Table of Content

|  |  |  |
| --- | --- | --- |
| S.NO | Topic | SubTopic |
| 1 | Introduction to MongoDB | NoSQL vs RDBMS, Use Cases, MongoDB Overview |
| 2 | MongoDB Atlas Setup | Cloud setup, Atlas cluster, Compass connection |
| 3 | Local Setup & mongosh | Install locally, use mongosh, CLI basics |
| 4 | Create Database & Collections | Create DB/collections using mongosh |
| 5 | Basic CRUD Operations (Insert) | insertOne, insertMany, document structure |
| 6 | Basic CRUD (Read/Query) | find, findOne, filters, operators |
| 7 | Basic CRUD (Update/Delete) | updateOne, deleteMany, $set, $inc |
| 8 | JSON vs BSON | Structure, datatypes, storage representation |
| 9 | MongoDB Compass | GUI interface, monitoring, DB navigation |
| 10 | Schema-less Design Concepts | Flexibility vs structure, flat vs nested docs |
| 11 | Embedding vs Referencing | 1:1, 1:N, N:N, normalization, denormalization |
| 12 | Indexing Basics | Single, compound, multikey indexes |
| 13 | Advanced Index Types | TTL, Text, Hashed, Geospatial, Wildcard indexes |
| 14 | Aggregation Framework – Part 1 | $match, $group, $project, $lookup, etc. |
| 15 | Aggregation Framework – Part 2 | $sort, $skip, $bucket, $facet |
| 16 | Transactions & ACID | Multi-doc transactions, rollback, sessions |
| 17 | Real-world Use Case Practice | Build schema for cart/blog/e-commerce app |
| 18 | Final Review + Quiz | Mini-project review, schema test, MCQs |

Unit 1: Introduction to MongoDB

# 1.1 What is MongoDB?

**MongoDB** is a **NoSQL**, **document-oriented** database. It stores data in **JSON-like** documents called **BSON** (Binary JSON), making it more flexible than traditional relational databases (RDBMS).

# 1.2 Difference Between MongoDB and RDBMS

|  |  |  |
| --- | --- | --- |
| Feature | MongoDB (NoSQL) | RDBMS (MySQL, PostgreSQL) |
| Data Model | Document-oriented (BSON) | Table-based (rows and columns) |
| Schema | Dynamic (schema-less) | Fixed (predefined schema) |
| Joins | Limited joins (embedded docs or refs) | Supports JOIN operations |
| Scalability | Horizontal (via sharding) | Vertical scaling (limited by hardware) |
| Transactions | Supported (multi-doc since v4.0) | Supported natively |

# 1.3 Why Use MongoDB?

* Flexible schema design
* Developer-friendly (JSON-style)
* High performance and scalability
* Ideal for modern web apps (Node.js, MERN stack)

# 1.4 MongoDB Architecture

MongoDB stores data in this structure:

**MongoDB Server**

└── Database

└── Collection

└── Document (key-value pairs)

**Key Terms:**

* **Database**: A container for collections.
* **Collection**: A group of documents (equivalent to tables).
* **Document**: Actual data, stored in BSON format.

*Example Document:*

{

"name": "John Doe",

"age": 22,

"isStudent": true,

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

}

# 1.4 Key Concepts to Remember

|  |  |
| --- | --- |
| Term | Description |
| NoSQL | Non-relational databases (flexible schema) |
| BSON | Binary JSON – stores MongoDB documents internally |
| Document | Primary unit of data in MongoDB (like a row, but flexible) |
| Collection | A group of related documents (like a table) |
| Schema-less | No fixed structure for fields in documents |

# 1.5 Summary

* MongoDB is a schema-less, NoSQL document database.
* It uses BSON (Binary JSON) for storing documents.
* It’s ideal for modern, scalable applications (especially in the MERN stack).
* Data is stored in collections and documents instead of rows and tables.

Unit 2: MongoDB Atlas Setup

# 2.1 Why MongoDB Atlas?

MongoDB Atlas is a **cloud-based database platform** offered by MongoDB Inc. It helps you:

* Host MongoDB without local installation.
* Easily scale, back up, and manage data.
* Share access with collaborators from anywhere.
* Avoid managing server infrastructure.

