**Unit-4**

**MongoDB**

**MongoDB** is an open-source NoSQL database written in C++ language. It uses JSON like documents with optional schemas.

• It provides easy scalability and is a cross-platform, document-oriented database. • MongoDB works on the concept of Collection and Document.

• It combines the ability to scale out with features such as secondary indexes, range queries, sorting, aggregations, and geospatial indexes.

• MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

**Features of MongoDB –**

• Indexing: It supports generic secondary indexes and provides unique, compound, geospatial, and full-text indexing capabilities as well.

• Aggregation: It provides an aggregation framework based on the concept of data processing pipelines.

• Special collection and index types: It supports time-to-live (TTL) collections for data that should expire at a certain time

• File storage: It supports an easy-to-use protocol for storing large files and file metadata.

• Sharding: Sharding is the process of splitting data up across machines.

**\*\*\* How to:**

a. Add/ Insert data in MongoDB?

b. Update a Document?

c. Delete a Document?

d. Perform queries in MongoDB?

**Ans** **a.** The basic method for adding data to MongoDB is “inserts”. To insert a single document, use the collection’s insertOne method:

> db.books.insertOne({"title" : "Start With Why"})

For inserting multiple documents into a collection, we use insertMany. This method enables passing an array of documents to the database.

**b.** Once a document is stored in the database, it can be changed using one of several update methods: updateOne, updateMany, and replaceOne. updateOne and updateMany each takes a filter document as their first parameter and a modifier document, which describes changes to make, as the second parameter. replaceOne also takes a filter as the first parameter, but as the second parameter replaceOne expects a document with which it will replace the document matching the filter. For example, in order to replace a document:

{ "\_id" : ObjectId("4b2b9f67a1f631733d917a7a"),

"name" : "alice",

"friends" : 24,

"enemies" : 2

}

**c.** The CRUD API in MongoDB provides deleteOne and deleteMany for this purpose. Both of these methods take a filter document as their first parameter. The filter specifies a set of criteria to match against in removing documents. For example:

> db.books.deleteOne({"\_id" : 3})

**d.** The find method is used to perform queries in MongoDB. Querying returns a subset of documents in a collection, from no documents at all to the entire collection. Which documents get returned is determined by the first argument to find, which is a document specifying the query criteria.

Example:

> db.users.find({"age" : 24}

**SQL vs NoSQL: Key Differences**

SQL and NoSQL are two types of databases used for different purposes. Here’s a breakdown of their key **concepts, differences, and use cases**.

**1. What is SQL? (Relational Databases)**

SQL (**Structured Query Language**) databases store **data in tables** with predefined schemas.

✅ **Characteristics:**

* **Structured & organized** (tables, rows, columns)
* Uses **SQL queries** for data manipulation (SELECT, INSERT, UPDATE, etc.)
* **ACID-compliant** (Atomicity, Consistency, Isolation, Durability)
* Good for **structured data** and complex relationships.

✅ **Examples of SQL Databases:**

* MySQL
* PostgreSQL
* Microsoft SQL Server
* Oracle Database

**📌 Example SQL Table (Users)**

| **ID** | **Name** | **Age** | **Email** |
| --- | --- | --- | --- |
| 1 | John | 28 | john@mail.com |
| 2 | Sarah | 24 | sarah@mail.com |

👉 **Query to get users older than 25:**

sql

CopyEdit

SELECT \* FROM Users WHERE Age > 25;

**2. What is NoSQL? (Non-Relational Databases)**

NoSQL databases store data in a **flexible format** like JSON, key-value, or graphs.

✅ **Characteristics:**

* **Schema-less & flexible** (no predefined structure)
* Uses **different data models** (document, key-value, column-family, graph)
* **Scales horizontally** (distributed databases)
* Good for **large-scale, real-time applications**.

