📝 MongoDB

🔷 Introduction

- **MongoDB** is a NoSQL, document-oriented database.
- Stores data in **JSON**.
- Schema-less: documents can have different structures.

🔷 Basic Terminology

RDBMS	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field

🔷 MongoDB Data Types

- String, Number (Int, Long, Double), Boolean
- Array
- Object (Embedded documents)
- Date
- Null
- ObjectId (Unique identifier)

📁 Collection Commands

```
db.createCollection("students")       # Create collection
show collections                      # List collections
db.students.drop()                    # Drop collection
```

📄 Document Commands

```
// Insert
        db.students.insertOne({ name: "John", age: 22, course: "MERN" })
        db.students.insertMany([{ name: "A" }, { name: "B" }])

// Read
        db.students.find()
        db.students.find({ age: { $gt: 18 } })

// Update
        db.students.updateOne({ name: "John" }, { $set: { age: 23 } })
        db.students.updateMany({}, { $set: { active: true } })

// Delete
        db.students.deleteOne({ name: "John" })
        db.students.deleteMany({ active: false })
```

MongoDB Query Operators

There are many query operators that can be used to compare and reference document fields.

Comparison

The following operators can be used in queries to compare values:

- $eq : Values are equal
- $ne : Values are not equal
- $gt : Value is greater than another value
- $gte : Value is greater than or equal to another value
- $lt : Value is less than another value
- $lte : Value is less than or equal to another value
- $in : Value is matched within an array

$in Syntax :

```
{ field: { $in: [value1, value2, ...] } }
```

📘 Example 1: Match students with course in a list

```
db.students.find({
course: { $in: ["Web", "MERN"] }
})
```

🔷 *This will return all students whose course is either **"Web"** or **"MERN"**.*

📘 Example 2: Match a value **within an array field:**

```
{
name: "Ankit",
skills: ["HTML", "CSS", "JavaScript"]
}
```

To find students who have "CSS" in their skills array:

```
db.students.find({
skills: { $in: ["CSS"] }
})
```

Logical

The following operators can logically compare multiple queries.

- $and : Returns documents where both queries match
- $or : Returns documents where either query matches
- $not : Returns documents where the query does not match

$and **Syntax:**

```
{
  $and: [
    { field1: condition1 },
    { field2: condition2 }
  ]
}
```

Example: Find students with age > 20 and course = "Web"

```
db.students.find({
  $and: [
    { age: { $gt: 20 } },
    { course: "Web" }
  ]
})
```

This will return only those students who are:

- Older than 20
- Enrolled in the "Web" course

Shortcut:

MongoDB treats multiple conditions in a single object as an implicit $and. So, this works the same:

```
db.students.find({
  age: { $gt: 20 },
  course: "Web"
})
```

$or Syntax :

```
{ $or: [ { condition1 }, { condition2 } ] }
```

Example:

Find students who are **younger than 20 OR enrolled in "MERN"**:

```
db.students.find({
  $or: [
    { age: { $lt: 20 } },
    { course: "MERN" }
  ]
})
```

$not – Inverts the Condition :

Example:

Find students whose age is NOT greater than 20:

```
db.students.find({
  age: { $not: { $gt: 20 } }
})
```

🧠 MongoDB Practice Set

✅ Task 1: Basic CRUD

1. Create a database called **school**.
2. Create a collection called **students**.
3. Insert 5 documents with fields: **name, age, course, city**.
4. Find all students from **city "Delhi"**.
5. Update age of student named "**Amit" to 25**.
6. Delete student whose name is **"Ravi"**.

✅ Task 2: Advanced Queries

1. Find students with **age > 20 and course = "Web"**.
2. Find students who are not from **"Delhi"**.
3. Add a new field **isActive: true** to all documents.

✅ Task 3: Array Operations

1. Add a field skills as an array: ["HTML", "CSS"]
2. Find students who know "CSS".
3. Add "JavaScript" to the skills array of one student.

MongoDB Update Operators

- $currentDate: Sets the field value to the current date
- $inc: Increments the field value
- $rename: Renames the field
- $set: Sets the value of a field
- $unset: Removes the field from the document

🔹 Syntax:

```
db.collection.updateOne(
{ /* filter */ },
{ $currentDate: { fieldName: true } }
)
```

Or if you want a timestamp instead of just a date:

```
$currentDate: { fieldName: { $type: "timestamp" } }
```

Example 1: Add a lastUpdated field with current date

```
db.students.updateOne(
{ name: "Amit" },
{ $currentDate: { lastUpdated: true } }
)
```

This will add:

```
"lastUpdated": ISODate("2025-04-09T12:34:56.000Z")
```

Example 2: Add a lastLogin timestamp

```
db.students.updateOne(
 { name: "Ravi" },
 { $currentDate: { lastLogin: { $type: "timestamp" } } }
)
```

