**🛠️ MongoDB Atlas Setup (Step-by-Step Full Guide)**

**1️⃣ Go to MongoDB Website**

**✅ Actions:**

* Open <https://www.mongodb.com>
* Click on **“Try Atlas Free”**
* Sign up using your **email or Google account**

**2️⃣ Choose the Free Forever Plan**

**🆓 M0 Free Tier:**

* Perfect for practice & small projects
* No credit card needed

**3️⃣ Create a Cluster (Your Cloud Database)**

**👨‍🔧 What You’ll Be Asked:**

* **Cluster Name** → e.g., myProjectDB
* **Security Setup** → Click "Auto-Configure" to whitelist your current IP
* **Sample Data** → Optional (if you want to play with sample collections)

**4️⃣ Select Cloud Provider & Region**

| **Option** | **Recommendation** |
| --- | --- |
| 🌩 Cloud Provider | AWS / Azure / Google Cloud — any OK |
| 🌍 Region | India users: Select **Mumbai** for better speed |

**5️⃣ Deploy Your Cluster**

**⏳ It may take 1–2 minutes.**

Once done, you’ll get:

* 👤 A **username**
* 🔐 A **password** (set by you)
* 🔗 A **connection string** like:

bash

CopyEdit

mongodb+srv://<username>:<password>@cluster0.mongodb.net/?retryWrites=true&w=majority

📌 **Copy & save this** → You’ll need it for connecting from your app

**🔌 Connect Node.js App to MongoDB Atlas**

**✅ Step 1: Choose “Connect with Application”**

1. Click on **Connect**
2. Choose 👉 **Connect your application**
3. Copy the generated **connection string**

**✅ Step 2: Customize Your Connection String**

Example:

bash

CopyEdit

mongodb+srv://likan:123456@cluster0.mongodb.net/myDatabaseName?retryWrites=true&w=majority

**Replace:**

* <username> → your MongoDB username
* <password> → your password
* myDatabaseName → any DB name (it will be created automatically when you add data)

**✅ Step 3: Create database.js in Your Project**

**1. Go to your Node.js project folder**

**2. Create a file called: database.js**

**3. Paste the following code:**

js

CopyEdit

// database.js

const mongoose = require('mongoose');

// 🔗 Replace with your actual MongoDB URI

const uri = 'mongodb+srv://likan:123456@cluster0.mongodb.net/myDatabaseName?retryWrites=true&w=majority';

// 🚀 Connect to MongoDB Atlas

mongoose.connect(uri, {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log('✅ MongoDB connected successfully'))

.catch((err) => console.error('❌ MongoDB connection error:', err));

**✅ Step 4: Install mongoose if not done**

bash

CopyEdit

npm install mongoose

🧠 mongoose is an ODM (Object Document Mapper) for MongoDB that helps structure and validate your MongoDB data inside Node.js

**🧠 Bonus Tips**

**🔒 Security Reminder:**

* Never hardcode your MongoDB URI in production!
* Use environment variables or .env files.

env

CopyEdit

# .env

MONGO\_URI="mongodb+srv://likan:123456@cluster0.mongodb.net/myDatabaseName"

And in your code:

js

CopyEdit

require('dotenv').config();

mongoose.connect(process.env.MONGO\_URI, { ... });

**🌐 Can You Use This in Real Apps?**

✅ YES! MongoDB Atlas is perfect for both **practice** and **production** apps. You can easily scale your cluster later when needed.

**📁 Folder Structure Sample:**

pgsql

CopyEdit

your-project/

│

├── database.js ← MongoDB connection logic

├── app.js ← Main Node/Express server

├── models/ ← Mongoose models

├── routes/ ← Route handlers

└── .env ← For secret keys & DB strings

# 🧭 Using ****MongoDB Compass**** (Step-by-Step Guide)

MongoDB Compass is the official **GUI tool** for MongoDB that helps you **visualize, create, read, update, and delete data** easily — no need to write code for everything!

## ✅ 1. Download MongoDB Compass

🔗 [**Download MongoDB Compass**](https://www.mongodb.com/products/tools/compass)

* Choose version based on your OS (Windows, macOS, Linux)
* Install it like any regular software

**✅ 2. Connect Compass to Your Atlas Cluster**

**Open Compass → Click "New Connection"**

* Paste your MongoDB **Connection String** (from Atlas)

perl

CopyEdit

mongodb+srv://<username>:<password>@cluster0.mongodb.net/myDatabaseName

📌 Replace:

* <username> → your MongoDB username
* <password> → your password
* myDatabaseName → optional database to connect

✅ Then click **Connect**

**✅ 3. Create a New Database (Visually)**

Once connected:

* Click **“Create Database”**
* Fill in:
  + **Database Name** → e.g., ecommerceDB
  + **Collection Name** → e.g., products
* Click **"Create Database"**

**🧠 Notes:**

* A **collection** is like a **table** in SQL
* You must give a starting collection name (can add more later)
* ❌ Don't enable "Time-Series Collection" unless you’re storing time-based sensor data

## ✅ 4. Insert Documents (Data)

To add data to your collection:

1. Select your collection (e.g., users)
2. Click **"Add Data" → "Insert Document"**
3. You’ll see a JSON editor:

json

CopyEdit

{

"name": "Likan",

"email": "likan@example.com",

"role": "admin"

}

1. Click **"Insert"**

### 🧠 Notes:

* MongoDB auto-generates a unique \_id for every document
* You can also edit data inline or delete documents from GUI

## 5. View, Filter & Explore Data

* You can **search/filter** data using MongoDB query syntax in the search bar

{ "role": "admin" }

* Use the tabs to:
  + View documents
  + See schema overview
  + Run aggregations (for advanced queries)
  + View indexes and performance insights

## 6. Real-Time Changes

* Any changes made via code (Node.js, Mongoose) are visible immediately in Compass.
* Any insertions in Compass are reflected in your code as well — it's synced

## 💡 Why Use MongoDB Compass?

| **Benefit** | **Description** |
| --- | --- |
| 🖥️ GUI Interface | No need to write CLI queries for basic operations |
| 🔍 Schema Visualizer | Automatically analyzes data structure |
| 🧪 Aggregation Playground | Run complex queries visually |
| 📈 Performance Stats | See index usage and query efficiency |
| 🔄 Real-Time Sync | Works with Atlas or local MongoDB |

## ⚠️ Troubleshooting Tip:

If connection fails:

* Make sure your **IP is whitelisted** in MongoDB Atlas → Go to **Network Access → Add IP Address**
* Ensure you're using the correct **username & password**

**💻 MongoDB CRUD with Node.js (Without Mongoose)**

**✅ 1. Install the MongoDB Driver (Official)**

bash

CopyEdit

npm install mongodb

This will install the official mongodb package needed to connect and interact with your MongoDB Atlas cluster.

**❓ What is NPM?**

**NPM (Node Package Manager)** is like an app store for Node.js. It helps you:

* Download & install **libraries**
* Manage project **dependencies**
* Keep code **modular and reusable**

So when you run npm install mongodb, it fetches the official MongoDB driver for Node.js from the npm registry.