## Step 1: Create a MongoDB Atlas Account

1. Go to: <https://www.mongodb.com/cloud/atlas/register>
2. Fill in the details or use Google/GitHub to sign up.
3. After login, click **"Build a Database"**.

## Step 2: Create a Free Shared Cluster

1. Choose:
   * **Cloud Provider**: AWS / GCP / Azure
   * **Region**: Choose nearest to your location (e.g., Mumbai for India)
2. Cluster Tier:  
   Choose the free option → M0 Sandbox (Always Free)
3. Name your cluster (e.g., Cluster0) and click **Create Cluster**.

## Step 3: Configure Security

#### 1. Add a Database User

* Username: student
* Password: strongpassword123 (use your own safe password)

Save this information securely.

#### 2. Add IP Address to Access List

* Choose: **Allow access from anywhere**
* This will set: 0.0.0.0/0

In real-world production apps, **use your current IP only** for better security.

## Step 4: Connect to the Cluster

After the cluster is ready:

1. Click **"Connect"** → Choose **"Connect using MongoDB Compass"**.
2. Copy the **Connection String** like:

mongodb+srv://student:strongpassword123@cluster0.mongodb.net/

## Step 5: Install and Use MongoDB Compass

#### Install MongoDB Compass:

Download from: <https://www.mongodb.com/try/download/compass>

Install and open Compass.

#### Connect:

1. Paste the connection URI into Compass.
2. Click **Connect**.

You will see your cluster and databases visually!

# 2.2 Bonus: Modify Connection URI

To connect via terminal (mongosh), use a modified URI like:

mongosh "mongodb+srv://student:<password>@cluster0.mongodb.net/myDatabase"

Replace <password> and myDatabase as needed.

# 2.3 Summary Table

|  |  |
| --- | --- |
| Step | Action |
| Create Atlas Account | Sign up at mongodb.com |
| Build Free Cluster | M0 cluster (AWS/GCP/Azure) |
| Add DB User | Create username + password |
| Whitelist IP | Use 0.0.0.0/0 for demo purposes |
| Get Connection String | Use in Compass or shell |
| Install Compass | GUI client to connect and view documents |
| Create DB/Collection | Add data manually via Compass |

Unit 3 MongoDB Set Up

# 3.1 MongoDB installation in Windows

## Step 1: Download MongoDB Community Server

1. Visit the official MongoDB download page:  
   👉 <https://www.mongodb.com/try/download/community>
2. Choose the following options:
   * **Version**: Latest Stable (e.g., 6.x)
   * **Platform**: Windows
   * **Package**: .msi (Windows Installer)
3. Click **Download** and wait for the file to finish.

## Step 2: Run the MongoDB Installer

1. **Double-click** the .msi file you downloaded.
2. In the **Setup Wizard**, click **Next**.
3. **Choose Setup Type**:
   * Select **Complete** (recommended for beginners).
4. **Service Configuration**:
   * Keep default settings:
     + **Install MongoDB as a Service**
     + Run service as **Network Service User**
5. Optional: You can also install **MongoDB Compass** (GUI tool) from the same wizard.
6. Click **Install** and allow the installer to finish.

## Step 3: Verify MongoDB Installation

After installation, MongoDB binaries are usually located at:

C:\Program Files\MongoDB\Server\<version>\bin

To make the command-line tools accessible from any folder, **add this bin path to the System Environment Variables**:

### Add to PATH:

1. Open Windows Search → type “Environment Variables”.
2. Click “Edit the system environment variables”.
3. Click **Environment Variables** button.
4. Under “System variables”, select Path, then click **Edit**.
5. Click **New**, then add the path:

C:\Program Files\MongoDB\Server\6.0\bin

Adjust version number (6.0) based on what you installed.

1. Click OK → OK → OK to save and exit.

## Step 4: Test Installation (via Terminal)

Open **Command Prompt (cmd)** and type:

mongod --version

If installed correctly, you should see version info like:

db version v6.0.6

To test MongoDB server is running:

mongod

This starts the server. You will see logs like:

Waiting for connections on port 27017

Leave this window open while MongoDB is running.

## Step 5: Open a New Terminal and Use mongosh

MongoDB now uses mongosh (Mongo Shell) for interacting with the database.

Open a new command prompt window and run:

mongosh

You’ll see:

Current Mongosh Log ID: ...

