**NODE**

**Node 1st basic setup**

Create a folder > Open VS code > Terminal > npm init (For package.json) or npm init -y (For default package.json) > If you use default then you have to change main file index.js to app.js in package.json file > create a file app.js

**Use http module**

const http = require('http');

http.createServer((req, res) =>{

    res.write("Hello World")

    res.end

}).listen(3005)

Write this code in app.js > node ./app.js > enter then server will be start

**Use File system (fs) module**

**Append file (Add Data in File)**

Create a blank text file hello.txt >

// fs module

fs.appendFile('hello.txt','Hello World',(error) =>{

    if(error){

        console.log(error);

    }

})

write this code in app.js > node ./ app.js > Then Hello World will be added in hello.txt

**Delete File (Using unlink)**

// fs module (Unlink)

fs.unlink('hello.txt', (error) => {

    if (error) {

        console.log(error);

    }

})

**Path Module**

console.log(path.join(\_\_dirname,'hello.txt'));

It will be tell hello.txt is present which file path

**EXPRESS**

Go to terminal > npm i express

**How to Create Server in Express ?**

const express = require('express'); // Import express

const app = express(); // Store into app variable

// This is Home Page

app.get('/',(req, res) =>{

    res.send("<h1>Home Page</h1>")

})

// This is About Page

app.get('/about',(req, res) =>{

    res.send("<h1>About Page</h1>")

})

// This is Contact Page

app.get('/contact',(req, res) =>{

    res.send("<h1>Contact Page</h1>")

})

// Port

const port = 3004

app.listen(port,() =>{

    console.log(`Server is Running on port ${port}`);

})

node./app.js > Server will be start on localhost:3004

**How to use nodemon for to run server all time ?**

npm i nodemon

Go to package.json > "start":"nodemon app.js",> write this in script area

**MVC File Setup**

app.js : It is main rendering file.

1) Create a folder app ,inside app folder there will be folder : config, controller , middleware, model, router.

2) Create a public folder where I will keep my all static data just like image .

3) Create views folder where will keep all html data which will be ejs format. npm i ejs

4) Create uploads folder where I keep my mongoDB data base data.

5) In controller folder there will a file just like my controller.js where keep all api activity.

class mycontroller {

    async home (req, res){

        //res.send('<h1>Home Page')

        res.render('home')

    }

    async about (req, res){

        //res.send('<h1>About Page')

        res.render('about')

    }

    async contact(req, res){

       // res.send('<h1>Contact Page')

       res.render('contact')

    }

}

module.exports = new mycontroller();

6) In router folder there will be keep my allrouter.js

const express = require('express');

const mycontroller = require('../controller/mycontroller');

const router = express.Router();

router

    .get('/', mycontroller.home)

    .get('/about', mycontroller.about)

    .get('/contact', mycontroller.contact)

module.exports = router;

7) ejs file will be as it is html type data.

**ejs installation and process**

1) npm i ejs

2) Keep all ejs file in views folder

3) In app.js

const app = express(); // Store into app variable

app.set('view engine','ejs'); // View Engine of ejs

app.set('views','views'); // Run views file

**How to use Bootstrap Or Bootstrap Template in ejs**

npm i ejs > create a views folder there will be keep two folders common and uncommon > In common folder there will be keep all common file just like html head tag area , navbar, footer etc > In uncommon folder there will be keep all page just like home, about etc.> Replace all uncommon file’s <head> tag with<%-include('../common/head.ejs') %>this code and where you want to put nav or footer there also put this code but file name will be different > If you use template then put template’s all html and css file in public folder >

app.use(express.static(\_\_dirname + '/public')); write this code in your main file just like index.js or app.js > Because I keep all page in uncommon folder show I have to put uncommon async home(req, res) {

        // I use 'uncommon/' because home is present in uncommon folder which is present in view

        res.render('uncommon/home',

            {

                title: 'Home Page'

            }

        )

    }

Here > Put other body code in all uncommon file as it is.

**How to show json data on ejs template ?**

Create a json file just like data.json > all controller file >

allcontroller.js

const fs = require('fs'); // For File Read

const path = require('path'); // Import path to work with file paths

// Blog page data.json handling area

    async blog(req, res) {

        const jsonFilePath = path.join(\_\_dirname, '../database/data.json');

        const data = await fs.readFileSync(jsonFilePath, 'utf8');

        const blogData = JSON.parse(data); // Convert JSON data to normal object

        res.render('uncommon/blog', { blog: blogData });

    }

Put this code on controller .Here I put data in blog show I use it on blog controller >

blog.ejs

 <% blog.posts.forEach(post => { %>

            <article class="entry" data-aos="fade-up">

              <div class="entry-img">

                <img

                  src="<%= post.img %>"

                  alt="<%= post.title %>"

                  class="img-fluid"

                />

              </div>

              <h2 class="entry-title">

                <a href="blog-single.html"><%= post.title %></a>

              </h2>

              <div class="entry-meta">

                <ul>

                  <li class="d-flex align-items-center">

                    <i class="icofont-user"></i>

                    <a href="blog-single.html"><%= post.author %></a>

                  </li>

                  <li class="d-flex align-items-center">

                    <i class="icofont-wall-clock"></i>

                    <a href="blog-single.html">

                      <time datetime="<%= post.date %>"

                        ><%= new Date(post.date).toLocaleDateString() %></time

                      >

                    </a>

                  </li>

                  <li class="d-flex align-items-center">

                    <i class="icofont-comment"></i>

                    <a href="blog-single.html">0 Comments</a>

                    <!-- Placeholder for comments -->

                  </li>

                </ul>

              </div>

              <div class="entry-content">

                <p><%= post.content.substring(0, 150) %>...</p>

                <div class="read-more">

                  <a href="blog-single.html">Read More</a>

                </div>

              </div>

            </article>

            <!-- End blog entry -->

            <% }); %>

Run forEach loop like this on ejs file.

**MongoDB**

**Setup**

Dowload and Install MongoDB compus app > chrome > MongoDB atlast > Try Free > Create account > Continoue with Google > Fillup form and create account > Deploy your cluster > Select MO Free > Create Deployment > Solve Puzzle > Open a small interface > Close > Network Access > Add an IP Address > Allow Access for anywhare > Confirm > Click on Database Access > Add new Database user > Authentication Secure Password > Copy that password on notepad > put user name > Click on Build in Role > Atlas Admin > Add user.

Go to Network access show it is activate.

After active Click on Clusters > Connect > MongoDB for VS code > Copy that Link and Save on Notepad > Done

Open MongoDB Compus > Add new connection > Paste the link which you copied from mongoDB cloud > Put your password on password area like this: mongodb+srv://mousom:el3Rx4KaOJZNsHsH@cluster0.92voh.mongodb.net/ > Save and Connect.

After completing next step is add data in database using vs code .

**Conecting code with VS code and make API**

npm i mongoose > Create an .env file npm i dotenv and make it this file in global variable > MONGO\_URL = mongodb+srv://mousom: el3Rx4KaOJZNsHsH@cluster0.92voh.mongodb.net/crud

Keep this code in .env file> Create a folder in your app folder config > In config folder make a file db.js >

**db.js**

const mongoose = require('mongoose');

const connectDB = async () => {

    try {

        const conn = await mongoose.connect(process.env.MONGO\_URL)

        console.log('MongoDB connected')

    } catch (error) {

        console.log('mongodb error', error)

    }

}

module.exports = connectDB;

**app.js**

const express = require('express'); // Import For Express

const dotenv = require('dotenv'); // For .env file

const cors = require('cors'); // For to run different server when I run use React with node

const connectDB = require('./app/config/db.js'); // Connect Database

dotenv.config(); // .env with config

const app = express();

connectDB()

app.use(express.json()); // use Express

app.use((cors())); // Use Cors

//For API Routing

const allrouter = require('./app/router/allrouter');

app.use('/api', allrouter);

const port = 3004

app.listen(port, () => {

    console.log(`Server is running on port ${port}`);

});

Put this code on your app.js or index.js file > Run server npm start message will come mongodb connected.

Now Create a Api for CRUD.