**📁 2. database.js — Full Code Example (Insert + Read)**

js

CopyEdit

// database.js

// ✅ 1. Import MongoClient from mongodb

const { MongoClient } = require("mongodb");

// ✅ 2. Replace this with your MongoDB URI

const url = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

// ✅ 3. Define the database name

const dbName = "HelloWorld";

// ✅ 4. Create a new client instance

const client = new MongoClient(url);

async function main() {

// ✅ 5. Connect to the cluster

await client.connect();

console.log("✅ Connected successfully to MongoDB Atlas");

// ✅ 6. Select the database & collection

const db = client.db(dbName);

const collection = db.collection("User");

// ✅ 7. Insert one document

const data = {

firstname: "Ranveer",

lastname: "Singh",

city: "Mumbai",

phoneNumber: "987543210"

};

const insertResult = await collection.insertOne(data);

console.log("📥 Inserted document:", insertResult.insertedId);

// ✅ 8. Read all documents

const findResult = await collection.find({}).toArray();

console.log("🔍 All documents:", findResult);

// ✅ 9. Count total documents

const countResult = await collection.countDocuments({});

console.log("📊 Total documents in User collection:", countResult);

// ✅ 10. Find documents matching a filter

const filteredCount = await collection.countDocuments({ firstname: "Deepika" });

console.log("📁 Documents with firstname 'Deepika':", filteredCount);

return "✅ Done with operations.";

}

// ✅ 11. Run the main function

main()

.then(console.log)

.catch(console.error)

.finally(() => client.close()); // ✅ 12. Always close the client

**✅ Full Code with *Line-by-Line Explanation***

js

CopyEdit

// ✅ 1. Import MongoClient from the official mongodb package

const { MongoClient } = require("mongodb");

🔸 require("mongodb"): loads the MongoDB driver  
🔸 { MongoClient }: we’re pulling out just MongoClient from the driver  
👉 Think of MongoClient as the tool that lets you connect to MongoDB

js

CopyEdit

// ✅ 2. This is your MongoDB Atlas connection string (replace with your own)

const url = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

🔸 This url is how your app connects to your cloud database  
🔸 It includes:

* likan: your username
* 123456: your password (replace with real one)
* cluster0.mongodb.net: your cluster's address  
  🔒 **Never expose this string publicly**

js

CopyEdit

// ✅ 3. This is the name of the database you'll use

const dbName = "HelloWorld";

🧠 MongoDB will **automatically create** this database if it doesn’t already exist when you insert data.

js

CopyEdit

// ✅ 4. This creates a new client object using your connection URL

const client = new MongoClient(url);

🔧 This is like preparing a database connection  
You haven’t connected *yet* — you’re just setting it up

js

CopyEdit

// ✅ 5. Our async main() function — we will do all the database work inside it

async function main() {

🔹 Marked as async so we can use await inside — makes code easier to read  
(You can’t use await without async)

js

CopyEdit

// ✅ 6. Connect to MongoDB Atlas cluster

await client.connect();

console.log("✅ Connected successfully to MongoDB Atlas");

🔸 await client.connect() = actually opens the connection  
🧠 Without await, it would return a promise and your code would run before the connection finishes

js

CopyEdit

// ✅ 7. Get a reference to the database

const db = client.db(dbName);

🔹 client.db("HelloWorld") = selects the database named HelloWorld  
If it doesn’t exist yet, MongoDB will create it when needed

js

CopyEdit

// ✅ 8. Access (or create) a collection named "User"

const collection = db.collection("User");

🗂️ A **collection** is like a table in SQL  
This says “Hey Mongo, I want to use a collection called User”

js

CopyEdit

// ✅ 9. Create the document (a JS object) we want to insert

const data = {

firstname: "Ranveer",

lastname: "Singh",

city: "Mumbai",

phoneNumber: "987543210"

};

🧾 This is your **document** — a plain JavaScript object you’ll insert into the DB  
Every field here becomes a field in the database

js

CopyEdit

// ✅ 10. Insert the document into the User collection

const insertResult = await collection.insertOne(data);

console.log("📥 Inserted document:", insertResult.insertedId);

🧠 insertOne() is a method that inserts 1 document  
🔹 insertResult.insertedId gives back the unique ID MongoDB assigned to the document

js

CopyEdit

// ✅ 11. Find (read) all documents in the User collection

const findResult = await collection.find({}).toArray();

console.log("🔍 All documents:", findResult);

🔹 find({}): the empty {} means “get *all* documents”  
🔹 .toArray(): turns the result into a JavaScript array  
📄 This is how you **read** data

js

CopyEdit

// ✅ 12. Count the number of documents

const countResult = await collection.countDocuments({});

console.log("📊 Total documents in User collection:", countResult);

🔹 Counts how many total documents are inside the User collection  
🎯 Very useful for checking how much data you've stored

js

CopyEdit

// ✅ 13. Find how many users have firstname "Deepika"

const filteredCount = await collection.countDocuments({ firstname: "Deepika" });

console.log("📁 Documents with firstname 'Deepika':", filteredCount);

🔎 This is a **filtered query**  
Only returns documents where firstname equals "Deepika"  
You can change the filter to city: "Mumbai" etc.

js

CopyEdit

// ✅ 14. Finish the function with a success message

return "✅ Done with operations.";

}

🟢 This is just a return value to mark end of function

js

CopyEdit

// ✅ 15. Run the function and handle errors

main()

.then(console.log)

.catch(console.error)

.finally(() => client.close());

| **Block** | **Purpose** |
| --- | --- |
| .then() | Runs if main() succeeds |
| .catch() | Runs if something goes wrong (like bad connection string) |
| .finally() | Always runs — we use it to **close the database connection** safely |

**✅ Summary in Plain English**

| **Section** | **What It Does** |
| --- | --- |
| MongoClient | Lets you connect to MongoDB |
| connect() | Opens the connection |
| db() | Selects the database |
| collection() | Selects a table-like storage |
| insertOne() | Adds a new record |
| find() | Reads records |
| countDocuments() | Counts records (optionally with filters) |
| close() | Closes connection to DB safely |

**🔥 Full MongoDB CRUD with Node.js (No Mongoose)**

📁 File: database.js

# 🧠 MongoDB CRUD (All in One Script) — Beginner Super Guide

📁 **File**: database.js  
🧪 Use: Native MongoDB Driver (not Mongoose)  
✅ Fully commented and complete

js

CopyEdit

// 1️⃣ Import MongoClient from the mongodb package

const { MongoClient } = require("mongodb");

// 2️⃣ Your MongoDB connection URI (use your real one)

const url = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

// 3️⃣ Define the database name

const dbName = "HelloWorld";

// 4️⃣ Create a new MongoClient object

const client = new MongoClient(url);

// 5️⃣ Main async function

async function main() {

await client.connect(); // Connect to MongoDB

console.log("✅ Connected to MongoDB Atlas");

const db = client.db(dbName); // Get your database

const collection = db.collection("User"); // Create or access the 'User' collection

// ================== INSERT ONE ==================

const oneUser = {

firstname: "Likan",

lastname: "Mishra",

city: "Sambalpur",

phone: "7000000000"

};

const insertOneResult = await collection.insertOne(oneUser);

console.log("📥 Inserted One Document ID:", insertOneResult.insertedId);

// ================== INSERT MANY ==================

const manyUsers = [

{ firstname: "Ranveer", lastname: "Singh", city: "Mumbai", phone: "9999911111" },

{ firstname: "Deepika", lastname: "Padukone", city: "Mumbai", phone: "8888822222" },

{ firstname: "Aamir", lastname: "Khan", city: "Delhi", phone: "7777733333" },

];

const insertManyResult = await collection.insertMany(manyUsers);

console.log("📥 Inserted Many IDs:", Object.values(insertManyResult.insertedIds));