Using MongoDB: 6.0.x

test>

You're now connected to the MongoDB shell and ready to start using it!

## Default Data Storage Location

When MongoDB runs, it stores data at:

C:\data\db

If this folder doesn’t exist, create it manually:

mkdir C:\data\db

MongoDB won’t start unless this folder exists.

## Summary

|  |  |
| --- | --- |
| Step | Description |
| Download Installer | From MongoDB official website |
| Install MongoDB | Complete setup with default service |
| Add to PATH | So you can run commands from anywhere in terminal |
| Start MongoDB Server | mongod |
| Connect with Shell | mongosh |
| Test Insert & Query | Use basic CRUD in MongoDB shell |

# **3.2 Mongoosh**

**Goal**: Set up MongoDB on your local system and interact with it using the **MongoDB Shell (mongosh)**.

## What is mongosh?

* mongosh is the **official MongoDB shell**, used to interact with your MongoDB database from the command line.
* It supports:
  + CRUD operations
  + Admin tasks
  + Writing JavaScript in shell
  + Connecting to local or remote MongoDB instances

## Step-by-Step Installation (Windows)

If you’ve already installed MongoDB (from Unit 1), skip to Step 5.

### Step 1: Download MongoDB Community Edition

* Go to: <https://www.mongodb.com/try/download/community>
* Choose:
  + **Platform**: Windows
  + **Package**: .msi
* Click **Download** and install it.

### Step 2: Install MongoDB

1. Launch the .msi file.
2. Select **Complete Setup**.
3. Check the box to install **MongoDB as a service**.
4. (Optional) Choose to install **MongoDB Compass**.
5. Complete the installation.

### Step 3: Set MongoDB in PATH

To use MongoDB from any terminal window:

1. Go to:  
   C:\Program Files\MongoDB\Server\<version>\bin
2. Copy the path and add it to **System Environment Variables**:
   * Search: “Edit the system environment variables”
   * Go to: Environment Variables → Path → Edit → New → Paste the path
   * Click OK and apply changes

### Step 4: Create MongoDB Data Directory

MongoDB stores its data in a folder. You need to create it:

mkdir C:\data\db

This is the default directory for MongoDB.

### Step 5: Start MongoDB Server

1. Open **Command Prompt** and run:

mongod

If successful, you’ll see logs saying:

Waiting for connections on port 27017

Your database server is now running.

Keep this terminal open! It’s running your database server.

### Step 6: Open a New Terminal and Launch mongosh

Now, open **another Command Prompt** and run:

mongosh

This connects to your local MongoDB server.

## Basic Commands to Practice

Once inside mongosh, try the following:

### 1. Create/Select a Database

use bookstore

If the database doesn’t exist, MongoDB creates it when you first insert data.

### 2. Insert a Document

db.books.insertOne({

title: "Clean Code",

author: "Robert Martin",

pages: 464

})

### 3. View Documents

db.books.find()

## Useful mongosh Commands

|  |  |
| --- | --- |
| Command | Description |
| show dbs | List all databases |
| use <dbName> | Switch to a database |
| show collections | List collections in the current DB |
| db.collection.insertOne() | Insert a single document |
| db.collection.find() | View all documents |
| db.collection.drop() | Delete a collection |
| exit | Exit the shell |

## Summary Table

|  |  |
| --- | --- |
| Task | Command / Action |
| Start MongoDB Server | mongod |
| Open MongoDB Shell | mongosh |
| Switch DB | use myDB |
| Insert Data | db.myCollection.insertOne({}) |
| View Data | db.myCollection.find() |
| Stop Server | Close the terminal where mongod is running |

Unit 4: Creating Databases & Collections

# **4.1 Databases & Collections**

**Goal**: Learn how to **create databases and collections** in MongoDB using the **MongoDB shell (mongosh)**, and understand naming conventions and how collections are created dynamically.

### Key Concepts

* **Database**: A container for collections.
* **Collection**: A group of MongoDB documents, like a table in RDBMS.
* **Document**: A single record (in JSON/BSON format).
* MongoDB is **schema-less**, meaning:
  + You don’t define tables and columns ahead of time.
  + Fields can vary from document to document in the same collection.