Create a folder in app name will be model > in model folder create a file student.js >

**student.js**

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const StudentSchema = new Schema({

    name: {

        type: String,

        required: true,

        minlength: [3, 'Name must be at least 3 characters long']

    },

    email: {

        type: String,

        required: true,

        match: [/^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/, 'Email address should follow the format: abc@gmail.com']

    },

    phone: {

        type: Number,

        required: true,

        min: [1000000000, 'Phone number must be exactly 10 digits'],

        max: [9999999999, 'Phone number must be exactly 10 digits']

    }

}, { timestamps: true }); // timestamps show create date and update date

const StudentModel = mongoose.model('student', StudentSchema);

module.exports = StudentModel;

This is my student.js code > allRouter.js will be as same as previous >

**allcontroller.js**

const Student = require('../model/student');

class allcontroller {

    // Create API

    async create(req, res) {

        try {

            const studentdata = new Student(req.body);

            const data = await studentdata.save();

            res.status(200).json({ message: "Data added successfully", data });

        } catch (error) {

            const statusCode = error.name === 'ValidationError' ? 400 : 500;

            const message = error.name === 'ValidationError'

                ? { message: "Validation error", errors: Object.values(error.errors).map(err => err.message) }

                : { message: "An unexpected error occurred" };

            console.error(error);

            res.status(statusCode).json(message);

        }

    }

    // Get API

    async getall(req, res) {

        try {

            const data = await Student.find()

            res.status(200).json({

                message: "Data get successfully",

                total: data.length,

                students: data

            })

        } catch (error) {

            console.log(error);

            res.status(500).json({ message: "Error retrieving student data" });

        }

    }

    // Get Single

    async getsingle(req, res) {

        try {

            const id = req.params.id;

            const data = await Student.findById(id);

            if (data) {

                res.status(200).json(data);

            } else {

                res.status(404).json({ message: "Student not found" });

            }

        } catch (error) {

            console.log(error);

            res.status(500).json({ message: "Error retrieving student data" });

        }

    }

    // Update Data

    async studentupdate(req, res) {

        try {

            const updatedStudent = await Student.findByIdAndUpdate(

                req.params.id,

                req.body,

                { new: true, runValidators: true }

            );

            if (!updatedStudent) return res.status(404).json({ message: "Student not found" });

            res.status(200).json({ message: "Student updated successfully", data: updatedStudent });

        } catch (error) {

            const statusCode = error.name === 'ValidationError' ? 400 : 500;

            const message = error.name === 'ValidationError'

                ? { message: "Validation error", errors: Object.values(error.errors).map(err => err.message) }

                : { message: "Error updating student data" };

            console.error(error);

            res.status(statusCode).json(message);

        }

    }

    // Delete Student

    async studentdelete(req, res) {

        try {

            const deletedStudent = await Student.findByIdAndDelete(req.params.id);

            res.status(deletedStudent ? 200 : 404).json(

                deletedStudent ? { message: "Student deleted successfully" } : { message: "Student not found" }

            );

        } catch (error) {

            console.error(error);

            res.status(500).json({ message: "Error deleting student" });

        }

    }

}

module.exports = new allcontroller()

Now For post API use check it using post man and for get api you check in browser. As same as previous process.

For to see full project : <https://github.com/mousomray/student_crud_api_creation_node_express_mongoDB>

**Make ejs CRUD and make backend validation**

i) For validation we put validation in to schema that means in model folder. For to show array data in api we have to put

p\_size: {

        type: Array, // Array of strings for sizes

        validate: {

            validator: function (v) {

                return v.length > 0; // Must have at least one size

            },

            message: "Please select atleast one size."

        }

    },

ii) Don’t write required : true always write required : put a message .

iii) Validation will be active in controller page. (code is present in github)

iv) For to perform crud in ejs first we have to enable urlencoded.

app.use(express.urlencoded({ extended: true }));//The code `app.use(express.urlencoded({ extended: true }));` allows your Express app to parse URL-encoded data from form submissions, enabling access to form fields in `req.body`. extended:true allow nested object and false doesn't allow nested object

v) We have to enable method override for overriding method . Becasue form field does not support PUT, PATCH,DELETE.

npm i method-override

const methodOverride = require('method-override');//method-override is used to allow form submissions in EJS to handle HTTP methods like DELETE or PUT by overriding them with POST.

app.use(methodOverride('\_method')); //Enable method override for DELETE and PUT or PATCH requests

vi) For ejs CRUD we have to maintain that file folder set up

Controller : 2 controller apicontroller.js and uicontroller.js

Router : 2 router apirouter.js and uirouter.js

Views : Different views file for different page

vii) For best dynamic routing in ejs

router.get('/product/:id/detailsproduct', productcontroller.detailsproduct)//Product details page

router.get('/product/:id/editproduct', productcontroller.singleproduct) // It is Single Product

Use this type of process /product/:id/detailsproduct for to perfect route in particular page. Use this type of routing only uirouter page.

Viii) Where you handle delete or edit form where your form action will be

<form

                    action="/product/<%= value.\_id %>?\_method=DELETE"

                    method="POST"

                    style="display: inline"

                  >

?\_method=DELETE it is method overriding . For edit you habe to use =PUT

ix) Details product button will be like

<div style="display: flex; justify-content: center; gap: 10px">

                  <form

                    action="/product/<%= value.\_id %>/detailsproduct"

                    method="GET"

                    style="display: inline"

                  >

                    <button

                      type="submit"

                      style="

in action= “/product/<%= value.\_id%>/detailsproduct” means after clicking button it will be go to details product it is confirmed.

For to see full project : <https://github.com/mousomray/product_employee_api_and_ui_creation_ejs_node_express_mongoDB>

**How to Handle Form Data (Image post in API)**

npm install multer > Create a helper folder in app folder > in help folder create a js file a code will present in that js file where we handle image just like image type , size etc > create a folder name will be uploads create in normal not in app folder all image will be store there > app.use('/uploads', express.static(\_\_dirname + '/uploads')); do it in app.js file>Go to api controller file and put image handling code in create and update area code is present in github > Go to API routing area and put image handling code in create and update are code is also present in github.

**Handle Form data in ejs**

First handle ui controller file add and update area > Handle routing as same as which was did in api routing code also present in github > In view file <form action="/addemployeecreate" method="POST"enctype="multipart/form-data"> use enctype for handling form data > make image input type will be file. > Update page and show data page will be as same as previous but some changes are make all code are present in github.

github : <https://github.com/mousomray/product_employee_api_and_ui_creation_ejs_node_express_mongoDB>

**How to do JWT authentication ?**

Create a schema model user which is our authentication api model > Go to .env file and keep your api key which will be a random text or number . > Create a file middleware and create a file auth.js keep your authentication logic here just like password hasing function, token varification etc > Create a controller file authcontroller.js here we handle our actual authentication system just like login, register, userdashboard, forgetpassword etc > make a routing system for login, register and others. But which api you want to keep in under the JWT where you put Auth in routing area Auth is came from in middleware Auth.js file which is our Auth function. All code present in github.

For to see full project : <https://github.com/mousomray/tourist_student_crud_auth_api_creation_node_express_mongoDB>

**How to perform private routing in ejs ?**

npm i cookie-parser and npm i express-session > use cookie-parser and session in your app.js file code is present in github > Go to middleware folder and create another auth file uiauth.js there will be present our private routing logic > Handle your uiauthcontroller with private routing logic but every ejs file will be render using {user:req.user} this code for to available user in all get request.Code is present in github > Which page you want to keep under private routing please put uiAuth on your that routing area just like router.get('/studentlist', uiAuth, studentuicontroller.getall);

> Put your private routing logic in your navbar.js file

See full project : <https://github.com/mousomray/tourist_student_crud_auth_email_verification_api_ui_creation_node_express_mongodb>

**How to make my email ready for to verify user ?**

Go to managing your google account > Security > Turn on 2 step verification > Turn on 2 step verification > Done > Back to search and type App Password > App Passwords > Open notepad > create app name just like node js auth > create > copy the password and paste it on notepad and save for reuse.

Now ready for coding :

npm install nodemailer > Keep my secreat thing in my .env file just like

EMAIL\_HOST = 'smtp.gmail.com'

EMAIL\_PORT = 587

EMAIL\_USER = 'sayantanroy184@gmail.com'

EMAIL\_PASS = 'pzch xjff nfmq bndk'

EMAIL\_FROM = 'sayantanroy184@gmail.com'

> you have to add a field in your user schema isverified and roles code is present in github > create an another schema otpVerified.js it is references model it will be our email verification model > create an emailvarification function in your helper folder where actually present the body of our otp >which transporter are you show , that transpoter function you have to create in config folder > call the emailverification function in your register controller area. >Create a OTP verify controller and route for to verify your email > Put a logic in login area if user is not verified then will be show message > check your otp valid or not check it using postman put email and otp field > Login your account.

Which condition you add in register and login api controller same logic apply in your ui login , register area.

See full project : <https://github.com/mousomray/tourist_student_crud_auth_email_verification_api_ui_creation_node_express_mongodb>

**How to perform Forget Password with user and email verification?**

First we have to create a controller resetpasswordlink in authcontroller.js file and here will be present a link which will be our actual forget password page frontend file plaese keep your .env file > create a resetpasswordlink router in authrouter.js file > Create a controller for forget password > Create a router for forget password with route parameter :id and :token >

This is for API handling now process to handle ejs.

First you have to create a resetpasswordlink ui controller in uiauth.js file logic almost same as handling api controller > You have to create two routing for reset password form and add data into that form > you have to create a passwordreset form passwordreset.ejs > Create a forget password ui controller > Create two forget password form router with two parameters :id and :token > Create a forget password actual form forgetpassword.ejs which will be our actual forget password form there will be also add parameter in your create form action .