// ================== FIND ALL ==================

const allUsers = await collection.find({}).toArray();

console.log("🔍 All Users:");

console.table(allUsers); // Nicely prints the table in terminal

// ================== FIND WITH FILTER ==================

const mumbaiUsers = await collection.find({ city: "Mumbai" }).toArray();

console.log("🌆 Users in Mumbai:");

console.table(mumbaiUsers);

// ================== FIND ONE ==================

const onePerson = await collection.findOne({ firstname: "Likan" });

console.log("🧑 First User with firstname 'Likan':", onePerson);

// ================== COUNT ==================

const totalCount = await collection.countDocuments({});

console.log("📊 Total Documents in Collection:", totalCount);

// ================== UPDATE ONE ==================

const updateOne = await collection.updateOne(

{ firstname: "Likan" }, // Filter

{ $set: { phone: "1111122222" } } // Update field

);

console.log("🛠️ Updated One Document:", updateOne.modifiedCount);

// ================== UPDATE MANY ==================

const updateMany = await collection.updateMany(

{ city: "Mumbai" }, // Filter

{ $set: { isMetroCity: true } } // Add field

);

console.log("🛠️ Updated Many Documents:", updateMany.modifiedCount);

// ================== ADD NEW FIELD ==================

await collection.updateOne(

{ firstname: "Aamir" },

{ $set: { profession: "Actor" } }

);

// ================== RENAME FIELD ==================

await collection.updateOne(

{ firstname: "Deepika" },

{ $rename: { phone: "phoneNumber" } }

);

// ================== UNSET (REMOVE FIELD) ==================

await collection.updateOne(

{ firstname: "Ranveer" },

{ $unset: { lastname: "" } }

);

// ================== DELETE ONE ==================

const deleteOne = await collection.deleteOne({ firstname: "Likan" });

console.log("🗑️ Deleted One Document:", deleteOne.deletedCount);

// ================== DELETE MANY ==================

const deleteMany = await collection.deleteMany({ city: "Delhi" });

console.log("🗑️ Deleted Many Documents:", deleteMany.deletedCount);

// ================== FINAL READ ==================

const finalState = await collection.find({}).toArray();

console.log("📦 Final Data in Collection:");

console.table(finalState);

return "✅ CRUD Operations Completed";

}

// Run the function

main()

.then(console.log)

.catch(console.error)

.finally(() => client.close()); // Always close the client

# 🔎 Sample Console Output (Expected)

yaml

CopyEdit

✅ Connected to MongoDB Atlas

📥 Inserted One Document ID: 652e2...

📥 Inserted Many IDs: [ '652e3...', '652e4...', '652e5...' ]

🔍 All Users:

┌─────────┬────────────┬───────────────┬────────────┬──────────────┐

│ \_id │ firstname │ lastname │ city │ phone │

├─────────┼────────────┼───────────────┼────────────┼──────────────┤

│ ... │ Likan │ Mishra │ Sambalpur │ 7000000000 │

│ ... │ Ranveer │ Singh │ Mumbai │ 9999911111 │

│ ... │ Deepika │ Padukone │ Mumbai │ 8888822222 │

│ ... │ Aamir │ Khan │ Delhi │ 7777733333 │

└─────────┴────────────┴───────────────┴────────────┴──────────────┘

🌆 Users in Mumbai:

🧑 First User with firstname 'Likan': { ...document here... }

📊 Total Documents in Collection: 4

🛠️ Updated One Document: 1

🛠️ Updated Many Documents: 2

🗑️ Deleted One Document: 1

🗑️ Deleted Many Documents: 1

📦 Final Data in Collection:

[ Ranveer, Deepika ]

✅ CRUD Operations Completed

**🧠 Cheat Sheet of MongoDB Methods**

| **Operation** | **Method** | **Example** |
| --- | --- | --- |
| Insert One | insertOne(doc) | Add 1 user |
| Insert Many | insertMany([docs]) | Add multiple users |
| Find All | find({}).toArray() | Get all records |
| Find One | findOne(filter) | Get one match |
| Find with Filter | find({ key: value }) | Search by field |
| Count | countDocuments({}) | Count documents |
| Update One | updateOne(filter, update) | Update first match |
| Update Many | updateMany(filter, update) | Update multiple |
| Delete One | deleteOne(filter) | Remove one |
| Delete Many | deleteMany(filter) | Remove multiple |