✅ **Types of NoSQL Databases:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Document** | Stores data as JSON-like documents | MongoDB, CouchDB |
| **Key-Value** | Simple key-value pairs | Redis, DynamoDB |
| **Column-Family** | Stores data in columns, not rows | Cassandra, HBase |
| **Graph** | Stores relationships in nodes/edges | Neo4j, ArangoDB |

**📌 Example NoSQL (MongoDB Document)**

json

CopyEdit

{

"\_id": 1,

"name": "John",

"age": 28,

"email": "john@mail.com"

}

👉 **Query to get users older than 25 in MongoDB:**

js

CopyEdit

db.users.find({ age: { $gt: 25 } });

**3. Key Differences: SQL vs NoSQL**

| **Feature** | **SQL (Relational)** | **NoSQL (Non-Relational)** |
| --- | --- | --- |
| **Data Model** | Tables (Rows & Columns) | Documents, Key-Value, Graph, Columnar |
| **Schema** | Fixed Schema (Structured) | Dynamic Schema (Flexible) |
| **Query Language** | SQL (SELECT, JOIN, etc.) | NoSQL queries (MongoDB uses JSON-based queries) |
| **Scalability** | **Vertical Scaling** (More CPU/RAM) | **Horizontal Scaling** (Distributed systems) |
| **Transactions** | ACID-compliant (strong consistency) | BASE (Eventual Consistency) |
| **Best For** | Complex relationships, structured data | Big Data, Real-time applications |

**4. When to Use SQL vs NoSQL?**

| **Use Case** | **SQL** | **NoSQL** |
| --- | --- | --- |
| Banking & Financial Apps | ✅ Yes | ❌ No (Requires ACID) |
| Real-Time Analytics | ❌ No | ✅ Yes |
| E-commerce & Inventory | ✅ Yes | ✅ Yes (Depends on structure) |
| Social Networks | ❌ No | ✅ Yes (Graph databases) |
| Content Management | ❌ No | ✅ Yes (Flexible structure) |

**📌 Final Thoughts**

✅ **SQL** is best for structured data, transactions, and relational integrity.  
✅ **NoSQL** is best for unstructured data, scalability, and flexibility.  
✅ **Hybrid Approach**: Some systems use **both** SQL & NoSQL (e.g., MySQL + MongoDB).

**Create and Manage MongoDB**

**MongoDB is a NoSQL database that stores data in a flexible JSON-like format. Here's a step-by-step guide on creating and managing a MongoDB database.**

**1. Install & Setup MongoDB**

**🖥️ Install MongoDB**

* **Windows: Download from MongoDB Official Site and install.**
* **Mac (Homebrew):**