### MongoDB Architecture Recap

MongoDB ➝ Database ➝ Collection ➝ Document

# 4.2 Creating a Database

In MongoDB, **you switch to a database first**. If it doesn't exist, it will be created when you store data in it.

use studentDB

This switches to the database studentDB.  
Nothing is created yet — it exists **only in memory** until data is inserted.

# 4.3 Creating a Collection

### Option A: Implicit Creation (Recommended)

MongoDB automatically creates a collection when you insert your first document.

db.students.insertOne({

name: "Ravi",

department: "Mechanical",

year: 2

})

→ This automatically creates the students collection inside studentDB.

### Option B: Explicit Creation (Rarely used)

db.createCollection("faculty")

✔️ This manually creates a collection.

### Verifying the Database & Collections

show dbs // Lists all databases

show collections // Lists all collections in the current DB

### Practice Example

use collegeDB

db.courses.insertMany([

{ name: "Data Structures", credits: 4 },

{ name: "Operating Systems", credits: 3 }

])

db.courses.find()

### Output:

[

{ \_id: ObjectId("..."), name: "Data Structures", credits: 4 },

{ \_id: ObjectId("..."), name: "Operating Systems", credits: 3 }

]

### Collection Naming Conventions

* Use **lowercase** letters
* Use **plural** nouns (e.g., students, orders)
* Avoid special characters or spaces
* Use camelCase or underscores (e.g., userProfiles, order\_items)

✔️ Good: products, courseList, user\_accounts  
❌ Bad: 123data, My Table, Product$Info

## 4.4 Summary Table

|  |  |
| --- | --- |
| Action | Command |
| Switch to a DB | use dbName |
| Insert Document | db.collection.insertOne({}) |
| Create Collection | db.createCollection("name") |
| List Collections | show collections |
| List Databases | show dbs |
| View Collection Documents | db.collection.find() |

Unit 5: Basic CRUD (Insert)

# 5.1 CRUD Overview

CRUD stands for:

* **C**reate → insertOne(), insertMany()
* **R**ead → find(), findOne()
* **U**pdate → updateOne(), updateMany()
* **D**elete → deleteOne(), deleteMany()

This unit focuses on the **Create** operations.

# 5.2 MongoDB Document Structure

A **document** in MongoDB is a JSON-like structure (actually stored as BSON) with key-value pairs.

Example:

{

"name": "Anjali",

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

"age": 21

}

Note: MongoDB automatically adds an \_id field to each document if not provided.

## 1. insertOne() – Insert a Single Document

**Syntax**:

db.collection.insertOne({ key1: value1, key2: value2 })

**Example**:

use studentDB

db.students.insertOne({

name: "Karan",

department: "ECE",

year: 3

})

Output:

{

acknowledged: true,

insertedId: ObjectId("...")

}

## 2. insertMany() – Insert Multiple Documents

**Syntax**:

db.collection.insertMany([

{ key1: value1, key2: value2 },

{ key1: value1, key2: value2 },

])

**Example**:

db.students.insertMany([

{ name: "Deepa", department: "CSE", year: 2 },

{ name: "Arun", department: "MECH", year: 4 }

])

Output:

{

acknowledged: true,

insertedIds: {

"0": ObjectId("..."),

"1": ObjectId("...")

}

}

# 5.3 Nested Documents

MongoDB supports **nested documents** (documents inside documents):

db.students.insertOne({

name: "Nisha",

contact: {

email: "nisha@gmail.com",

phone: "9876543210"

},

address: {

city: "Chennai",

pincode: 600001

}

})

This structure is flexible and doesn't need a fixed schema.

# 5.4 Data Types in MongoDB Documents

|  |  |
| --- | --- |
| Type | Example |
| String | "name": "Ravi" |
| Number | "age": 22 |
| Boolean | "isActive": true |
| Array | "skills": ["Node", "MongoDB"] |
| Object | "profile": { "email": "x@y.com" } |
| Date | "joined": new Date() |

# 5.5 Practice Task

1. Use/Create a new database:

use libraryDB

1. Insert a single book:

db.books.insertOne({

title: "MongoDB for Beginners",

author: "Jane Doe",

year: 2023,

price: 499

})

1. Insert multiple books:

db.books.insertMany([

{

title: "Learn Node.js",

author: "John Smith",

year: 2021,

price: 699

},

{

title: "JavaScript Essentials",

author: "Anita Sharma",

year: 2022,

price: 399

}

])