**Advanced Queries –**

**const { MongoClient } = require("mongodb");**

**const url = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";**

**const client = new MongoClient(url);**

**const dbName = "HelloWorld";**

**async function main() {**

**await client.connect();**

**console.log("✅ Connected to MongoDB");**

**const db = client.db(dbName);**

**const collection = db.collection("Employees");**

**await collection.deleteMany({});**

**const employees = [**

**{ name: "Likan", age: 30, city: "Delhi", skills: ["JavaScript", "Node"], salary: 70000 },**

**{ name: "Amit", age: 24, city: "Mumbai", skills: ["Java", "MongoDB"], salary: 50000 },**

**{ name: "Sara", age: 28, city: "Delhi", skills: ["Python"], salary: 60000 },**

**{ name: "Zara", age: 32, city: "Bangalore", skills: ["JavaScript", "React"], salary: 80000 },**

**{ name: "John", age: 40, city: "Mumbai", skills: ["Go", "DevOps", "Cloud"], salary: 90000 },**

**{ name: "Ravi", age: 22, city: "Delhi", skills: [], salary: 30000 },**

**];**

**await collection.insertMany(employees);**

**console.log("📥 Data seeded\n");**

**// 1. age > 30**

**console.log("\n🔍 age > 30");**

**const res1 = await collection.find({ age: { $gt: 30 } }).toArray();**

**console.table(res1);**

**// 2. salary < 60000**

**console.log("\n🔍 salary < 60000");**

**const res2 = await collection.find({ salary: { $lt: 60000 } }).toArray();**

**console.table(res2);**

**// 3. city != 'Mumbai'**

**console.log("\n🔍 city != 'Mumbai'");**

**const res3 = await collection.find({ city: { $ne: "Mumbai" } }).toArray();**

**console.table(res3);**

**// 4. age == 30**

**console.log("\n🔍 age == 30");**

**const res4 = await collection.find({ age: { $eq: 30 } }).toArray();**

**console.table(res4);**

**// 5. city == 'Delhi' AND salary > 50000**

**console.log("\n🔍 city == 'Delhi' AND salary > 50000");**

**const res5 = await collection.find({**

**$and: [{ city: "Delhi" }, { salary: { $gt: 50000 } }]**

**}).toArray();**

**console.table(res5);**

**// 6. city == 'Mumbai' OR city == 'Bangalore'**

**console.log("\n🔍 city == 'Mumbai' OR city == 'Bangalore'");**

**const res6 = await collection.find({**

**$or: [{ city: "Mumbai" }, { city: "Bangalore" }]**

**}).toArray();**

**console.table(res6);**

**// 7. city NOT 'Delhi'**

**console.log("\n🔍 city NOT 'Delhi'");**

**const res7 = await collection.find({**

**city: { $not: { $eq: "Delhi" } }**

**}).toArray();**

**console.table(res7);**

**// 8. NOT (city == 'Mumbai' OR salary > 80000)**

**console.log("\n🔍 NOT (city == 'Mumbai' OR salary > 80000)");**

**const res8 = await collection.find({**

**$nor: [{ city: "Mumbai" }, { salary: { $gt: 80000 } }]**

**}).toArray();**

**console.table(res8);**

**// 9. city IN ['Delhi', 'Mumbai']**

**console.log("\n🔍 city IN ['Delhi', 'Mumbai']");**

**const res9 = await collection.find({**

**city: { $in: ["Delhi", "Mumbai"] }**

**}).toArray();**

**console.table(res9);**

**// 10. city NOT IN ['Delhi', 'Mumbai']**

**console.log("\n🔍 city NOT IN ['Delhi', 'Mumbai']");**

**const res10 = await collection.find({**

**city: { $nin: ["Delhi", "Mumbai"] }**

**}).toArray();**

**console.table(res10);**

**// 11. skills contains 'JavaScript'**

**console.log("\n🔍 skills contains 'JavaScript'");**

**const res11 = await collection.find({**

**skills: "JavaScript"**

**}).toArray();**

**console.table(res11);**

**// 12. skills.size == 3**

**console.log("\n🔍 skills.size == 3");**

**const res12 = await collection.find({**

**skills: { $size: 3 }**

**}).toArray();**

**console.table(res12);**

**// 13. skills contains ALL ['JavaScript', 'React']**

**console.log("\n🔍 skills contains ALL ['JavaScript', 'React']");**

**const res13 = await collection.find({**

**skills: { $all: ["JavaScript", "React"] }**

**}).toArray();**

**console.table(res13);**

**// 14. Top 3 highest salary**

**console.log("\n🔍 Top 3 highest salary");**

**const res14 = await collection.find({}).sort({ salary: -1 }).limit(3).toArray();**

**console.table(res14);**

**// 15. Skip top 2, next 2 highest salary**

**console.log("\n🔍 Skip top 2, then show next 2 highest salaries");**

**const res15 = await collection.find({}).sort({ salary: -1 }).skip(2).limit(2).toArray();**

**console.table(res15);**

**return "\n✅ All queries completed (no loop used)";**

**}**

**main()**

**.then(console.log)**

**.catch(console.error)**

**.finally(() => client.close());**

### ✅ Output for Each Query (with Matching Documents)

css

CopyEdit

🔍 Query: age > 30

✔️ { name: 'Zara', age: 32, city: 'Bangalore', skills: ['JavaScript', 'React'], salary: 80000 }

✔️ { name: 'John', age: 40, city: 'Mumbai', skills: ['Go', 'DevOps', 'Cloud'], salary: 90000 }

🔍 Query: salary < 60000

✔️ { name: 'Amit', age: 24, city: 'Mumbai', skills: ['Java', 'MongoDB'], salary: 50000 }

✔️ { name: 'Ravi', age: 22, city: 'Delhi', skills: [], salary: 30000 }

🔍 Query: city != 'Mumbai'

✔️ { name: 'Likan', age: 30, city: 'Delhi', skills: ['JavaScript', 'Node'], salary: 70000 }

✔️ { name: 'Sara', age: 28, city: 'Delhi', skills: ['Python'], salary: 60000 }

✔️ { name: 'Zara', age: 32, city: 'Bangalore', skills: ['JavaScript', 'React'], salary: 80000 }

✔️ { name: 'Ravi', age: 22, city: 'Delhi', skills: [], salary: 30000 }

🔍 Query: age == 30

✔️ { name: 'Likan', age: 30, city: 'Delhi', skills: ['JavaScript', 'Node'], salary: 70000 }

🔍 Query: city == 'Delhi' AND salary > 50000

✔️ { name: 'Likan', age: 30, city: 'Delhi', skills: ['JavaScript', 'Node'], salary: 70000 }

✔️ { name: 'Sara', age: 28, city: 'Delhi', skills: ['Python'], salary: 60000 }

🔍 Query: city == 'Mumbai' OR city == 'Bangalore'

✔️ { name: 'Amit', age: 24, city: 'Mumbai', skills: ['Java', 'MongoDB'], salary: 50000 }

✔️ { name: 'Zara', age: 32, city: 'Bangalore', skills: ['JavaScript', 'React'], salary: 80000 }

✔️ { name: 'John', age: 40, city: 'Mumbai', skills: ['Go', 'DevOps', 'Cloud'], salary: 90000 }

🔍 Query: city NOT 'Delhi'

✔️ { name: 'Amit', age: 24, city: 'Mumbai', skills: ['Java', 'MongoDB'], salary: 50000 }

✔️ { name: 'Zara', age: 32, city: 'Bangalore', skills: ['JavaScript', 'React'], salary: 80000 }

✔️ { name: 'John', age: 40, city: 'Mumbai', skills: ['Go', 'DevOps', 'Cloud'], salary: 90000 }

🔍 Query: NOT (city == 'Mumbai' OR salary > 80000)

✔️ { name: 'Likan', age: 30, city: 'Delhi', skills: ['JavaScript', 'Node'], salary: 70000 }

✔️ { name: 'Sara', age: 28, city: 'Delhi', skills: ['Python'], salary: 60000 }

✔️ { name: 'Zara', age: 32, city: 'Bangalore', skills: ['JavaScript', 'React'], salary: 80000 }

✔️ { name: 'Ravi', age: 22, city: 'Delhi', skills: [], salary: 30000 }

🔍 Query: city IN ['Delhi', 'Mumbai']

✔️ { name: 'Likan', age: 30, city: 'Delhi', ... }

✔️ { name: 'Amit', age: 24, city: 'Mumbai', ... }

✔️ { name: 'Sara', age: 28, city: 'Delhi', ... }

✔️ { name: 'John', age: 40, city: 'Mumbai', ... }

✔️ { name: 'Ravi', age: 22, city: 'Delhi', ... }

🔍 Query: city NOT IN ['Delhi', 'Mumbai']

✔️ { name: 'Zara', age: 32, city: 'Bangalore', ... }

🔍 Query: skills contains 'JavaScript'

✔️ Likan, Zara

🔍 Query: skills.size == 3

✔️ John

🔍 Query: skills contains ALL ['JavaScript', 'React']

✔️ Zara

🔍 Query: Top 3 highest salary

✔️ John (90000), Zara (80000), Likan (70000)

🔍 Query: Skip top 2, next 2 salaries

✔️ Likan (70000), Sara (60000)

**🔁 .sort(), .limit(), .skip() in MongoDB**

These 3 are used together to:

* Sort the documents
* Pick how many you want
* Skip certain number for pagination

**✅ 1️⃣ .sort({ field: 1 | -1 })**

Used to **order** your data by a specific field.

js

CopyEdit

.sort({ salary: -1 }) // Highest to lowest

.sort({ age: 1 }) // Lowest to highest

| **Field** | **1** | **-1** |
| --- | --- | --- |
| age: 1 | Ascending | Descending |
| name: 1 | A → Z | Z → A |

**✅ 2️⃣ .limit(n)**

Used to **limit** the number of documents you get.

js

CopyEdit

.limit(3)

📌 Shows only **top 3 results** (based on whatever you sorted)

**✅ 3️⃣ .skip(n)**

Used to **skip** the first N results — mostly used in pagination.

js

CopyEdit

.skip(2)

Skips the **first 2 documents**, and shows the rest.