**sh**

**brew tap mongodb/brew**

**brew install mongodb-community@6.0**

* **Linux (Ubuntu/Debian):**

**sh**

**sudo apt update**

**sudo apt install -y mongodb**

**Start MongoDB Server**

**Start MongoDB in the background:**

**sh**

**mongod --dbpath /data/db**

**👉 Default MongoDB runs on port 27017.**

**2. Connect to MongoDB**

**Open the MongoDB shell by running:**

**sh**

**mongosh**

**You should see:**

**shell**

**> MongoDB shell version v6.0**

**> Connecting to: mongodb://localhost:27017/**

**3. Create a Database**

**To create a database (or switch to it):**

**sh**

**use myDatabase**

**📌 If the database does not exist, MongoDB creates it automatically when data is added.**

**4. Create and Manage Collections**

**MongoDB stores data in collections (like tables in SQL).**

**Create a Collection**

**sh**

**db.createCollection("users")**

**OR insert a document (auto-creates the collection):**

**sh**

**db.users.insertOne({ name: "John", age: 30, email: "john@mail.com" })**

**5. Insert Data**

**Insert One Document**

**sh**

**db.users.insertOne({ name: "Alice", age: 25, email: "alice@mail.com" })**

**Insert Multiple Documents**

**sh**

**db.users.insertMany([**

**{ name: "Bob", age: 28 },**

**{ name: "Charlie", age: 22 }**

**])**

**6. Read Data (Find Documents)**

**Find All Documents**

**sh**

**db.users.find()**

**Find Specific Document**

**sh**

**db.users.find({ name: "Alice" })**

**Find Users Older Than 25**

**sh**

**db.users.find({ age: { $gt: 25 } })**

**7. Update Documents**

**Update One Document**

**sh**

**db.users.updateOne(**

**{ name: "Alice" },**

**{ $set: { age: 26 } }**

**)**

**Update Multiple Documents**

**sh**

**db.users.updateMany(**

**{ age: { $lt: 30 } },**

**{ $set: { status: "young" } }**

**)**

**8. Delete Documents**

**Delete One Document**

**sh**

**db.users.deleteOne({ name: "Alice" })**

**Delete Multiple Documents**

**sh**

**db.users.deleteMany({ age: { $lt: 25 } })**

**9. Manage Databases & Collections**

**List All Databases**

**sh**

**show dbs**

**List All Collections**

**sh**

**show collections**

**Delete a Collection**

**sh**

**CopyEdit**

**db.users.drop()**

**Delete a Database**

**sh**

**use myDatabase**

**db.dropDatabase()**

**10. Connect MongoDB to Node.js**

**📌 Install MongoDB Node.js driver:**

**sh**

**npm install mongodb**

**📌 Create server.js:**

**js**

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

**const url = "mongodb://localhost:27017";**

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

**async function run() {**

**try {**

**await client.connect();**

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

**const db = client.db("myDatabase");**

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

**await users.insertOne({ name: "David", age: 35 });**

**const result = await users.find().toArray();**

**console.log(result);**

**} finally {**

**await client.close();**

**}**

**}**

**run().catch(console.error);**

**✅ Run it with:**

**sh**

**node server.js**

**📌 Summary**

| **Task** | **Command** |
| --- | --- |
| **Start MongoDB** | **mongod --dbpath /data/db** |
| **Connect to Mongo Shell** | **Mongosh** |
| **Create Database** | **use myDatabase** |
| **Create Collection** | **db.createCollection("users")** |
| **Insert Data** | **db.users.insertOne({ name: "John" })** |
| **Find Data** | **db.users.find()** |
| **Update Data** | **db.users.updateOne({ name: "John" }, { $set: { age: 31 } })** |
| **Delete Data** | **db.users.deleteOne({ name: "John" })** |
| **List Databases** | **show dbs** |
| **List Collections** | **show collections** |
| **Drop Collection** | **db.users.drop()** |
| **Drop Database** | **db.dropDatabase()** |

**Data Migration into MongoDB**

Migrating data into **MongoDB** can be done using various methods depending on the source. Here’s a step-by-step guide to migrating data from different sources.

**1️⃣ Migration from JSON/CSV Files**

**📝 Import JSON Data**

If you have data in **JSON format**, use mongoimport:

sh

mongoimport --db myDatabase --collection users --file users.json --jsonArray

✅ Example users.json:

json

[

{ "name": "Alice", "age": 25, "email": "alice@mail.com" },

{ "name": "Bob", "age": 28, "email": "bob@mail.com" }

]

**📄 Import CSV Data**

For **CSV files**, use:

sh

mongoimport --db myDatabase --collection users --type csv --file users.csv --headerline

✅ Example users.csv:

graphql

name,age,email

Alice,25,alice@mail.com

Bob,28,bob@mail.com

**2️⃣ Migration from SQL Databases (MySQL, PostgreSQL)**

Since **SQL and NoSQL** have different structures, data needs to be transformed.

**🔁 Steps:**

1. **Export SQL data as JSON or CSV**
2. **Transform data to fit MongoDB schema**
3. **Use mongoimport or a Node.js script**

**📌 Example: Export MySQL Data**

Run this query in MySQL to export as CSV:

sql

SELECT \* FROM users INTO OUTFILE '/path/to/users.csv'

FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n';

Then, import into MongoDB using:

sh

mongoimport --db myDatabase --collection users --type csv --file /path/to/users.csv --headerline

**🔥 Alternative: Using Node.js for Migration**

Install dependencies:

sh

npm install mysql mongodb

Then, create migrate.js:

js

const mysql = require("mysql");