See full project : <https://github.com/mousomray/tourist_student_crud_auth_email_verification_api_ui_creation_node_express_mongodb>

# **How to setup webskitter file folder structure ?**

npm i route-label

npm i morgan

npm i body-parser

# **Coding area**

Write important Application credential on .env file

1. Create a file index.js in your config folder and define apllication configaration
2. Create a module utils in helper folder and handle utils module
3. Create a folder module there will be all folders which will be our pages just like product,about, banner etc.
4. In module nested folder just like product there will be three products controller, model and repository.
5. Controller will be our admin controller , model will be our api model and in repository folder there will be keep our mongoDB code
6. Create a router folder in module there will be api, admin and auth router which will be defined in .env file
7. Create a webservice folder all api controller will be present there
8. Handle this whole name routing system in app.js file code is already present
9. All route will be operate through route lebel not direct route just like <%= generateUrl('product')%> product is my route name

# **How to Apply CK editor?**

Put ck editor js link in your common head.ejs file

1. Put ck edit code in your ejs with script tag
2. Which field you want to keep under ckEditor that input should be textarea
3. When you show textarea field when you show list of data for ejs you have to use<%- %> in the place of <%= %> and for react you have to use dangeraously set innerhtml

For to see full project : <https://github.com/mousomray/nodeJS_webskitter_folder_structure-product_auth_handle>

**How to implement pagination in nodeJS ?**

Code is present in github normal mvc : <https://github.com/mousomray/student_crud_api_creation_node_express_mongoDB>

Code is also present in github webskitters mvc :

<https://github.com/mousomray/role_authorization_mern/tree/master/server>

**There are two types of search query params and search using post data how to implement it ?**

Full code : <https://github.com/mousomray/nodeJS_webskitter_folder_structure-product_auth_handle>

**How to put multiple image in node and next ?**

Check from here: <https://github.com/mousomray/multiple_image_handle_mern_auth_nextJS>

These are github code for uploads project in github

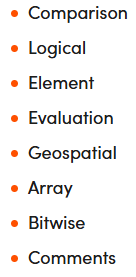
git rm -r --cached uploads // For to untrack folder to git

Remove-Item -Recurse -Force client\.git / For uploading file which is alrady in git repository

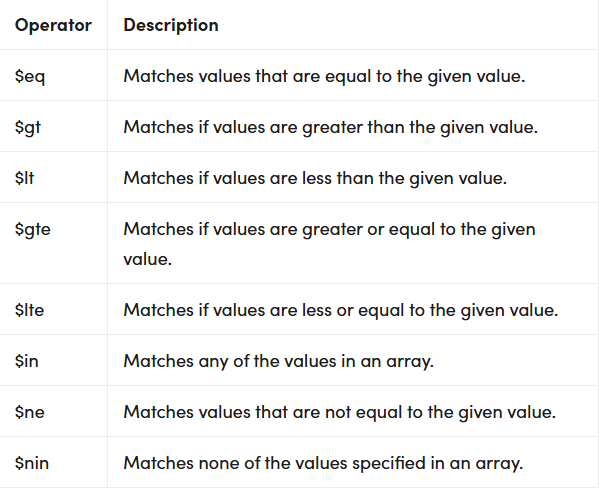
**MongoDB Operator**

What is Operator ?

Ans : Operator is a special keyword and symbol which is to perform operation like comparison , logical, bitwise etc.



Comparison operator : A **comparison operator** in programming is used to compare two values and determine their relationship, such as equality, inequality, or order.



db.inventory.find({"\_id": { $eq: "LS0009100"}}).pretty()

db.inventory.find({"quantity": { $gt: 5000}}).pretty()

db.inventory.find({"quantity": { $lt: 5000}}).pretty()

db.inventory.find({"quantity": { $gte: 12000}}).pretty()

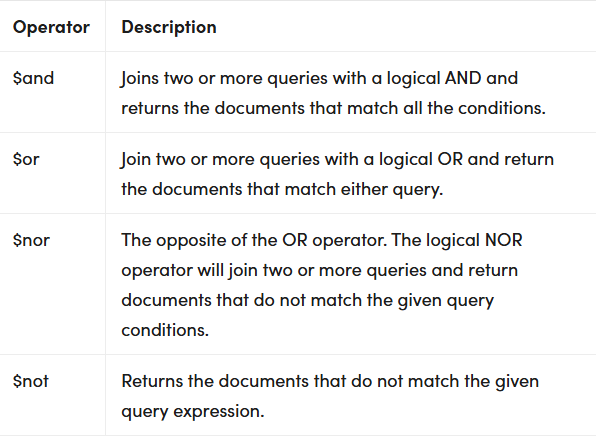
db.inventory.find({"quantity": { $lte: 1000}}).pretty()

db.inventory.find({"price": { $in: [3, 6]}}).pretty()

db.inventory.find({"price": { $nin: [5.23, 3, 6, 3.59, 4.95]}}).pretty()

db.inventory.find({"price": { $ne: 5.23}})

## Logical Operators : A ****logical operator**** is used to combine or modify Boolean expressions, returning a Boolean result. Common logical operators include:



db.employees.find({ $and: [{"job\_role": "Store Associate"}, {"emp\_age": {$gte: 20, $lte: 30}}]}).pretty()

db.employees.find({ $or: [{"job\_role": "Senior Cashier"}, {"job\_role": "Store Manager"}]}).pretty()

db.employees.find({ $nor: [{"job\_role": "Senior Cashier"}, {"job\_role": "Store Manager"}]}).pretty()

db.employees.find({ "emp\_age": { $not: { $gte: 40}}})

### MongoDB-তে $nor এবং $not অপারেটরের উদ্দেশ্য:

#### ১. $nor ****অপারেটর****

$nor অপারেটরটি **লজিক্যাল NOR** অপারেটর। এটি এমন ডকুমেন্টগুলো রিটার্ন করে যেগুলোতে দেওয়া কন্ডিশনগুলোর **কোনো একটিও সত্য নয়**। অর্থাৎ, যখন কন্ডিশনগুলো সব মিথ্যা হয়, তখন $nor ব্যবহার করে ডকুমেন্ট রিটার্ন করা যায়।

**উদাহরণ:**

db.employees.find({

$nor: [

{ emp\_age: { $lt: 30 } }, // emp\_age 30 এর চেয়ে ছোট নয়

{ emp\_name: "Rahul" } // emp\_name "Rahul" নয়

]

});

* এই কুয়েরি এমন ডকুমেন্ট রিটার্ন করবে যেখানে:
  + emp\_age 30 এর কম নয়, এবং
  + emp\_name "Rahul" নয়।

**সংক্ষেপে:** $nor অপারেটর তখন ব্যবহার করা হয় যখন একাধিক শর্ত থাকতে পারে, কিন্তু শর্তগুলোর কোনোটিই পূরণ না হওয়া ডকুমেন্ট চাই।

#### ২. $not ****অপারেটর****

$not অপারেটরটি একটি শর্তকে **উল্টো করে** (negate) দেয়। এটি নির্দিষ্ট একটি শর্তে **সত্য নয়** এমন ডকুমেন্টগুলো রিটার্ন করে।

**উদাহরণ:**

db.employees.find({

emp\_age: { $not: { $gt: 30 } } // emp\_age 30 এর বেশি নয়

});

* এই কুয়েরি এমন ডকুমেন্ট রিটার্ন করবে যেখানে emp\_age 30 এর বেশি নয়।

**সংক্ষেপে:** $not অপারেটর একটি নির্দিষ্ট শর্তের উল্টো রেজাল্ট দেয়।

### $nor ****এবং**** $not ****এর মধ্যে পার্থক্য:****

| ****পয়েন্ট**** | $nor | $not |
| --- | --- | --- |
| **সংজ্ঞা** | একাধিক শর্তের কোনোটিই সত্য নয় এমন ডকুমেন্ট রিটার্ন করে। | একটি নির্দিষ্ট শর্তের উল্টো মান রিটার্ন করে। |
| **কতগুলো শর্তে কাজ করে** | একাধিক শর্তে কাজ করে। | একটি শর্তে কাজ করে। |
| **ব্যবহার ক্ষেত্র** | লজিক্যাল NOR অপারেশন। | শর্তকে negate করতে। |
| **উদাহরণ** | $nor: [{cond1}, {cond2}] | field: { $not: { cond } } |

**উদাহরণ দিয়ে পার্থক্য:**

#### $nor উদাহরণ:

db.employees.find({

$nor: [

{ emp\_age: { $lt: 30 } }, // emp\_age 30 এর চেয়ে ছোট নয়

{ emp\_name: "Rahul" } // emp\_name "Rahul" নয়

]

});

* ডকুমেন্ট রিটার্ন হবে যেখানে emp\_age 30 এর চেয়ে ছোট নয় এবং emp\_name "Rahul" নয়।

#### $not উদাহরণ:

db.employees.find({

emp\_age: { $not: { $gt: 30 } } // emp\_age 30 এর বেশি নয়

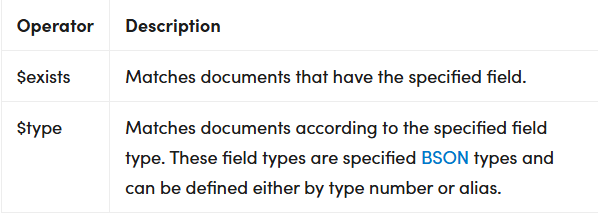
});

* ডকুমেন্ট রিটার্ন হবে যেখানে emp\_age 30 এর বেশি নয়।

**মোট কথা:**

* $nor একাধিক শর্তের উপর ভিত্তি করে কাজ করে, শর্তগুলোর সবগুলো মিথ্যা হতে হবে।
* $not একটি নির্দিষ্ট শর্তের বিপরীত রেজাল্ট দেয়।

## Element Operators : The element query operators are used to identify documents using the fields of the document. The table given below lists the current element .