**📦 Combined Example:**

js

CopyEdit

collection

.find({})

.sort({ salary: -1 }) // highest paid first

.skip(2) // skip top 2

.limit(2); // get next 2

**🔍 What's Happening?**

1. MongoDB sorts employees by salary DESCENDING (💰 highest first)
2. Skips the top 2 richest
3. Returns the **3rd and 4th richest**

**🧠 Real-World Use Case: Pagination**

js

CopyEdit

const page = 2;

const limit = 5;

collection

.find({})

.skip((page - 1) \* limit) // e.g., skip 5

.limit(limit); // show next 5

This lets you paginate results like:

| **Page** | **Skipped** | **Shown** |
| --- | --- | --- |
| 1 | 0 | 1 → 5 |
| 2 | 5 | 6 → 10 |
| 3 | 10 | 11 → 15 |

**🔥 Visual Example (with .sort().limit().skip())**

js

CopyEdit

.sort({ salary: -1 }) → [John, Zara, Likan, Sara, Amit, Ravi]

.skip(2) → [Likan, Sara, Amit, Ravi]

.limit(2) → [Likan, Sara]

**MongoDB Operators You Must Know**

| **Operator** | **What It Does** |
| --- | --- |
| $set | Add or update fields |
| $unset | Remove fields |
| $rename | Rename a field |
| $inc | Increment numeric value |
| $push | Add to array |
| $pull | Remove from array |
| $in | Match if field is in a list |

**📌 Pro Tips**

* MongoDB is **schema-less** → it accepts any fields
* \_id is auto-generated for each document
* If a field doesn't exist, $set will create it
* Always close your DB connection using .finally(() => client.close())

🛠️ Must-Know MongoDB Operators (With Native Node.js Driver) –

const { MongoClient } = require("mongodb");

const url = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

const client = new MongoClient(url);

const dbName = "HelloWorld";

async function main() {

await client.connect();

console.log("✅ Connected");

const db = client.db(dbName);

const collection = db.collection("OperatorsDemo");

// 🧹 Clean slate

await collection.deleteMany({});

await collection.insertOne({

name: "Likan",

age: 30,

city: "Sambalpur",

score: 5,

hobbies: ["gaming", "music"]

});

console.log("📥 Inserted base document\n");

// 1️⃣ $set → Add or Update Fields

console.log("\n🔧 Using $set → Add new field 'status'");

await collection.updateOne({ name: "Likan" }, { $set: { status: "active" } });

console.table(await collection.find({}).toArray());

// 2️⃣ $unset → Remove Field

console.log("\n❌ Using $unset → Remove 'city'");

await collection.updateOne({ name: "Likan" }, { $unset: { city: "" } });

console.table(await collection.find({}).toArray());

// 3️⃣ $rename → Rename Field

console.log("\n✏️ Using $rename → Rename 'score' to 'points'");

await collection.updateOne({ name: "Likan" }, { $rename: { score: "points" } });

console.table(await collection.find({}).toArray());

// 4️⃣ $inc → Increment Numeric Field

console.log("\n➕ Using $inc → Increment 'points' by 10");

await collection.updateOne({ name: "Likan" }, { $inc: { points: 10 } });

console.table(await collection.find({}).toArray());

// 5️⃣ $push → Add Value to Array

console.log("\n➕ Using $push → Add 'coding' to hobbies array");

await collection.updateOne({ name: "Likan" }, { $push: { hobbies: "coding" } });

console.table(await collection.find({}).toArray());

// 6️⃣ $pull → Remove Value from Array

console.log("\n➖ Using $pull → Remove 'music' from hobbies array");

await collection.updateOne({ name: "Likan" }, { $pull: { hobbies: "music" } });

console.table(await collection.find({}).toArray());

// 7️⃣ $in → Query: Is hobby in [‘gaming’, ‘coding’]?

console.log("\n🎯 Using $in → Find docs with hobby in ['gaming', 'coding']");

const inResult = await collection.find({ hobbies: { $in: ["gaming", "coding"] } }).toArray();

console.table(inResult);

return "✅ All update operator examples complete";

}

main()

.then(console.log)

.catch(console.error)

.finally(() => client.close());

## 🔍 Output Preview (Console):

bash

CopyEdit

📥 Inserted base document

🔧 Using $set → Add 'status': 'active'

✔️ { name: 'Likan', age: 30, city: 'Sambalpur', status: 'active' }

❌ Using $unset → Removed 'city'

✔️ { name: 'Likan', age: 30, status: 'active' }

✏️ Using $rename → 'score' → 'points'

✔️ { name: 'Likan', points: 5, status: 'active' }

➕ Using $inc → points: 5 → 15

✔️ { name: 'Likan', points: 15 }

➕ Using $push → hobbies += 'coding'

✔️ hobbies: [ 'gaming', 'music', 'coding' ]

➖ Using $pull → remove 'music'

✔️ hobbies: [ 'gaming', 'coding' ]

🎯 Using $in → Find where hobby in ['gaming', 'coding']

✔️ Matched: 'Likan'

## 🔒 Security Tip

Never commit your real MongoDB password into GitHub.  
Use .env file:

ini

CopyEdit

# .env

MONGO\_URI="mongodb+srv://likan:123456@cluster0.mongodb.net"

js

CopyEdit

require('dotenv').config();

const client = new MongoClient(process.env.MONGO\_URI);

with mongoose –

CRUD -  
// 1\_crud\_mongoose\_v2.js

const mongoose = require("mongoose");

// 🧠 What is a Database (DB)?

// A MongoDB "Database" is like a folder that stores multiple "collections"

// Example DB name: test, myAppDB, employeeDB

// 🧠 What is a Collection?

// A "collection" is like a table in SQL — it stores multiple "documents" (rows)

// Example collection name: users, orders, products

// 🧠 What is a Field?

// Each document (row) has multiple fields (columns) like name, age, city

// 🔌 Replace with your actual connection string from MongoDB Atlas

const uri = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

// 🔗 Connect to MongoDB

mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("✅ Connected to MongoDB Atlas"))

.catch(err => console.error("❌ Connection Error:", err));

// 🧱 Schema — defines structure of documents in a collection

const userSchema = new mongoose.Schema({

name: String,

age: Number,

city: String,

skills: [String], // Array of skills

isActive: Boolean

});

// 🏷️ Model — like a controller to interact with "User" collection

const User = mongoose.model("User", userSchema);

// 🚀 Main function to run CRUD operations

async function main() {

// 🧹 Delete all old documents before starting

await User.deleteMany({});

console.log("🧹 Cleared existing user data");

// ✅ 1. Insert One User

const user1 = await User.create({

name: "Likan",

age: 30,

city: "Sambalpur",

skills: ["Node.js", "MongoDB"],

isActive: true

});

// 🧠 Why use .toObject()?

// It converts Mongoose's Document instance into plain JavaScript object (so console.table can format it better)

console.log("\n✅ 1. One User Inserted:");

console.table([user1.toObject()]);

// ✅ 2. Insert Many Users

const manyUsers = await User.insertMany([

{ name: "Zara", age: 24, city: "Mumbai", skills: ["React"], isActive: true },

{ name: "Ravi", age: 28, city: "Delhi", skills: ["Java"], isActive: false },

{ name: "Sara", age: 35, city: "Bangalore", skills: ["Python"], isActive: true },

]);

console.log("\n✅ 2. Many Users Inserted:");

console.table(manyUsers.map(u => u.toObject()));

// ✅ 3. Find All Users

const allUsers = await User.find();

console.log("\n🔍 3. All Users:");

console.table(allUsers.map(u => u.toObject()));

// ✅ 4. Find One User (find Zara)

const singleUser = await User.findOne({ name: "Zara" });

console.log("\n🔍 4. Find One (name = Zara):");

console.table([singleUser.toObject()]);

// ✅ 5. Update One (Likan's city → Bhubaneswar)

await User.updateOne({ name: "Likan" }, { $set: { city: "Bhubaneswar" } });

const updated = await User.findOne({ name: "Likan" });

console.log("\n✏️ 5. Updated One (Likan's city):");

console.table([updated.toObject()]);