Connect Mongodb With Node.Js

Using **Mongoose**

1. **Install Mongoose**

```
npm install mongoose
```

2. **Connect to MongoDB**

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydatabase', {
useNewUrlParser: true,
useUnifiedTopology: true
})
.then(() => console.log("MongoDB connected successfully"))
.catch((err) => console.error("MongoDB connection error:", err));
```

Replace mydatabase with your DB name. Use mongodb+srv://... URI if connecting to MongoDB Atlas.

3. **Create a Schema and Model**

```
const userSchema = new mongoose.Schema({
name: String,
email: String,
age: Number
});

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

// Example: Create a new user
const newUser = new User({ name: "Deepak", email: "deepak@example.com", age: 25 });
newUser.save().then(() => console.log("User saved"));
```

useNewUrlParser: true

Setting useNewUrlParser: true tells Mongoose to use the **new, modern connection string parser**.

Required when using connection strings for things like **MongoDB Atlas** (e.g., with multiple hosts, options, credentials).

useUnifiedTopology: true

Improves the way Mongoose manages **connections, monitoring, and failover**.

configure MongoDB Atlas

Step 1: Create MongoDB Atlas Account

1. Go to https://www.mongodb.com/cloud/atlas
2. Sign up (or log in if you already have an account)

Step 2: Create a Cluster

1. Click on **"Build a Database"**
2. Choose a **free tier** if you're just getting started
3. Select:
   - **Cloud provider** (AWS, GCP, or Azure)
   - **Region** (preferably near your location)
4. Click **"Create Cluster"**

Step 3: Create Database User

1. Go to **Database Access** in the sidebar
2. Click **"Add New Database User"**
3. Set a **username** and **password**
4. Give **Read and Write access to any database**
5. Click **"Add User"**

Step 4: Whitelist Your IP

1. Go to **Network Access** from the sidebar
2. Click **"Add IP Address"**
3. Choose **"Allow Access from Anywhere"** (for dev use) → 0.0.0.0/0
4. Or enter your specific IP address
5. Click **"Confirm"**

Step 5: Create a Database & Collection

1. Go to **Database > Clusters**
2. Click **"Browse Collections"**
3. Click **"Add My Own Data"**
4. Enter:
   - Database name (e.g., myAppDB)
   - Collection name (e.g., users)

Step 6: Connect Your Application

1. Go to **Clusters > Connect > Connect Your Application**
2. Copy the **Connection String** (looks like this):

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

Replace:

- <username> with your DB username
- <password> with your DB password
- <dbname> with your database name

Step 7: Use It in Your Code (Example in Node.js)

```
const mongoose = require('mongoose');

mongoose.connect(

'mongodb+srv://<username>:<password>@cluster0.mongodb.net/myAppDB?retryWrites=true&w=majority',
  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  }
).then(() => {
  console.log("Connected to MongoDB Atlas");
}).catch(err => {
  console.error("Error connecting to MongoDB Atlas", err);
});
```

Step-by-Step Guide to Add MongoDB URL in .env

1. **Install dotenv package**

```
npm install dotenv
```

2. **Create a .env file** in the root of your project:

```
MONGO_URL=mongodb://localhost:27017/mydatabase
```

3. **Update index.js to use .env**

```
require('dotenv').config();
```

Then update your mongoose.connect line like this:

```
mongoose.connect(process.env.MONGO_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log("MongoDB connected successfully"))
.catch((err) => console.error("MongoDB connection error:", err));
```

Final index.js Snippet (first few lines):

```
require('dotenv').config(); // 👈 Load .env variables
const express = require('express');
const app = express();
const cors = require('cors');
const mongoose = require('mongoose');
const PORT = 5000;

app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log("MongoDB connected successfully"))
.catch((err) => console.error("MongoDB connection error:", err));
```

4. **Don’t forget to add .env to .gitignore**

In your .gitignore file, add:

```
.env
```

Modified userSchema with createdAt and updatedAt:

```
const userSchema = new mongoose.Schema({
  name: String
}, {
  timestamps: true // This adds createdAt and updatedAt fields automatically
});
```

Full Updated Section in Your Code:

```
const userSchema = new mongoose.Schema({
  name: String
}, {
  timestamps: true  // Automatically manages createdAt and updatedAt
});
const User = mongoose.model('users', userSchema);
```

Now, every document in the users collection will have:

```
{
  "_id": "...",
  "name": "John Doe",
  "createdAt": "2025-04-10T08:30:00.000Z",
  "updatedAt": "2025-04-10T08:30:00.000Z",
  "__v": 0
}
```