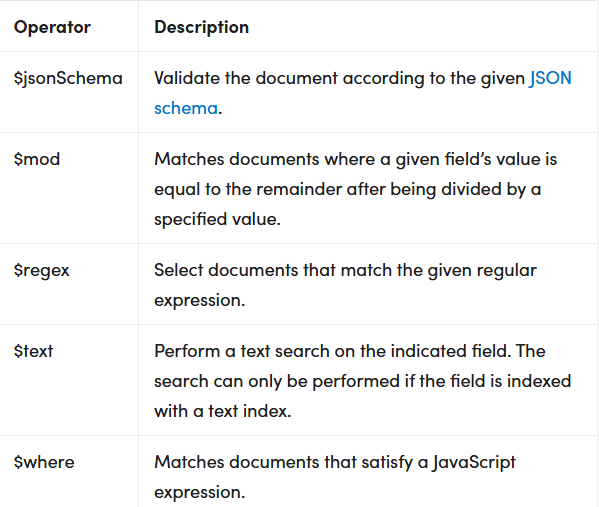
db.employees.find({ "emp\_age": { $exists: true, $gte: 30}}).pretty()

db.employees.find({ "address": { $exists: true}}).pretty()

db.employees.find({ "emp\_age": { $type: "bool"}})

db.employees.find({ "emp\_age": { $type: "double"}})

## Evaluation Operators :



### MongoDB-তে উল্লেখিত অপারেটরগুলোর ব্যাখ্যা:

#### ****1.**** $jsonSchema****:****

এই অপারেটরটি MongoDB-তে একটি ডকুমেন্ট যাচাই করার জন্য ব্যবহার করা হয়। এটি নির্ধারিত **JSON Schema** অনুযায়ী ডকুমেন্টের গঠন, ডেটা টাইপ, এবং নির্দিষ্ট নিয়মগুলি মেনে চলছে কিনা তা যাচাই করে।

**ব্যবহার:** ডেটা ভ্যালিডেশন এবং ডেটার নির্দিষ্ট স্ট্রাকচার নিশ্চিত করার জন্য।

**উদাহরণ:**

db.employees.insertOne({

$jsonSchema: {

bsonType: "object",

required: ["emp\_name", "emp\_age"],

properties: {

emp\_name: { bsonType: "string" },

emp\_age: { bsonType: "int", minimum: 18, maximum: 60 }

}

}

});

* এই উদাহরণে, emp\_name স্ট্রিং হতে হবে এবং emp\_age 18 থেকে 60-এর মধ্যে থাকতে হবে।

#### ****2.**** $mod****:****

এই অপারেটরটি এমন ডকুমেন্ট খুঁজে পেতে ব্যবহার করা হয় যেখানে কোনো একটি ফিল্ডের মান নির্দিষ্ট মান দ্বারা ভাগ করলে নির্দিষ্ট **remainder (ভাগশেষ)** মেলে।

**উদাহরণ:**

db.employees.find({ emp\_age: { $mod: [5, 0] } });

* এই কুয়েরি এমন ডকুমেন্ট রিটার্ন করবে যেখানে emp\_age 5 দিয়ে ভাগ করলে ভাগশেষ **0** থাকে (যেমন 25, 30, 35)।

#### ****3.**** $regex****:****

এই অপারেটরটি ব্যবহার করে নির্দিষ্ট **রেগুলার এক্সপ্রেশন** এর সাথে মিল আছে এমন ডকুমেন্ট নির্বাচন করা হয়।

**উদাহরণ:**

db.employees.find({ emp\_name: { $regex: "^A" } });

* এই কুয়েরি এমন ডকুমেন্ট রিটার্ন করবে যেখানে emp\_name অক্ষর **"A"** দিয়ে শুরু হয়।

**বিঃদ্রঃ:** $regex কেস সেনসিটিভ হয়, তবে কেস ইনসেনসিটিভ করতে /i ফ্ল্যাগ ব্যবহার করা যায়।

db.employees.find({ emp\_name: { $regex: "^A", $options: "i" } });

#### ****4.**** $text****:****

এই অপারেটরটি ডাটাবেসে **টেক্সট সার্চ** করার জন্য ব্যবহার করা হয়। এটি কেবলমাত্র তখনই কাজ করবে যখন সংশ্লিষ্ট ফিল্ডে **text index** তৈরি করা থাকে।

**উদাহরণ:**

db.employees.createIndex({ emp\_name: "text" });

db.employees.find({ $text: { $search: "developer" } });

* এই কুয়েরি emp\_name-এ "developer" শব্দটি খুঁজে বের করবে।

#### ****5.**** $where****:****

এই অপারেটরটি জটিল লজিক বা শর্ত যাচাইয়ের জন্য ব্যবহার করা হয়, যেখানে **JavaScript expression** ব্যবহার করে ডকুমেন্ট নির্বাচন করা যায়।

**উদাহরণ:**

db.employees.find({

$where: "this.emp\_age > 30 && this.emp\_salary > 50000"

});

* এই কুয়েরি এমন ডকুমেন্ট রিটার্ন করবে যেখানে emp\_age 30-এর বেশি এবং emp\_salary 50000-এর বেশি।

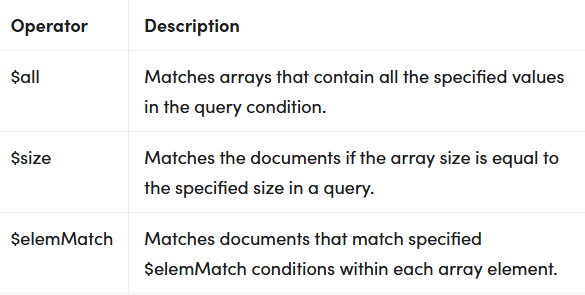
**বিঃদ্রঃ:** $where অপারেটর ব্যবহার করার সময় পারফরম্যান্স কম হতে পারে, তাই এটি সাবধানে ব্যবহার করা উচিত।

### ****সারাংশ টেবিল:****

| ****অপারেটর**** | ****উদ্দেশ্য**** |
| --- | --- |
| $jsonSchema | ডকুমেন্টের গঠন এবং ডেটার সঠিকতা যাচাই করে। |
| $mod | একটি ফিল্ডের মান নির্দিষ্ট মান দ্বারা ভাগ করলে নির্দিষ্ট ভাগশেষ মেলে। |
| $regex | রেগুলার এক্সপ্রেশনের সাথে মিল খুঁজে বের করে। |
| $text | টেক্সট ফিল্ডে দ্রুত অনুসন্ধান করতে ব্যবহার করা হয় (text index প্রয়োজন)। |
| $where | জটিল শর্ত যাচাই করার জন্য JavaScript expression ব্যবহার করে। |

**প্রয়োজন অনুসারে কোন অপারেটর ব্যবহার করবেন তার উপর নির্ভর করবে আপনার কুয়েরির ধরন এবং ডাটাবেস কাঠামো।**

## Array Operators : MongoDB array operators are designed to query documents with arrays. Here are the array operators provided by MongoDB.



### MongoDB-তে ****Array Operators**** নিয়ে বিস্তারিত ব্যাখ্যা:

MongoDB-তে অ্যারে ডেটার উপর কাজ করার জন্য কিছু নির্দিষ্ট অপারেটর ব্যবহার করা হয়। এই অপারেটরগুলি অ্যারে ফিল্ডের ভেতরের ডেটা খুঁজে বের করার জন্য খুবই কার্যকর। নিচে $all, $size, এবং $elemMatch অপারেটরের বিস্তারিত ব্যাখ্যা দেওয়া হলো।

#### ****1.**** $all ****অপারেটর:****

* $all অপারেটর এমন ডকুমেন্ট রিটার্ন করে যেখানে অ্যারে ফিল্ডটি একটি নির্দিষ্ট মানের সেটের সবগুলো মান ধারণ করে।
* সহজ কথায়, এটি নিশ্চিত করে যে অ্যারেটি কুয়েরিতে উল্লেখিত সব মান অন্তর্ভুক্ত করেছে।

**উদাহরণ:**  
ধরি, আমাদের ডেটাবেসে tags নামে একটি ফিল্ড আছে যেখানে অ্যারে ডেটা সংরক্ষিত:

{ "\_id": 1, "tags": ["mongodb", "database", "nosql"] }

{ "\_id": 2, "tags": ["javascript", "programming", "mongodb"] }

কুয়েরি:

db.collection.find({ tags: { $all: ["mongodb", "database"] } });

**আউটপুট:**  
এই কুয়েরি সেই ডকুমেন্ট রিটার্ন করবে যেখানে tags অ্যারেটি "mongodb" এবং "database" দুটো মানই ধারণ করে।  
আউটপুট হবে:

{ "\_id": 1, "tags": ["mongodb", "database", "nosql"] }

#### ****2.**** $size ****অপারেটর:****

* $size অপারেটর এমন ডকুমেন্ট রিটার্ন করে যেখানে একটি অ্যারের দৈর্ঘ্য (length) নির্দিষ্ট সংখ্যার সমান।
* এটি অ্যারেটির সাইজ চেক করার জন্য ব্যবহার করা হয়।

**উদাহরণ:**  
ধরি, আমাদের ডেটাবেসে scores নামে একটি ফিল্ড আছে যেখানে অ্যারে ডেটা সংরক্ষিত:

{ "\_id": 1, "scores": [95, 88, 92] }

{ "\_id": 2, "scores": [70, 85] }

কুয়েরি:

db.collection.find({ scores: { $size: 3 } });