// ✅ 6. Update Many (Set isActive = false for all)

await User.updateMany({}, { $set: { isActive: false } });

const afterUpdateMany = await User.find();

console.log("\n✏️ 6. Updated Many (isActive → false):");

console.table(afterUpdateMany.map(u => u.toObject()));

// ✅ 7. Delete One (remove Sara)

await User.deleteOne({ name: "Sara" });

const afterDeleteOne = await User.find();

console.log("\n❌ 7. Deleted One (Sara):");

console.table(afterDeleteOne.map(u => u.toObject()));

// ✅ 8. Delete Many (Delete users from city Mumbai or Delhi)

await User.deleteMany({ city: { $in: ["Delhi", "Mumbai"] } });

const finalUsers = await User.find();

console.log("\n❌ 8. Deleted Many (city in Delhi or Mumbai):");

console.table(finalUsers.map(u => u.toObject()));

}

// 🏁 Start everything

main()

.then(() => mongoose.disconnect()) // Close connection after done

.catch(console.error);

# 🖥️ ✅ Terminal Output: With All CRUD Steps

rust

CopyEdit

✅ Connected to MongoDB Atlas

🧹 Cleared existing user data

✅ 1. One User Inserted:

┌─────────┬──────────┬─────┬──────────────┬────────────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼──────────────┼────────────────────┼───────────┤

│ 0 │ 'Likan' │ 30 │ 'Sambalpur' │ ['Node.js','MongoDB'] │ true │

└─────────┴──────────┴─────┴──────────────┴────────────────────┴───────────┘

✅ 2. Many Users Inserted:

┌─────────┬──────────┬─────┬────────────┬──────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼────────────┼──────────────┼───────────┤

│ 0 │ 'Zara' │ 24 │ 'Mumbai' │ ['React'] │ true │

│ 1 │ 'Ravi' │ 28 │ 'Delhi' │ ['Java'] │ false │

│ 2 │ 'Sara' │ 35 │ 'Bangalore'│ ['Python'] │ true │

└─────────┴──────────┴─────┴────────────┴──────────────┴───────────┘

🔍 3. All Users:

┌─────────┬──────────┬─────┬──────────────┬────────────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼──────────────┼────────────────────┼───────────┤

│ 0 │ 'Likan' │ 30 │ 'Sambalpur' │ ['Node.js','MongoDB'] │ true │

│ 1 │ 'Zara' │ 24 │ 'Mumbai' │ ['React'] │ true │

│ 2 │ 'Ravi' │ 28 │ 'Delhi' │ ['Java'] │ false │

│ 3 │ 'Sara' │ 35 │ 'Bangalore' │ ['Python'] │ true │

└─────────┴──────────┴─────┴──────────────┴────────────────────┴───────────┘

🔍 4. Find One (name = Zara):

┌─────────┬──────────┬─────┬──────────┬────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼──────────┼────────────┼───────────┤

│ 0 │ 'Zara' │ 24 │ 'Mumbai' │ ['React'] │ true │

└─────────┴──────────┴─────┴──────────┴────────────┴───────────┘

✏️ 5. Updated One (Likan's city):

┌─────────┬──────────┬─────┬───────────────┬────────────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼───────────────┼────────────────────┼───────────┤

│ 0 │ 'Likan' │ 30 │ 'Bhubaneswar' │ ['Node.js','MongoDB'] │ true │

└─────────┴──────────┴─────┴───────────────┴────────────────────┴───────────┘

✏️ 6. Updated Many (isActive → false):

┌─────────┬──────────┬─────┬───────────────┬────────────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼───────────────┼────────────────────┼───────────┤

│ 0 │ 'Likan' │ 30 │ 'Bhubaneswar' │ ['Node.js','MongoDB'] │ false │

│ 1 │ 'Zara' │ 24 │ 'Mumbai' │ ['React'] │ false │

│ 2 │ 'Ravi' │ 28 │ 'Delhi' │ ['Java'] │ false │

│ 3 │ 'Sara' │ 35 │ 'Bangalore' │ ['Python'] │ false │

└─────────┴──────────┴─────┴───────────────┴────────────────────┴───────────┘

❌ 7. Deleted One (Sara):

┌─────────┬──────────┬─────┬───────────────┬────────────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼───────────────┼────────────────────┼───────────┤

│ 0 │ 'Likan' │ 30 │ 'Bhubaneswar' │ ['Node.js','MongoDB'] │ false │

│ 1 │ 'Zara' │ 24 │ 'Mumbai' │ ['React'] │ false │

│ 2 │ 'Ravi' │ 28 │ 'Delhi' │ ['Java'] │ false │

└─────────┴──────────┴─────┴───────────────┴────────────────────┴───────────┘

❌ 8. Deleted Many (city in Delhi or Mumbai):

┌─────────┬──────────┬─────┬───────────────┬────────────────────┬───────────┐

│ (index) │ name │ age │ city │ skills │ isActive │

├─────────┼──────────┼─────┼───────────────┼────────────────────┼───────────┤

│ 0 │ 'Likan' │ 30 │ 'Bhubaneswar' │ ['Node.js','MongoDB'] │ false │

└─────────┴──────────┴─────┴───────────────┴────────────────────┴───────────┘

Filters –

// 2\_filters\_mongoose.js

const mongoose = require("mongoose");

// ✅ Replace with your connection string

const uri = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("✅ Connected to MongoDB"))

.catch(err => console.error("❌ Connection Error:", err));

// 🧱 Schema & Model

const userSchema = new mongoose.Schema({

name: String,

age: Number,

city: String,

skills: [String],

salary: Number

});

const User = mongoose.model("FilterUser", userSchema);

async function runFilters() {

await User.deleteMany({});

// 🌱 Insert sample data

await User.insertMany([

{ name: "Likan", age: 30, city: "Delhi", skills: ["JS", "Node"], salary: 70000 },

{ name: "Amit", age: 24, city: "Mumbai", skills: ["Java", "Mongo"], salary: 50000 },

{ name: "Sara", age: 28, city: "Delhi", skills: ["Python"], salary: 60000 },

{ name: "Zara", age: 32, city: "Bangalore", skills: ["React", "JS"], salary: 80000 },

{ name: "John", age: 40, city: "Mumbai", skills: ["Go", "Cloud"], salary: 90000 },

{ name: "Ravi", age: 22, city: "Delhi", skills: [], salary: 30000 }

]);