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

// MySQL Connection

const sqlDb = mysql.createConnection({

host: "localhost",

user: "root",

password: "password",

database: "mySQLDatabase"

});

// MongoDB Connection

const mongoClient = new MongoClient("mongodb://localhost:27017");

async function migrateData() {

try {

await mongoClient.connect();

console.log("Connected to MongoDB");

const mongoDb = mongoClient.db("myDatabase");

const usersCollection = mongoDb.collection("users");

sqlDb.query("SELECT \* FROM users", async (err, result) => {

if (err) throw err;

const users = result.map(user => ({

name: user.name,

age: user.age,

email: user.email

}));

await usersCollection.insertMany(users);

console.log("Migration complete!");

mongoClient.close();

});

} catch (error) {

console.error("Error migrating data:", error);

}

}

migrateData();

Run the script:

sh

node migrate.js

✅ This script **fetches data from MySQL** and **inserts it into MongoDB**.

**3️⃣ Migration from Firebase to MongoDB**

Use the Firebase Admin SDK to export data.

**📌 Install Firebase SDK**

sh

npm install firebase-admin mongodb

**📌 Export Data from Firebase and Import into MongoDB**

js

const admin = require("firebase-admin");

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

// Firebase setup

admin.initializeApp({

credential: admin.credential.cert(require("./firebase-key.json")),

databaseURL: "https://your-project.firebaseio.com"

});

// MongoDB setup

const mongoClient = new MongoClient("mongodb://localhost:27017");

async function migrateData() {

try {

await mongoClient.connect();

console.log("Connected to MongoDB");

const db = mongoClient.db("myDatabase");

const usersCollection = db.collection("users");

const snapshot = await admin.firestore().collection("users").get();

const users = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));

await usersCollection.insertMany(users);

console.log("Firebase data migrated to MongoDB!");

mongoClient.close();

} catch (error) {

console.error("Migration error:", error);

}

}

migrateData();

Run:

sh

node firebase-migrate.js

✅ This script **extracts Firebase Firestore data** and **moves it into MongoDB**.

**4️⃣ Migration from Excel to MongoDB**

**📌 Steps:**

1. **Convert Excel to CSV/JSON** (using Python or Excel)
2. **Import into MongoDB using mongoimport**
3. **Or use Node.js to read and insert data**

**🔥 Using Node.js to Import Excel**

Install dependencies:

sh

npm install xlsx mongodb

Create excel-to-mongo.js:

js

const xlsx = require("xlsx");

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

const workbook = xlsx.readFile("users.xlsx");

const sheetName = workbook.SheetNames[0];

const users = xlsx.utils.sheet\_to\_json(workbook.Sheets[sheetName]);

const mongoClient = new MongoClient("mongodb://localhost:27017");

async function migrateData() {

try {

await mongoClient.connect();

console.log("Connected to MongoDB");

const db = mongoClient.db("myDatabase");

await db.collection("users").insertMany(users);

console.log("Excel data imported!");

mongoClient.close();

} catch (error) {

console.error("Migration error:", error);

}

}

migrateData();

Run:

sh

node excel-to-mongo.js

✅ **Reads Excel file** and **imports data into MongoDB**.

**Summary of Migration Methods**

| **Source** | **Method** |
| --- | --- |
| **JSON** | mongoimport –jsonArray |
| **CSV** | mongoimport --type csv –headerline |
| **MySQL/PostgreSQL** | Export as JSON/CSV, use mongoimport or Node.js |
| **Firebase Firestore** | Use Firebase Admin SDK & Node.js |
| **Excel (XLSX)** | Convert to CSV/JSON or use xlsx + Node.js |

**🚀 Key Takeaways**

✅ mongoimport is the easiest way for JSON/CSV files.  
✅ For **SQL to NoSQL**, transform data before inserting.  
✅ Use **Node.js scripts** for complex migrations.  
✅ Always **validate and clean data** before importing.