**আউটপুট:**  
এই কুয়েরি সেই ডকুমেন্ট রিটার্ন করবে যেখানে scores অ্যারেটির দৈর্ঘ্য ৩।  
আউটপুট হবে:

{ "\_id": 1, "scores": [95, 88, 92] }

#### ****3.**** $elemMatch ****অপারেটর:****

* $elemMatch অপারেটর এমন ডকুমেন্ট রিটার্ন করে যেখানে অ্যারেতে থাকা এক বা একাধিক এলিমেন্ট নির্দিষ্ট শর্ত পূরণ করে।
* এটি অ্যারেতে থাকা অবজেক্ট বা জটিল ডেটা যাচাই করতে ব্যবহৃত হয়।

**উদাহরণ:**  
ধরি, আমাদের ডেটাবেসে products নামে একটি ফিল্ড আছে যেখানে অ্যারেতে অবজেক্ট ডেটা আছে:

{

"\_id": 1,

"products": [

{ "name": "laptop", "price": 1000 },

{ "name": "mouse", "price": 50 }

]

}

{

"\_id": 2,

"products": [

{ "name": "keyboard", "price": 150 },

{ "name": "monitor", "price": 300 }

]

}

কুয়েরি:

db.collection.find({

products: { $elemMatch: { price: { $gte: 100 }, name: "laptop" } }

});

**আউটপুট:**  
এই কুয়েরি সেই ডকুমেন্ট রিটার্ন করবে যেখানে products অ্যারেতে এমন একটি এলিমেন্ট আছে যার price ১০০ বা তার বেশি এবং name "laptop"।  
আউটপুট হবে:

{

"\_id": 1,

"products": [

{ "name": "laptop", "price": 1000 },

{ "name": "mouse", "price": 50 }

]

}

### ****সারাংশ টেবিল:****

| ****অপারেটর**** | ****উদ্দেশ্য**** | ****উদাহরণ**** | ****ব্যাখ্যা**** |
| --- | --- | --- | --- |
| $all | অ্যারেতে নির্দিষ্ট মানের সবগুলো উপস্থিতি যাচাই করা। | { tags: { $all: ["mongodb", "database"] } } | অ্যারেতে "mongodb" এবং "database" দুটো মানই থাকতে হবে। |
| $size | অ্যারের দৈর্ঘ্য নির্দিষ্ট মানের সমান কিনা যাচাই করা। | { scores: { $size: 3 } } | অ্যারেটির দৈর্ঘ্য ৩ হলে ডকুমেন্ট রিটার্ন হবে। |
| $elemMatch | অ্যারের এক বা একাধিক এলিমেন্ট একটি নির্দিষ্ট শর্ত পূরণ করছে কিনা যাচাই করা। | { products: { $elemMatch: { price: { $gte: 100 } } } } | অ্যারেতে এমন একটি এলিমেন্ট থাকতে হবে যার price ১০০ বা তার বেশি। |

**MongoDB-তে এই অপারেটরগুলি অ্যারে ফিল্ড নিয়ে কাজ করার জন্য খুবই শক্তিশালী এবং জটিল অনুসন্ধান সহজ করে।**

## Comment Operator : The MongoDB comment query operator associates a comment to any expression taking a query predicate. Adding comments to queries enables database administrators to trace and interpret MongoDB logs using the comments easily.

db.promo.find({ "period": { $eq: 7}, $comment: "Find Weeklong Promos"}).pretty()

### ****MongoDB-তে**** $comment ****অপারেটরের উদ্দেশ্য কী?****

MongoDB-তে $comment **অপারেটর** মূলত ডিবাগিং (debugging) এবং কুয়েরি বিশ্লেষণের (query analysis) জন্য ব্যবহৃত হয়। এটি ডেটাবেসে কোনো ডেটা ফিল্টার বা প্রভাবিত করে না। এর প্রধান কাজ হলো কুয়েরির সাথে অতিরিক্ত মন্তব্য (comments) যোগ করা, যা লজ ফাইল বা প্রোফাইলারে (profiler) দেখা যাবে।

### ****MongoDB**** $comment ****অপারেটরের মূল উদ্দেশ্য:****

1. **ডিবাগিং সহজ করা:**  
   $comment অপারেটর ব্যবহার করে আপনি কুয়েরির সাথে একটি বার্তা যোগ করতে পারেন, যা ডিবাগিং প্রক্রিয়ায় সহায়ক হতে পারে।
2. **কুয়েরি বিশ্লেষণ:**  
   প্রোফাইলার বা কুয়েরি লগ ফাইল পরীক্ষা করার সময় $comment আপনাকে নির্দিষ্ট কুয়েরি চিহ্নিত করতে সাহায্য করে।
3. **ট্র্যাকিং:**  
   অ্যাপ্লিকেশন থেকে চালিত কুয়েরিগুলোতে $comment যোগ করে, আপনি সহজে বুঝতে পারবেন কুয়েরিটি অ্যাপ্লিকেশনের কোন অংশ থেকে এসেছে।

### ****কিভাবে**** $comment ****কাজ করে?****

$comment অপারেটর একটি স্ট্রিং বার্তা যোগ করে এবং এই বার্তাটি কেবল **ডিবাগিং এবং লগিং টুলে** দৃশ্যমান হয়।

**উদাহরণ:**  
ধরি, আমাদের employees নামে একটি কালেকশন আছে:

db.employees.find(

{ department: "IT" },

{ $comment: "Finding all IT department employees" }

);

#### কী হবে?

* এই কুয়েরি "IT" বিভাগে থাকা ডকুমেন্ট রিটার্ন করবে।
* $comment অপারেটরের অংশটি কুয়েরির আউটপুটে প্রভাব ফেলবে না।
* আপনি প্রোফাইলার বা ডিবাগিং টুলে **"Finding all IT department employees"** বার্তাটি দেখতে পাবেন।

### ****MongoDB প্রোফাইলারে**** $comment ****কিভাবে দেখায়?****

প্রোফাইলার চালালে, $comment অপারেটরের মান কুয়েরির সাথে যুক্ত হয়ে প্রদর্শিত হয়:

{

"op": "query",

"ns": "test.employees",

"command": {

"find": "employees",

"filter": { "department": "IT" },

"$comment": "Finding all IT department employees"

}

}

### ****কেন**** $comment ****গুরুত্বপূর্ণ?****

* জটিল কুয়েরি বিশ্লেষণের সময় নির্দিষ্ট কুয়েরি চিহ্নিত করা সহজ হয়।
* অ্যাপ্লিকেশন থেকে চালিত কুয়েরির উৎস ট্র্যাক করতে সহায়ক।
* টিমে কাজ করার সময়, ডেভেলপাররা কুয়েরি ব্যাখ্যা বা উদ্দেশ্য সম্পর্কে দ্রুত ধারণা পেতে পারে।

### ****সারাংশ:****

* $comment **অপারেটর ডেটাবেসের কোনো ডেটা প্রভাবিত করে না।**
* এটি শুধুমাত্র **কুয়েরি ডিবাগিং** এবং **বিশ্লেষণ** এর জন্য ব্যবহৃত হয়।
* এটি কুয়েরির উদ্দেশ্য ব্যাখ্যা করতে এবং নির্দিষ্ট কুয়েরি সহজে চিহ্নিত করতে সাহায্য করে।

MongoDB-তে $comment মূলত ডেভেলপমেন্ট এবং ডিবাগিং প্রক্রিয়াকে আরও দক্ষ করে তুলতে ব্যবহৃত হয়।

Geospatial Operator :

### ****MongoDB-তে Geospatial Operator কী?****

MongoDB-তে **Geospatial Operators** হল এমন অপারেটর, যা ভৌগোলিক ডেটা (geospatial data) বা স্থানীয় তথ্য নিয়ে কাজ করতে ব্যবহৃত হয়। এই অপারেটরগুলো ভৌগোলিক স্থানাঙ্ক (longitude এবং latitude), দূরত্ব, এবং অঞ্চলভিত্তিক অনুসন্ধানের জন্য ব্যবহৃত হয়।

MongoDB-তে Geospatial অপারেশন করার জন্য ডেটা **2D স্পেস** বা **GeoJSON format**-এ সংরক্ষণ করতে হয় এবং ডেটাগুলোতে **geospatial index** তৈরি করতে হয়।

### ****Geospatial Operators-এর প্রধান কাজ:****

1. **স্থানের কাছাকাছি থাকা ডেটা খুঁজে পাওয়া।**
2. **কোনো নির্দিষ্ট দূরত্বের মধ্যে থাকা ডেটা রিটার্ন করা।**
3. **একটি নির্দিষ্ট অঞ্চলের ভেতর থাকা ডেটা নির্বাচন করা।**
4. **কোথাও অবস্থানের ভৌগোলিক সীমারেখা চেক করা।**

### ****MongoDB-তে গুরুত্বপূর্ণ Geospatial Operators:****

1. $near
2. $nearSphere
3. $geoWithin
4. $geoIntersects

### ****১.**** $near ****অপারেটর****

* $near অপারেটর একটি স্থানাঙ্কের কাছাকাছি থাকা পয়েন্টগুলো খুঁজে বের করে।
* এটি **2D plane** বা **GeoJSON point** ডেটার সাথে কাজ করে।