1. View inserted documents:

db.books.find()

# 5.6 Common Errors to Avoid

|  |  |
| --- | --- |
| Problem | Fix |
| Forgetting {} around insert data | Always use object format: { key: value } |
| Typo in collection name | Check spelling – MongoDB will create a new one if mistyped |
| Using invalid key names | Avoid $, . in field names unless required |

# 5.7 Summary Table

|  |  |
| --- | --- |
| Task | Command Example |
| Insert one document | db.students.insertOne({ name: "Raj", age: 20 }) |
| Insert many documents | db.students.insertMany([{...}, {...}]) |
| View all documents | db.students.find() |
| Use nested objects | { name: "A", contact: { phone: "123" } } |

Unit 6: Basic Read/Query Operations

# 6.1 Key Concepts

* **find()** → Returns a cursor to all matching documents (can return many).
* **findOne()** → Returns the first matching document only.
* **Query Filters** → Allow you to search by fields or conditions.
* **Comparison Operators** → $gt, $lt, $eq, $ne, $in, etc.

## 1. find() – Retrieve All or Filtered Documents

**Syntax**:

db.collection.find({ query })

### 🔸 Example: Get All Documents

db.students.find()

Retrieves all documents in the students collection.

## 2. findOne() – Retrieve First Match Only

db.students.findOne({ department: "CSE" })

Returns the first document where department is "CSE".

## 3. Query by Field Match

db.students.find({ year: 3 })

Get all students in 3rd year.

## 4. Using Comparison Operators

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| $eq | Equal to | { year: { $eq: 2 } } |
| $ne | Not equal to | { department: { $ne: "ECE" } } |
| $gt | Greater than | { year: { $gt: 2 } } |
| $lt | Less than | { year: { $lt: 4 } } |
| $gte | Greater or Equal | { year: { $gte: 3 } } |
| $lte | Less or Equal | { year: { $lte: 2 } } |
| $in | In array | { department: { $in: ["CSE", "IT"] } } |
| $nin | Not in array | { year: { $nin: [1, 2] } } |

## 5. Project Specific Fields (using projection)

db.students.find({ year: 3 }, { name: 1, department: 1, \_id: 0 })

Returns only the name and department fields for year 3 students (excludes \_id).

## 6. Querying Nested Fields

db.students.find({ "contact.phone": "9876543210" })

Match nested field values.

## 7. Query with Multiple Conditions (Logical AND)

db.students.find({ department: "CSE", year: 2 })

Default behavior is logical **AND**.

## 8. Logical Operators

|  |  |
| --- | --- |
| Operator | Usage Example |
| $or | { $or: [{ year: 2 }, { year: 3 }] } |
| $and | { $and: [{ department: "CSE" }, { year: 3 }] } |
| $not | { year: { $not: { $gt: 3 } } } |
| $nor | { $nor: [{ year: 1 }, { department: "EEE" }] } |

# 6.2 Example Data

db.students.insertMany([

{ name: "Aarav", year: 1, department: "CSE" },

{ name: "Bhavana", year: 3, department: "ECE" },

{ name: "Chirag", year: 2, department: "MECH" },

{ name: "Disha", year: 4, department: "CSE" }

])

## Practice Queries

1. Get all CSE students:

db.students.find({ department: "CSE" })

1. Get students in year 3 or 4:

db.students.find({ year: { $in: [3, 4] } })

1. Get students not in MECH department:

db.students.find({ department: { $ne: "MECH" } })

1. Get only names and years of students:

db.students.find({}, { name: 1, year: 1, \_id: 0 })

# 6.3 Summary Table

|  |  |
| --- | --- |
| Task | Example Query |
| Find all | db.students.find() |
| Find with filter | db.students.find({ year: 3 }) |
| Use comparison operators | db.students.find({ year: { $gt: 2 } }) |
| Logical OR | db.students.find({ $or: [{ year: 2 }, ...] }) |
| Project specific fields | { name: 1, \_id: 0 } |
| Nested fields | "contact.phone": "..." |

Unit 7 : Basic CRUD – Update in MongoDB

# 7.1 What is an Update?

An **update** modifies fields in existing documents within a collection.

