min: {salary: 2700}}) --will only change **3500 to 2700** since 2700(new value is min than existing)

$max** : db.Employee.updateMany({dept: ‘IT’},{$max: {salary: 2200}}) –will change **1400 and 2000 to 2200, since 2200 is greater than 1400 and 2000.

REGEX-patterns

db.Employee.find({'name':/^a/i})   -- Name starts with A, **i means case-insensitive**

db.Employee.find({$and: [{cgpa:{$gt:8, $lt:9}}, {‘name’: /^a/i}] })

db.students.find({ Email: /@vit\.ac\.in$/ }) --end with

db.students.find({ RegisterNumber: /^24/ }) --start with

db.students.find({ RegisterNumber: /^\d{2}MCA\d{4}$/ })

SORT

db.students.find().sort({ CGPA: -1 }).limit(5)

db.student.find().sort({name:1, cgpa:-1})

db.student.find().sort({cgpa:-1. name:1}) // **CORRECT**

highest cgpa then with name in ASC order

DELETE: deleteOne()   deleteMany()

db.Employee.deleteOne({Name: 'BMW'})

db.Employee.deleteMany({ dept: 'HR' })

db.Employee.find({age: {$gte: 25, $lte: 35} }) –**Age between 25 and 35 included**

db.Employee.find({$and: [{State: {$nin: [‘Assam’,’Bihar’]}}, {age: {$gt:18,$lt:40}}] })

**$eq, $ne:** db.Employee.find({age: {$eq: 25} })

**$lt, $lte, $gt, $gte:** db.Employee.find({age: {$gte: 18} })

**$in:** db.Employee.find({State: {$in: ['Assam', 'Punjab', 'Bihar']} }) — Match any value in an array

**$nin:** db.Employee.find({Program: {$nin: ['MBA','MSW']} }) --Match if not in the array

**$and:** db.Employee.find({$and: [{Program: 'MCA'},{cgpa: {$gt: 8} }] }) --Match all conditions

**$or:** db.Employee.find({$or: [{Program: 'MBA'},{cgpa: {$gt: 9}}] }) — Match any one condition

**$nor:** db.Employee.find({$nor: [{Program: 'MSW'},{State: 'Tamil Nadu'}] }) --Match if none of the conditions are true

**$not:** db.Employee.find({age: {$not: {$gte: 18}} }) — Negates a condition

**$type**: db.Employee.find({PhNumber: {$type: "string"} }) --int,double,bool,array,object

**$exists:** db.Employee.find({Email: {$exists: true} }) //Check if a field exists or not

Insert One Document with Array

```
db.Employee.insertOne({
  name: "Ravi",
  age: 30,
  State: "Karnataka",
  skills: ["Java", "MongoDB", "Node.js"]
})
```

db.Employee.find({ skills: "MongoDB" }) -- **Find Documents with a Specific Skill**

Bulk insert 3 student documents into the students collection

```
db.students.insertMany([
  {
    RegisterNumber: "24MCA0001",
    Program: "MCA",
    CGPA: 3.8,
    Email: "rahul@vit.ac.in",
    Address: "123 Street A",
    PhoneNumber: "9876543210",
    City: "Chennai",
    State: "Tamil Nadu"
  },
  {
    RegisterNumber: "24ME0002",
    Program: "ME",
    CGPA: 7.5,
    Email: "aarti@vit.ac.in",
    Address: "456 Street B",
    PhoneNumber: "9012345678",
    City: "Jaipur",
    State: "Rajasthan"
  }
])
```

B. **List students who joined in 2024 and have CGPA less than 4**

```
db.students.find({RegisterNumber: /^24/,  CGPA: {$lt: 4} })
```

C. **Increase CGPA by 0.01 for students with CGPA between 7.0 and 8.5**

db.std.updateMany({}, {$inc: {} })

```
db.students.updateMany(
  { CGPA: { $gte: 7.0, $lte: 8.5 } },
  { $inc: { CGPA: 0.01 } }
)
```

D. **List students of 2023 MBA batch with CGPA > 8 and from Rajasthan, Bihar, or Punjab**

```
db.students.find({
  RegisterNumber: /^23/,      // 2023 batch
  Program: "MBA",
  CGPA: { $gt: 8 },
  State: { $in: ["Rajasthan", "Bihar", "Punjab"] }
})
```

AGGREGATE

Count students in each State

```
db.students.aggregate([
  {
    $group: {
      _id: "$State",
      count: { $sum: 1 }
    }
  }
])
```

Match + Group: Count low scorers (CGPA < 6) by Program

```
db.students.aggregate([
  { $match: { CGPA: { $lt: 6 } } },
  {
    $group: {
      _id: "$Program",
      lowScorers: { $sum: 1 }
    }
  }
])
```

Q. write me a program fro Employee Management using nodejs and mongodb connectivity

Folder Structure

```
employee-management/
│
├── app.js
├── models/
│   └── Employee.js
```

1. Initialize Node.js Project

```
mkdir employee-management
cd employee-management
npm init -y
```

2. Install Required Packages

```
npm install express mongoose body-parser
```

3. Create models/Employee.js

```
// models/Employee.js
const mongoose = require('mongoose');

const employeeSchema = new mongoose.Schema({
  name: String,
  age: Number,
  department: String,
  salary: Number
});

module.exports = mongoose.model('Employee', employeeSchema);
```

💻 4. Create Main File app.js

```
// app.js
```

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const Employee = require('./models/Employee');

const app = express();
app.use(bodyParser.json());

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/employeeDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log("MongoDB connected"))
  .catch((err) => console.log(err));

// Add New Employee
app.post('/employee', async (req, res) => {
  try {
    const emp = new Employee(req.body);
    await emp.save();
    res.status(201).send(emp);
  } catch (err) {
    res.status(400).send(err);
  }
});

// Get All Employees
app.get('/employee', async (req, res) => {
  const employees = await Employee.find();
  res.send(employees);
});

// Get Employee by ID
app.get('/employee/:id', async (req, res) => {
  const emp = await Employee.findById(req.params.id);
  res.send(emp);
});

// Update Employee by ID
app.put('/employee/:id', async (req, res) => {
  const emp = await Employee.findByIdAndUpdate(req.params.id, req.body, { new: true });
  res.send(emp);
});

// Delete Employee by ID
app.delete('/employee/:id', async (req, res) => {
  await Employee.findByIdAndDelete(req.params.id);
  res.send({ message: "Employee deleted" });
});

//  Start Server
app.listen(3000, () => {
  console.log("Server running on http://localhost:3000");
});
```

🛠 5. Start MongoDB and Run App

Start MongoDB server (if local):

```
Cmd> mongod
```

Run your Node.js app:

```
Cmd> node app.js
```