**উদাহরণ:**  
ধরি, আমাদের একটি কালেকশনে দোকানগুলোর লোকেশন সংরক্ষিত আছে:

db.stores.find({

location: {

$near: {

$geometry: { type: "Point", coordinates: [77.5946, 12.9716] }, // বেঙ্গালুরুর স্থানাঙ্ক

$maxDistance: 5000 // ৫ কিলোমিটারের মধ্যে

}

}

});

* এই কুয়েরি বেঙ্গালুরুর স্থানাঙ্কের (longitude, latitude) ৫ কিলোমিটারের মধ্যে থাকা দোকানগুলোর ডেটা রিটার্ন করবে।

### ****২.**** $nearSphere ****অপারেটর****

* $nearSphere অপারেটর গোলাকার পৃষ্ঠের (spherical surface) উপর ভিত্তি করে কাছাকাছি থাকা পয়েন্টগুলো খুঁজে বের করে।
* এটি মিটার বা রেডিয়ানে দূরত্ব পরিমাপ করে।

**উদাহরণ:**

db.stores.find({

location: {

$nearSphere: {

$geometry: { type: "Point", coordinates: [77.5946, 12.9716] },

$maxDistance: 5000

}

}

});

* এই কুয়েরি গ্লোবাল গোলাকার পৃষ্ঠের ভিত্তিতে কাছাকাছি দোকানগুলো খুঁজে বের করবে।

### ****৩.**** $geoWithin ****অপারেটর****

* $geoWithin অপারেটর কোনো নির্দিষ্ট ভৌগোলিক অঞ্চলের ভেতরে থাকা ডেটা খুঁজে বের করে।
* এটি **polygon**, **circle**, বা **box** এর মাধ্যমে অঞ্চলের সীমা নির্ধারণ করে।

**উদাহরণ:**

db.stores.find({

location: {

$geoWithin: {

$geometry: {

type: "Polygon",

coordinates: [

[

[77.0, 12.5],

[77.0, 13.5],

[78.0, 13.5],

[78.0, 12.5],

[77.0, 12.5]

]

]

}

}

}

});

* এই কুয়েরি নির্দিষ্ট পলিগনের মধ্যে থাকা দোকানগুলোর তথ্য রিটার্ন করবে।

### ****৪.**** $geoIntersects ****অপারেটর****

* $geoIntersects অপারেটর এমন ডকুমেন্ট খুঁজে বের করে যেগুলো একটি নির্দিষ্ট জ্যামিতিক আকারের সাথে ছেদ করে বা একত্রিত হয়।

**উদাহরণ:**

db.routes.find({

geometry: {

$geoIntersects: {

$geometry: {

type: "LineString",

coordinates: [[77.0, 12.5], [78.0, 13.5]]

}

}

}

});

* এই কুয়েরি এমন ডেটা রিটার্ন করবে, যেগুলো উল্লেখিত লাইনটির সাথে মিল বা ছেদ করে।

### ****Geospatial Index কী?****

Geospatial অপারেটর সঠিকভাবে কাজ করার জন্য সংশ্লিষ্ট ফিল্ডে একটি **geospatial index** থাকা প্রয়োজন।

**ইনডেক্স তৈরি করার উদাহরণ:**

db.stores.createIndex({ location: "2dsphere" });

* এখানে, location ফিল্ডে একটি 2D স্পিয়ার ইনডেক্স তৈরি করা হয়েছে।

### ****MongoDB Geospatial অপারেটরের উপকারিতা:****

1. **স্থানভিত্তিক অনুসন্ধান সহজ:** কাছাকাছি পয়েন্ট, নির্দিষ্ট দূরত্ব বা এলাকাভিত্তিক ডেটা খুঁজে বের করা সহজ হয়।
2. **রিয়েল-টাইম সিস্টেমে ব্যবহার:** ডেলিভারি অ্যাপ্লিকেশন বা ম্যাপিং সিস্টেমে ব্যবহার উপযোগী।
3. **দ্রুত ফলাফল প্রদান:** Geospatial index-এর কারণে অনুসন্ধান খুব দ্রুত হয়।

### ****সারাংশ:****

MongoDB-তে **Geospatial Operators** মূলত ভৌগোলিক স্থানাঙ্ক, দূরত্ব, এবং নির্দিষ্ট অঞ্চলের ডেটা খুঁজে বের করতে ব্যবহৃত হয়। $near, $nearSphere, $geoWithin, এবং $geoIntersects এই অপারেটরগুলো জটিল ভৌগোলিক অনুসন্ধান কার্যক্রম সহজ করে।

**MongoDB-এর Geospatial অপারেশন স্থানভিত্তিক অ্যাপ্লিকেশন এবং ম্যাপিং ডেটার জন্য অত্যন্ত কার্যকর।**

Bitwise operator :

### ****MongoDB-তে Bitwise Operator কী?****

MongoDB-তে **Bitwise Operators** ব্যবহার করা হয় ডেটাবেসের ডকুমেন্টের ফিল্ডের উপর বিট লেভেলের (bit-level) অপারেশন করতে। এগুলো মূলত বাইনারি ডেটা বা সংখ্যার মধ্যে বিট-ভিত্তিক তুলনা ও গণনার জন্য ব্যবহৃত হয়।

Bitwise অপারেটর ডেটার নির্দিষ্ট বিটগুলোর অবস্থার উপর ভিত্তি করে কাজ করে এবং ডকুমেন্ট ফিল্টার করতে সাহায্য করে।

### ****MongoDB-তে Bitwise Operators-এর তালিকা ও কাজ****

1. $bitsAllSet
   * **উদ্দেশ্য:** নির্দিষ্ট বিটগুলো 1 অবস্থায় রয়েছে এমন ডেটা খুঁজে বের করা।
   * **ব্যবহার:** ফিল্ডের নির্দিষ্ট বিটগুলোর মান চেক করতে।

**উদাহরণ:**

db.collection.find({ flags: { $bitsAllSet: [1, 3] } });

* + এখানে flags ফিল্ডের ১ ও ৩ নম্বর বিট 1 কিনা তা চেক করবে।

1. $bitsAnySet
   * **উদ্দেশ্য:** নির্দিষ্ট বিটগুলোর মধ্যে কোনো একটি 1 অবস্থায় রয়েছে এমন ডেটা খুঁজে বের করা।
   * **ব্যবহার:** অন্তত একটি নির্দিষ্ট বিট 1 কিনা তা যাচাই করতে।

**উদাহরণ:**

db.collection.find({ flags: { $bitsAnySet: [0, 2] } });

* + এই কুয়েরি চেক করবে flags ফিল্ডের ০ বা ২ নম্বর বিট 1 আছে কিনা।

1. $bitsAllClear
   * **উদ্দেশ্য:** নির্দিষ্ট বিটগুলো 0 অবস্থায় রয়েছে এমন ডেটা খুঁজে বের করা।
   * **ব্যবহার:** ফিল্ডের নির্দিষ্ট বিটগুলো 0 কিনা তা যাচাই করতে।

**উদাহরণ:**

db.collection.find({ flags: { $bitsAllClear: [1, 4] } });

* + এই কুয়েরি চেক করবে flags ফিল্ডের ১ ও ৪ নম্বর বিট 0 অবস্থায় আছে কিনা।

1. $bitsAnyClear
   * **উদ্দেশ্য:** নির্দিষ্ট বিটগুলোর মধ্যে অন্তত একটি 0 অবস্থায় রয়েছে এমন ডেটা খুঁজে বের করা।
   * **ব্যবহার:** কমপক্ষে একটি বিট 0 কিনা তা যাচাই করতে।

**উদাহরণ:**

db.collection.find({ flags: { $bitsAnyClear: [0, 2, 5] } });

* + এই কুয়েরি চেক করবে flags ফিল্ডের ০, ২, অথবা ৫ নম্বর বিট 0 আছে কিনা।

### ****Bitwise Operators-এর ব্যবহার ক্ষেত্র****

1. **সিস্টেম স্টেট ট্র্যাক করা:**  
   উদাহরণস্বরূপ, যদি কোনো ডিভাইসের বিভিন্ন স্টেট (অন, অফ, এরর) flags হিসেবে সংরক্ষিত থাকে, Bitwise অপারেটরের মাধ্যমে নির্দিষ্ট স্টেট চেক করা যায়।
2. **একাধিক অবস্থা একসাথে যাচাই করা:**  
   যেমন: ব্যবহারকারীর পারমিশন (read, write, execute) চেক করা।
3. **বাইনারি ডেটা নিয়ে কাজ:**  
   বড় ডেটা বা ফাইল স্ট্রিম বিশ্লেষণের জন্য উপযোগী।

### ****MongoDB Bitwise Operators কেন কার্যকর?****

1. **দ্রুত ফিল্টারিং:** বড় সংখ্যক ডেটার নির্দিষ্ট বিটভিত্তিক শর্ত চেক করা সহজ ও দ্রুত।
2. **মেমোরি-সাশ্রয়ী:** স্টেট এবং স্ট্যাটাস সংরক্ষণের জন্য ফ্ল্যাগ ব্যবহার করে ডেটা সঞ্চয় সহজ হয়।
3. **স্টেট-ভিত্তিক সিস্টেমের জন্য অপরিহার্য:** ডিভাইস, ব্যবহারকারীর অ্যাক্সেস, এবং সিস্টেম লগ চেক করার ক্ষেত্রে কার্যকর।

### ****সারসংক্ষেপ****

MongoDB-তে Bitwise Operators ফিল্ডের বিট লেভেলের স্টেট চেক করে ডেটা ফিল্টার করতে ব্যবহৃত হয়। $bitsAllSet, $bitsAnySet, $bitsAllClear, এবং $bitsAnyClear এই অপারেটরগুলো সুনির্দিষ্ট শর্তে ডেটা খুঁজে বের করতে কার্যকর।  
**এই অপারেটরগুলো সাধারণত সিস্টেম স্টেট বা ফ্ল্যাগ-ভিত্তিক ডেটাবেস ম্যানেজমেন্টে বেশি ব্যবহৃত হয়।**

**MongoDB Aggregation**

Aggregation is the process of selecting data from a collection in MongoDB. It processes multiple documents and returns computed results.