// 🔍 Filters

console.log("\n1️⃣ age > 30 ($gt)");

console.table((await User.find({ age: { $gt: 30 } })).map(u => u.toObject()));

console.log("\n2️⃣ salary < 60000 ($lt)");

console.table((await User.find({ salary: { $lt: 60000 } })).map(u => u.toObject()));

console.log("\n3️⃣ city != 'Mumbai' ($ne)");

console.table((await User.find({ city: { $ne: "Mumbai" } })).map(u => u.toObject()));

console.log("\n4️⃣ age == 30 ($eq)");

console.table((await User.find({ age: { $eq: 30 } })).map(u => u.toObject()));

console.log("\n5️⃣ city = 'Delhi' AND salary > 50000 ($and)");

console.table((await User.find({ $and: [{ city: "Delhi" }, { salary: { $gt: 50000 } }] })).map(u => u.toObject()));

console.log("\n6️⃣ city = 'Mumbai' OR city = 'Bangalore' ($or)");

console.table((await User.find({ $or: [{ city: "Mumbai" }, { city: "Bangalore" }] })).map(u => u.toObject()));

console.log("\n7️⃣ NOT Delhi ($not)");

console.table((await User.find({ city: { $not: { $eq: "Delhi" } } })).map(u => u.toObject()));

console.log("\n8️⃣ city in ['Delhi', 'Mumbai'] ($in)");

console.table((await User.find({ city: { $in: ["Delhi", "Mumbai"] } })).map(u => u.toObject()));

console.log("\n9️⃣ city NOT IN ['Delhi', 'Mumbai'] ($nin)");

console.table((await User.find({ city: { $nin: ["Delhi", "Mumbai"] } })).map(u => u.toObject()));

console.log("\n🔟 skills has 'JS' ($elemMatch)");

console.table((await User.find({ skills: { $elemMatch: { $eq: "JS" } } })).map(u => u.toObject()));

console.log("\n1️⃣1️⃣ skills.length == 0 ($size)");

console.table((await User.find({ skills: { $size: 0 } })).map(u => u.toObject()));

console.log("\n1️⃣2️⃣ sort by age descending");

console.table((await User.find().sort({ age: -1 })).map(u => u.toObject()));

console.log("\n1️⃣3️⃣ pagination: skip 2, limit 2");

console.table((await User.find().skip(2).limit(2)).map(u => u.toObject()));

mongoose.disconnect();

}

runFilters();

output –

1️⃣ age > 30 ($gt)

┌─────────┬──────────┬─────┬────────────┬─────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼────────────┼─────────────────────┼─────────┤

│ 0 │ "Zara" │ 32 │ "Bangalore"│ ["React", "JS"] │ 80000 │

│ 1 │ "John" │ 40 │ "Mumbai" │ ["Go", "Cloud"] │ 90000 │

└─────────┴──────────┴─────┴────────────┴─────────────────────┴─────────┘

2️⃣ salary < 60000 ($lt)

┌─────────┬──────────┬─────┬──────────┬─────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼──────────┼─────────────────────┼─────────┤

│ 0 │ "Amit" │ 24 │ "Mumbai" │ ["Java", "Mongo"] │ 50000 │

│ 1 │ "Ravi" │ 22 │ "Delhi" │ [] │ 30000 │

└─────────┴──────────┴─────┴──────────┴─────────────────────┴─────────┘

3️⃣ city != 'Mumbai' ($ne)

┌─────────┬──────────┬─────┬────────────┬────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼────────────┼────────────────────┼─────────┤

│ 0 │ "Likan" │ 30 │ "Delhi" │ ["JS", "Node"] │ 70000 │

│ 1 │ "Sara" │ 28 │ "Delhi" │ ["Python"] │ 60000 │

│ 2 │ "Zara" │ 32 │ "Bangalore"│ ["React", "JS"] │ 80000 │

│ 3 │ "Ravi" │ 22 │ "Delhi" │ [] │ 30000 │

└─────────┴──────────┴─────┴────────────┴────────────────────┴─────────┘

4️⃣ age == 30 ($eq)

┌─────────┬──────────┬─────┬─────────┬───────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼─────────┼───────────────────┼─────────┤

│ 0 │ "Likan" │ 30 │ "Delhi" │ ["JS", "Node"] │ 70000 │

└─────────┴──────────┴─────┴─────────┴───────────────────┴─────────┘

5️⃣ city = 'Delhi' AND salary > 50000 ($and)

┌─────────┬──────────┬─────┬─────────┬──────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼─────────┼──────────────────┼─────────┤

│ 0 │ "Likan" │ 30 │ "Delhi" │ ["JS", "Node"] │ 70000 │

│ 1 │ "Sara" │ 28 │ "Delhi" │ ["Python"] │ 60000 │

└─────────┴──────────┴─────┴─────────┴──────────────────┴─────────┘

6️⃣ city = 'Mumbai' OR city = 'Bangalore' ($or)

┌─────────┬──────────┬─────┬────────────┬─────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼────────────┼─────────────────────┼─────────┤

│ 0 │ "Amit" │ 24 │ "Mumbai" │ ["Java", "Mongo"] │ 50000 │

│ 1 │ "Zara" │ 32 │ "Bangalore"│ ["React", "JS"] │ 80000 │

│ 2 │ "John" │ 40 │ "Mumbai" │ ["Go", "Cloud"] │ 90000 │

└─────────┴──────────┴─────┴────────────┴─────────────────────┴─────────┘

7️⃣ NOT Delhi ($not)

┌─────────┬──────────┬─────┬────────────┬─────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼────────────┼─────────────────────┼─────────┤

│ 0 │ "Amit" │ 24 │ "Mumbai" │ ["Java", "Mongo"] │ 50000 │

│ 1 │ "Zara" │ 32 │ "Bangalore"│ ["React", "JS"] │ 80000 │

│ 2 │ "John" │ 40 │ "Mumbai" │ ["Go", "Cloud"] │ 90000 │

└─────────┴──────────┴─────┴────────────┴─────────────────────┴─────────┘

8️⃣ city in ['Delhi', 'Mumbai'] ($in)

┌─────────┬──────────┬─────┬─────────┬────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼─────────┼────────────────────┼─────────┤

│ 0 │ "Likan" │ 30 │ "Delhi" │ ["JS", "Node"] │ 70000 │

│ 1 │ "Amit" │ 24 │ "Mumbai"│ ["Java", "Mongo"] │ 50000 │

│ 2 │ "Sara" │ 28 │ "Delhi" │ ["Python"] │ 60000 │

│ 3 │ "John" │ 40 │ "Mumbai"│ ["Go", "Cloud"] │ 90000 │

│ 4 │ "Ravi" │ 22 │ "Delhi" │ [] │ 30000 │

└─────────┴──────────┴─────┴─────────┴────────────────────┴─────────┘

9️⃣ city NOT IN ['Delhi', 'Mumbai'] ($nin)

┌─────────┬──────────┬─────┬────────────┬────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼────────────┼────────────────────┼─────────┤

│ 0 │ "Zara" │ 32 │ "Bangalore"│ ["React", "JS"] │ 80000 │

└─────────┴──────────┴─────┴────────────┴────────────────────┴─────────┘

🔟 skills has 'JS' ($elemMatch)

┌─────────┬──────────┬─────┬────────────┬────────────────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼────────────┼────────────────────┼─────────┤

│ 0 │ "Likan" │ 30 │ "Delhi" │ ["JS", "Node"] │ 70000 │

│ 1 │ "Zara" │ 32 │ "Bangalore"│ ["React", "JS"] │ 80000 │

└─────────┴──────────┴─────┴────────────┴────────────────────┴─────────┘

1️⃣1️⃣ skills.length == 0 ($size)

┌─────────┬──────────┬─────┬─────────┬────────┬─────────┐

│ (index) │ name │ age │ city │ skills │ salary │

├─────────┼──────────┼─────┼─────────┼────────┼─────────┤

│ 0 │ "Ravi" │ 22 │ "Delhi" │ [] │ 30000 │

└─────────┴──────────┴─────┴─────────┴────────┴─────────┘

1️⃣2️⃣ sort by age descending

(Sorted full list from age 40 to 22)

1️⃣3️⃣ pagination: skip 2, limit 2

┌─────────┬──────────┬─────┐

│ (index) │ name │ age │

├─────────┼──────────┼─────┤

│ 0 │ "Sara" │ 28 │

│ 1 │ "Zara" │ 32 │

└─────────┴──────────┴─────┘

### ✅ Advantages of Using Mongoose

* **Schema Definition:** Mongoose lets you define clear schemas for your collections, providing structure and consistency to your documents.
* **Built-in Validation:** It offers schema-level validation to ensure the data stored in the database is accurate and reliable.
* **Middleware Support:** Mongoose supports pre and post middleware hooks, enabling you to perform actions like validation, logging, or data transformation before or after database operations.