MongoDB provides two main update methods:

* updateOne() – Updates the **first matching** document.
* updateMany() – Updates **all matching** documents.

## 1 . Syntax of updateOne()

db.collection.updateOne(

{ <filter> },

{ <update operation> }

)

### Example:

db.students.updateOne(

{ name: "Aarav" },

{ $set: { year: 2 } }

)

This updates the year field to 2 for the first student named **Aarav**.

## 2. Syntax of updateMany()

db.collection.updateMany(

{ <filter> },

{ <update operation> }

)

### Example:

db.students.updateMany(

{ department: "CSE" },

{ $inc: { year: 1 } }

)

This **increments** the year by 1 for all students in the CSE department.

## 3. Common Update Operators

|  |  |  |
| --- | --- | --- |
| Operator | Purpose | Example |
| $set | Set a new value to a field | { $set: { age: 21 } } |
| $inc | Increment or decrement a number | { $inc: { marks: 5 } } |
| $unset | Remove a field from a document | { $unset: { mobile: "" } } |
| $rename | Rename a field | { $rename: { fullName: "name" }} |

# 7.2 Example Data

You can insert the following data to try updates:

db.students.insertMany([

{ name: "Aarav", year: 1, department: "CSE" },

{ name: "Bhavana", year: 3, department: "ECE" },

{ name: "Chirag", year: 2, department: "MECH" },

{ name: "Disha", year: 4, department: "CSE" },

{ name: "Elina", year: 1, department: "EEE" }

])

## Real-World Examples

### Set a New Field

db.students.updateOne(

{ name: "Disha" },

{ $set: { passed: true } }

)

### Increase Year by 1 for All CSE Students

db.students.updateMany(

{ department: "CSE" },

{ $inc: { year: 1 } }

)

### Remove a Field

db.students.updateOne(

{ name: "Elina" },

{ $unset: { department: "" } }

)

### Rename a Field

db.students.updateMany(

{},

{ $rename: { year: "semester" } }

)

# 7.3 Advance Update

## 1. ****Upsert (Update or Insert)****

When updating a document, if it doesn’t exist, MongoDB can **insert it instead** using the upsert: true option.

**Syntax**:

db.students.updateOne(

{ name: "Farhan" },

{ $set: { year: 1, department: "CSE" } },

{ upsert: true }

)

If "Farhan" doesn't exist, it will be created.

**Use Case**: Useful for idempotent operations like syncing data.

## 2. ****Replace a Document (****replaceOne****)****

Replaces the **entire document**, not just a field.

**Syntax**:

db.students.replaceOne(

{ name: "Disha" },

{ name: "Disha", year: 2, department: "ECE" }

)

Removes all other fields in the old document.

**Use Case**: When updating an outdated document schema entirely.

## 3. ****Return Updated Document (****findOneAndUpdate****)****

To return the **document after update**, you can use this method.

**Syntax**:

db.students.findOneAndUpdate(

{ name: "Aarav" },

{ $set: { department: "IT" } },

{ returnDocument: "after" } // or returnNewDocument: true in some drivers

)

Returns the document **after** the update is applied.

## 4. ****Multi-condition Updates****

Use **compound filters** for more precise updates.

**Example**:

db.students.updateMany(

{ year: { $lt: 3 }, department: "CSE" },

{ $inc: { year: 1 } }

)

Promotes only CSE students in years 1 and 2.

## 5. ****Use**** writeConcern ****(Optional Advanced)****

For production apps, especially with replicas, mention that updates can be made **reliable** using write concern.

**Example**:

db.students.updateOne(

{ name: "Aarav" },

{ $set: { year: 4 } },

{ writeConcern: { w: "majority", wtimeout: 5000 } }

)

# 7.3 Common Mistakes

|  |  |
| --- | --- |
| Mistake | Fix |
| Using update without $set | Always wrap new data in $set, $inc, etc. |
| Forgetting to use a filter | Always specify {} or a condition to avoid errors. |
| Confusing updateOne with updateMany | Use updateOne for a single update. |

# 7.4 Summary Table

|  |  |
| --- | --- |
| Method | Use Case |
| updateOne() | Update first matching document |
| updateMany() | Update multiple documents |
| $set | Set or overwrite values |
| $inc | Add/subtract numeric fields |
| $unset | Delete a field from document |
| $rename | Rename field in document |