**MongoDB Aggregation হলো একটি ডেটা প্রসেসিং ফ্রেমওয়ার্ক যা ডেটাকে বিভিন্ন ধাপে ধাপে প্রসেস করে এবং কাঙ্ক্ষিত আউটপুট প্রদান করে। এটি মূলত ডেটাকে ফিল্টার করা, গ্রুপ করা, সাজানো, গণনা করা এবং বিভিন্ন ধরণের ট্রান্সফরমেশন করার জন্য ব্যবহৃত হয়। Aggregation Framework ব্যবহার করে আমরা ডেটার উপর জটিল অপারেশন করতে পারি যা সাধারণ কুয়েরি দিয়ে করা সম্ভব নয়।**

### Aggregation কাজ করার পদ্ধতি:

MongoDB তে Aggregation পাইপলাইনের উপর ভিত্তি করে কাজ করে। পাইপলাইনে একাধিক **স্টেজ** (stage) থাকে, এবং প্রতিটি স্টেজ ডেটার উপর একটি নির্দিষ্ট অপারেশন চালায়। এক স্টেজের আউটপুট পরের স্টেজের ইনপুট হিসেবে ব্যবহৃত হয়।

**MongoDB Aggregation** হলো একটি ডেটা প্রসেসিং ফ্রেমওয়ার্ক যা ডেটাকে বিভিন্ন ধাপে ধাপে প্রসেস করে এবং কাঙ্ক্ষিত আউটপুট প্রদান করে। এটি মূলত ডেটাকে ফিল্টার করা, গ্রুপ করা, সাজানো, গণনা করা এবং বিভিন্ন ধরণের ট্রান্সফরমেশন করার জন্য ব্যবহৃত হয়। Aggregation Framework ব্যবহার করে আমরা ডেটার উপর জটিল অপারেশন করতে পারি যা সাধারণ কুয়েরি দিয়ে করা সম্ভব নয়।

### Aggregation কাজ করার পদ্ধতি:

MongoDB তে Aggregation পাইপলাইনের উপর ভিত্তি করে কাজ করে। পাইপলাইনে একাধিক **স্টেজ** (stage) থাকে, এবং প্রতিটি স্টেজ ডেটার উপর একটি নির্দিষ্ট অপারেশন চালায়। এক স্টেজের আউটপুট পরের স্টেজের ইনপুট হিসেবে ব্যবহৃত হয়।

### Aggregation এর কিছু সাধারণ স্টেজ:

1. **$match**: ডেটাকে ফিল্টার করার জন্য ব্যবহৃত হয় (WHERE ক্লজের মত)।
2. **$group**: ডেটাকে গ্রুপ করে গণনা করতে বা অন্য কোনো অপারেশন করতে ব্যবহৃত হয় (SQL-এর GROUP BY এর মত)।
3. **$project**: নির্দিষ্ট ফিল্ড নির্বাচন বা নতুন ফিল্ড তৈরির জন্য ব্যবহৃত হয়।
4. **$sort**: ডেটা সাজানোর জন্য ব্যবহৃত হয় (ASCENDING বা DESCENDING)।
5. **$limit**: নির্দিষ্ট সংখ্যক ডেটা প্রদর্শনের জন্য ব্যবহৃত হয়।
6. **$skip**: ডেটার কিছু অংশ বাদ দেওয়ার জন্য ব্যবহৃত হয়।
7. **$lookup**: অন্য কালেকশনের সাথে জয়েন করার জন্য ব্যবহৃত হয়।

### Aggregation Framework এর সুবিধা:

1. বড় ডেটা সেটের উপর জটিল অপারেশন করা সহজ।
2. SQL-এর মতো ডেটা গ্রুপিং ও ফিল্টারিং সমর্থন করে।
3. ডেটাকে আরও অর্থবহ ফর্ম্যাটে রূপান্তর করার ক্ষমতা দেয়।
4. **$lookup** স্টেজ ব্যবহার করে রিলেশনাল ডেটাবেসের মতো জয়েন অপারেশন করা যায়।

এটি ডেটা অ্যানালাইসিস ও রিপোর্টিংয়ের জন্য খুবই উপযোগী।

### ****MongoDB Aggregation স্টেজগুলোর উদাহরণ সহ ব্যাখ্যা (বাংলায়)****

#### ****1. $match****

**ব্যবহার**: ডেটাকে নির্দিষ্ট ক্রাইটেরিয়া অনুযায়ী ফিল্টার করতে ব্যবহৃত হয় (যেমন SQL-এর WHERE ক্লজ)।

**উদাহরণ:**  
orders কালেকশনে যদি এমন ডেটা থাকে যেখানে status: "shipped" শুধুমাত্র সেই ডেটাগুলো ফিল্টার করতে চাই:

db.orders.aggregate([

{

$match: { status: "shipped" }

}

])

**ফলাফল:**

[

{ "orderId": 1, "status": "shipped", "item": "book", "quantity": 10 },

{ "orderId": 3, "status": "shipped", "item": "pen", "quantity": 5 }

]

#### ****2. $group****

**MongoDB-তে** group **অপারেশন ডেটা একত্রিত করার জন্য ব্যবহৃত হয়। এটি সাধারণত aggregation framework-এর মধ্যে ব্যবহৃত হয়।** group **স্টেজ ব্যবহার করে, আমরা ডেটাকে গ্রুপ করে নির্দিষ্ট ফিল্ডের উপর ভিত্তি করে গণনা (যেমন sum, avg, max, min) করতে পারি।**

**ব্যবহার**: ডেটাকে গ্রুপ করে সংক্ষিপ্ত ফলাফল তৈরি করা হয় (যেমন SQL-এর GROUP BY)।

**উদাহরণ:**  
orders কালেকশন থেকে প্রতিটি আইটেমের মোট বিক্রয় গণনা করতে:

db.orders.aggregate([

{

$group: {

\_id: "$item",

totalQuantity: { $sum: "$quantity" }

}

}

])

**ফলাফল:**

[

{ "\_id": "book", "totalQuantity": 15 },

{ "\_id": "pen", "totalQuantity": 10 }

]

#### ****3. $project****

**ব্যবহার**: নির্দিষ্ট ফিল্ড নির্বাচন করা বা নতুন ফিল্ড তৈরি করা (যেমন SELECT statement)।

**উদাহরণ:**  
প্রতিটি অর্ডারের item এবং তার মোট মূল্য বের করতে চাই:

db.orders.aggregate([

{

$project: {

item: 1,

totalPrice: { $multiply: ["$price", "$quantity"] }

}

}

])

**ফলাফল:**

[

{ "item": "book", "totalPrice": 200 },

{ "item": "pen", "totalPrice": 50 }

]

#### ****4. $sort****

**ব্যবহার**: ডেটাকে নির্দিষ্ট ক্রমে সাজানো (ASCENDING বা DESCENDING)।

**উদাহরণ:**  
orders কালেকশন থেকে price এর উপর ভিত্তি করে ডেটা সাজাতে:

db.orders.aggregate([

{

$sort: { price: -1 } // DESCENDING

}

])

**ফলাফল:**

[

{ "item": "book", "price": 100 },

{ "item": "notebook", "price": 50 },

{ "item": "pen", "price": 10 }

]

#### ****5. $limit****

**ব্যবহার**: নির্দিষ্ট সংখ্যক ডেটা প্রদর্শন করতে।

**উদাহরণ:**  
শুধুমাত্র প্রথম ২টি ডেটা দেখতে চাই:

db.orders.aggregate([

{

$limit: 2

}

])

**ফলাফল:**

[

{ "item": "book", "price": 100, "quantity": 2 },

{ "item": "pen", "price": 10, "quantity": 5 }

]

#### ****6. $skip****

**ব্যবহার**: শুরু থেকে নির্দিষ্ট সংখ্যক ডেটা বাদ দিতে।

**উদাহরণ:**  
প্রথম ২টি ডেটা বাদ দিয়ে পরবর্তী ডেটা দেখতে চাই:

db.orders.aggregate([

{

$skip: 2

}

])

**ফলাফল:**

[

{ "item": "notebook", "price": 50, "quantity": 3 }

]