Update – Operators

// 3\_update\_ops\_mongoose.js

const mongoose = require("mongoose");

// ✅ Replace with your own MongoDB connection string

const uri = "mongodb+srv://likan:123456@cluster0.mongodb.net/?retryWrites=true&w=majority";

mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("✅ Connected to MongoDB"))

.catch(err => console.error("❌ Connection Error:", err));

// 🧱 Schema & Model

const userSchema = new mongoose.Schema({

name: String,

age: Number,

location: String,

city: String, // for renamed field

skills: [String],

score: Number,

status: String

});

const User = mongoose.model("UpdateUser", userSchema);

async function runUpdates() {

// 🧹 Clean existing data

await User.deleteMany({});

// 🌱 Insert initial user

const user = new User({

name: "Likan",

age: 30,

location: "Odisha",

skills: ["JS", "Node"],

score: 5,

status: "active"

});

await user.save();

console.log("\n📦 Initial User:");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

// 1️⃣ $set → Update score

await User.updateOne({ name: "Likan" }, { $set: { score: 10 } });

console.log("\n1️⃣ $set: Update score to 10");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

// 2️⃣ $unset → Remove status

await User.updateOne({ name: "Likan" }, { $unset: { status: "" } });

console.log("\n2️⃣ $unset: Remove status field");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

// 3️⃣ $rename → Rename 'location' to 'city'

await User.updateOne({ name: "Likan" }, { $rename: { location: "city" } });

console.log("\n3️⃣ $rename: Rename 'location' to 'city'");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

// 4️⃣ $inc → Increment score

await User.updateOne({ name: "Likan" }, { $inc: { score: 5 } });

console.log("\n4️⃣ $inc: Increment score by 5");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

// 5️⃣ $push → Add skill to array

await User.updateOne({ name: "Likan" }, { $push: { skills: "MongoDB" } });

console.log("\n5️⃣ $push: Add 'MongoDB' to skills");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

// 6️⃣ $pull → Remove 'JS' from array

await User.updateOne({ name: "Likan" }, { $pull: { skills: "JS" } });

console.log("\n6️⃣ $pull: Remove 'JS' from skills");

console.table([(await User.findOne({ name: "Likan" })).toObject()]);

await mongoose.disconnect();

}

runUpdates();

**✅ Terminal Output: Update Operators in Mongoose**

perl

CopyEdit

✅ Connected to MongoDB

📦 Initial User:

┌─────────┬────────────┐

│ (index) │ Values │

├─────────┼────────────┤

│ name │ 'Likan' │

│ age │ 30 │

│ location│ 'Odisha' │

│ city │ undefined │

│ skills │ ['JS', 'Node'] │

│ score │ 5 │

│ status │ 'active' │

└─────────┴────────────┘

1️⃣ $set: Update score to 10

┌─────────┬────────────┐

│ score │ 10 │

└─────────┴────────────┘

2️⃣ $unset: Remove status field

┌─────────┬────────────┐

│ status │ undefined │

└─────────┴────────────┘

3️⃣ $rename: Rename 'location' to 'city'

┌─────────┬────────────┐

│ location│ undefined │

│ city │ 'Odisha' │

└─────────┴────────────┘

4️⃣ $inc: Increment score by 5

┌─────────┬────────────┐

│ score │ 15 │

└─────────┴────────────┘

5️⃣ $push: Add 'MongoDB' to skills

┌─────────┬──────────────────────────┐

│ skills │ ['JS', 'Node', 'MongoDB']│

└─────────┴──────────────────────────┘

6️⃣ $pull: Remove 'JS' from skills

┌─────────┬────────────────────┐

│ skills │ ['Node', 'MongoDB']│

└─────────┴────────────────────┘

**💡 What Just Happened?**

Each log shows the state of the document after that operator was applied:

* $set: changed score from 5 → 10
* $unset: removed the status field
* $rename: moved value from location → city
* $inc: added 5 more to score → 15
* $push: added a new value to skills[]
* $pull: removed a value from skills[]

**>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>**

schema mongo db vs schema less monfoose

Cover rest of these topics with 2 catergories for each mongoose and mongo db

Leftover parts

### 🧭 MONGODB MASTER ROADMAP (2025 Edition)

#### 1️⃣ **Beginner Level: Foundation Layer**

* ✅ What is MongoDB? (NoSQL, document DB)
* ✅ BSON vs JSON
* ✅ Collections vs Tables
* ✅ Documents vs Rows
* ✅ Schema-less nature (unless using Mongoose)
* ✅ Installing MongoDB locally and using **MongoDB Atlas**

#### 2️⃣ **CRUD MASTERING with Node.js**

* 🛠️ Using MongoDB Native Driver
  + insertOne, insertMany
  + find, findOne
  + updateOne, updateMany, $set, $inc, $unset, $rename
  + deleteOne, deleteMany
* 📁 Using Mongoose for CRUD
  + Schema definition
  + Model creation
  + save(), find(), findByIdAndUpdate(), findByIdAndDelete()

🔍 Practice: Build a REST API using Express.js + MongoDB Native + Mongoose

#### 3️⃣ **FILTERING & QUERYING (Deep Dive)**

Master the full list of operators with behind-the-scenes behavior:

* $eq, $ne, $gt, $lt, $gte, $lte
* $in, $nin
* $and, $or, $nor, $not
* $exists, $regex
* $elemMatch, $size

🔁 Also:

* .sort(), .limit(), .skip()
* Use compound queries ({ $and: [ {age: {$gt: 18}}, {isActive: true} ] })

#### 4️⃣ **UPDATE OPERATORS (Internal Logic + Use Cases)**

* $set, $unset, $rename, $inc, $push, $pull, $addToSet
* Array modification
* Nested document update: 'user.name': 'Likan'

#### 5️⃣ **AGGREGATION PIPELINE**

🧠 This is a must for interviews!

* $match, $project, $group, $sort, $limit, $skip
* $unwind, $lookup, $addFields, $replaceRoot, $facet

📚 Use Cases:

* Analytics (grouping by month, sum)
* Pagination
* Lookup (JOIN in NoSQL)

#### 6️⃣ **INDEXING & PERFORMANCE**

* Creating indexes (.createIndex({ email: 1 }))
* Compound indexes
* TTL indexes
* Covered queries
* Index scan vs collection scan

📊 Tools: MongoDB Compass ➕ explain() method ➕ Profiler

#### 7️⃣ **MONGOOSE MASTERCLASS**

* Schema types: String, Number, Date, Boolean, Array, ObjectId
* Relationships: refs and populate
* Middlewares: pre, post
* Validation: built-in, custom
* Virtuals, statics, methods

#### 8️⃣ **SECURITY BEST PRACTICES**

* Hide credentials using .env
* Role-based access in MongoDB Atlas
* Field-level redaction
* Use HTTPS/SSL connections

#### 9️⃣ **DEPLOYMENT & BACKUP**

* MongoDB Atlas project setup
* Backups & restores
* Production cluster configuration
* MongoDB in Docker

#### 🔟 **INTERVIEW-LEVEL SCENARIOS & SYSTEM DESIGN**

* Designing user auth schema
* Handling nested comments (parent-child)
* Large-scale product catalog
* Indexing strategy for search filters
* Data modeling for denormalization