#### ****7. $lookup****

**ব্যবহার**: অন্য একটি কালেকশনের সাথে জয়েন করার জন্য।

**উদাহরণ:**  
ধরা যাক orders এবং customers নামে দুটি কালেকশন আছে। আমরা orders এর সাথে customers জয়েন করে প্রতিটি অর্ডারের গ্রাহকের নাম দেখতে চাই:

orders **কালেকশন:**

[

{ "orderId": 1, "customerId": 101, "item": "book" },

{ "orderId": 2, "customerId": 102, "item": "pen" }

]

customers **কালেকশন:**

[

{ "customerId": 101, "name": "John Doe" },

{ "customerId": 102, "name": "Jane Smith" }

]

**Aggregation Query:**

db.orders.aggregate([

{

$lookup: {

from: "customers", // অন্য কালেকশন

localField: "customerId", // orders কালেকশনের ফিল্ড

foreignField: "customerId", // customers কালেকশনের ফিল্ড

as: "customerDetails" // আউটপুট ফিল্ড

}

}

])

**ফলাফল:**

[

{

"orderId": 1,

"customerId": 101,

"item": "book",

"customerDetails": [

{ "customerId": 101, "name": "John Doe" }

]

},

{

"orderId": 2,

"customerId": 102,

"item": "pen",

"customerDetails": [

{ "customerId": 102, "name": "Jane Smith" }

]

}

]

### ****Main Aggregation logic****

**নীচে MongoDB-এর Aggregation Pipeline-এর ১৪টি স্টেজের উদাহরণ বাংলা ব্যাখ্যাসহ কোড এবং প্রত্যাশিত আউটপুটসহ দেওয়া হলো।**

### 1. ****$addFields / $set****

নতুন ফিল্ড যোগ করতে ব্যবহৃত হয়। $set $addFields-এর সমার্থক।

#### ****কোড উদাহরণ****

db.products.insertMany([

{ name: "Laptop", price: 500, quantity: 2 },

{ name: "Phone", price: 300, quantity: 5 }

]);

db.products.aggregate([

{ $addFields: { totalPrice: { $multiply: ["$price", "$quantity"] } } }

]);

#### ****আউটপুট****

[

{ "name": "Laptop", "price": 500, "quantity": 2, "totalPrice": 1000 },

{ "name": "Phone", "price": 300, "quantity": 5, "totalPrice": 1500 }

]

### 2. ****$bucket****

ডকুমেন্টগুলোকে নির্দিষ্ট বাউন্ডারির ভিত্তিতে গ্রুপ করে।

#### ****কোড উদাহরণ****

db.sales.insertMany([

{ product: "A", amount: 150 },

{ product: "B", amount: 80 },

{ product: "C", amount: 200 },

{ product: "D", amount: 300 }

]);

db.sales.aggregate([

{ $bucket: {

groupBy: "$amount",

boundaries: [0, 100, 200, 300],

default: "Other",

output: { count: { $sum: 1 } }

}}

]);

#### ****আউটপুট****

[

{ "\_id": 0, "count": 1 },

{ "\_id": 100, "count": 1 },

{ "\_id": 200, "count": 1 },

{ "\_id": "Other", "count": 1 }

]

### 3. ****$bucketAuto****

বাউন্ডারি স্বয়ংক্রিয়ভাবে তৈরি হয় এবং ডকুমেন্টগুলো নির্দিষ্ট সংখ্যক গ্রুপে ভাগ হয়।

#### ****কোড উদাহরণ****

db.sales.aggregate([

{ $bucketAuto: { groupBy: "$amount", buckets: 2 } }

]);

#### ****আউটপুট****

[

{ "\_id": { "min": 80, "max": 200 }, "count": 2 },

{ "\_id": { "min": 200, "max": 300 }, "count": 2 }

]

### 4. ****$changeStream****

ডেটার পরিবর্তন পর্যবেক্ষণ করতে ব্যবহৃত হয়।  
**নোট:** এটি শুধুমাত্র বিশেষ পরিস্থিতিতে ব্যবহার করা হয় এবং এখানে ডেমো সম্ভব নয়।

### 5. ****$count****

ডকুমেন্টের সংখ্যা গণনা করতে ব্যবহৃত হয়।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $match: { price: { $gte: 300 } } },

{ $count: "expensiveProducts" }

]);

#### ****আউটপুট****

[

{ "expensiveProducts": 2 }

]

### 6. ****$lookup****

দুটি কালেকশনের মধ্যে যোগ তৈরি করতে ব্যবহৃত হয়।

#### ****কোড উদাহরণ****

db.orders.insertMany([

{ orderId: 1, customerId: 101 },

{ orderId: 2, customerId: 102 }

]);

db.customers.insertMany([

{ customerId: 101, name: "Alice" },

{ customerId: 102, name: "Bob" }

]);

db.orders.aggregate([

{ $lookup: {

from: "customers",

localField: "customerId",

foreignField: "customerId",

as: "customerInfo"

}}

]);

#### ****আউটপুট****

[

{ "orderId": 1, "customerId": 101, "customerInfo": [ { "customerId": 101, "name": "Alice" } ] },

{ "orderId": 2, "customerId": 102, "customerInfo": [ { "customerId": 102, "name": "Bob" } ] }

]

### 7. ****$match****

ডেটা ফিল্টার করার জন্য ব্যবহৃত হয়।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $match: { price: { $gte: 300 } } }

]);

#### ****আউটপুট****

[

{ "name": "Laptop", "price": 500, "quantity": 2 },

{ "name": "Phone", "price": 300, "quantity": 5 }

]

### 8. ****$group****

ডকুমেন্টগুলোকে গ্রুপ করে এবং অপারেশন প্রয়োগ করে।

#### ****কোড উদাহরণ****

db.sales.aggregate([

{ $group: { \_id: "$product", totalAmount: { $sum: "$amount" } } }

]);

#### ****আউটপুট****

[

{ "\_id": "A", "totalAmount": 150 },

{ "\_id": "B", "totalAmount": 80 },

{ "\_id": "C", "totalAmount": 200 },

{ "\_id": "D", "totalAmount": 300 }

]

### 9. ****$project****

নির্দিষ্ট ফিল্ড নির্বাচন করতে ব্যবহৃত হয়।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $project: { name: 1, price: 1, \_id: 0 } }

]);

#### ****আউটপুট****

[

{ "name": "Laptop", "price": 500 },

{ "name": "Phone", "price": 300 }

]

### 10. ****$sort****

ডেটাকে নির্দিষ্ট ক্রমে সাজাতে ব্যবহৃত হয়।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $sort: { price: -1 } }

]);

#### ****আউটপুট****

[

{ "name": "Laptop", "price": 500, "quantity": 2 },

{ "name": "Phone", "price": 300, "quantity": 5 }

]

### 11. ****$unwind****

অ্যারে ফিল্ডকে ডিকনস্ট্রাক্ট করে।

#### ****কোড উদাহরণ****

db.orders.insertMany([

{ orderId: 1, items: ["Laptop", "Mouse"] }

]);

db.orders.aggregate([

{ $unwind: "$items" }

]);

#### ****আউটপুট****

[

{ "orderId": 1, "items": "Laptop" },

{ "orderId": 1, "items": "Mouse" }

]

### 12. ****$limit****

ডেটার প্রথম নির্দিষ্ট সংখ্যক ডকুমেন্ট রিটার্ন করে।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $limit: 1 }

]);

#### ****আউটপুট****

[

{ "name": "Laptop", "price": 500, "quantity": 2 }

]

### 13. ****$skip****

প্রথম কয়েকটি ডকুমেন্ট বাদ দেয়।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $skip: 1 }

]);

#### ****আউটপুট****

[

{ "name": "Phone", "price": 300, "quantity": 5 }

]

### 14. ****$sample****

এলোমেলোভাবে ডকুমেন্ট সিলেক্ট করে।

#### ****কোড উদাহরণ****

db.products.aggregate([

{ $sample: { size: 1 } }

]);

#### ****আউটপুট (উদাহরণস্বরূপ)****

[

{ "name": "Phone", "price": 300, "quantity": 5 }

]

**প্রতিটি উদাহরণ আপনার MongoDB ডাটাবেসে চেষ্টা করুন। এগুলো ডেটা বিশ্লেষণের জন্য খুবই কার্যকর।**

**Node With TypeScript**

create npm init -y and install dependency which you install generally start of every nodeJS project

1. Create main file which was app.js or index.js that will be index.ts
2. npm install typescript ts-node @types/node @types/express --save-dev Install this for typescript, connect node with typescript and use express with typescript
3. npx tsc --init After doing this there will be create a file tsconfig.json. Replace tsconfig.json file material with another code which I use earlier
4. Create a folder dist in your project root directory where will be keep our code which is need for deplyment build. after npm run build data will be added in dist folder
5. In TypeScript there will not be use old import export technique there will be use es6 import export technique
6. Make nodemon start area app.js to app.ts
7. When you install any dependency in typescript use @types/ before depencency name npm i @types/cors --save-dev
8. I have to create a folder in app folder name will be interface there will be create interface for all model for perfect validation and use it in model file
9. You have to use Promise for to handle async operation function in typescript
10. You have to install two depency normal js dependency and devdependency for typescript suppose I use jsonwebtoken then I have to use #npm i jsonwebtoken and #npm i @types/jsonwebtoken --save-dev

For to see full project : <https://github.com/mousomray/nodeTS_crud_auth_api_ejs>