**MongoDB with Node.js**

MongoDB is commonly used with **Node.js** to create dynamic and scalable applications. This guide covers **installation, connection, CRUD operations, and Express.js integration**.

**1️⃣ Install MongoDB and Node.js**

**📌 Install MongoDB**

* **Windows**: [Download MongoDB](https://www.mongodb.com/try/download/community) and install it.
* **Mac (Homebrew)**:

sh

brew tap mongodb/brew

brew install mongodb-community@6.0

* **Linux (Ubuntu/Debian)**:

sh

sudo apt update

sudo apt install -y mongodb

**📌 Start MongoDB**

Run MongoDB in the background:

sh

mongod --dbpath /data/db

**2️⃣ Install MongoDB Driver in Node.js**

Create a project and install dependencies:

sh

mkdir mongo-node-app

cd mongo-node-app

npm init -y

npm install mongodb

**3️⃣ Connect Node.js to MongoDB**

Create a file **db.js**:

js

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

const url = "mongodb://localhost:27017";

const client = new MongoClient(url);

async function connectDB() {

try {

await client.connect();

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

return client.db("myDatabase"); // Database name

} catch (error) {

console.error("❌ Connection failed:", error);

}

}

module.exports = connectDB;

Run:

sh

node db.js

**4️⃣ CRUD Operations in MongoDB**

Create a file **app.js**:

js

const connectDB = require("./db");

async function main() {

const db = await connectDB();

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

// Insert Data

await users.insertOne({ name: "Alice", age: 25, email: "alice@mail.com" });

// Read Data

const result = await users.find().toArray();

console.log("Users:", result);

// Update Data

await users.updateOne({ name: "Alice" }, { $set: { age: 26 } });

// Delete Data

await users.deleteOne({ name: "Alice" });

console.log("Operations completed!");

}

main();

Run:

sh

node app.js

✅ **CRUD operations** (Insert, Read, Update, Delete) are now working!

**5️⃣ MongoDB with Express.js API**

**📌 Install Express.js**

sh

npm install express body-parser cors mongodb

**📌 Create server.js**

js

const express = require("express");

const connectDB = require("./db");

const app = express();

app.use(express.json());

let db;

(async () => {

db = await connectDB();

})();

// Get All Users

app.get("/users", async (req, res) => {

const users = await db.collection("users").find().toArray();

res.json(users);

});

// Create User

app.post("/users", async (req, res) => {

await db.collection("users").insertOne(req.body);

res.send("User added!");

});

// Update User

app.put("/users/:name", async (req, res) => {

await db.collection("users").updateOne(

{ name: req.params.name },

{ $set: req.body }

);

res.send("User updated!");

});

// Delete User

app.delete("/users/:name", async (req, res) => {

await db.collection("users").deleteOne({ name: req.params.name });

res.send("User deleted!");

});

app.listen(3000, () => console.log("🚀 Server running on port 3000"));

**Run the API**

sh

node server.js

✅ Open **Postman or a browser** to test:

* **GET Users:** http://localhost:3000/users
* **POST User:** Send JSON { "name": "Bob", "age": 30 } to http://localhost:3000/users
* **PUT User:** Send { "age": 32 } to http://localhost:3000/users/Bob
* **DELETE User:** http://localhost:3000/users/Bob

**🔥 Summary**

| **Task** | **Code/Command** |
| --- | --- |
| **Start MongoDB** | mongod --dbpath /data/db |
| **Connect Node.js to MongoDB** | mongodb.MongoClient.connect() |
| **Insert Data** | collection.insertOne({ name: "Alice" }) |
| **Read Data** | collection.find().toArray() |
| **Update Data** | collection.updateOne({ name: "Alice" }, { $set: { age: 26 } }) |
| **Delete Data** | collection.deleteOne({ name: "Alice" }) |
| **Run Express API** | node server.js |

**MongoDB Services**

MongoDB offers a range of **services** to help developers build, scale, and manage databases efficiently. Below are the key **services offered by MongoDB**:

**1️⃣ MongoDB Atlas (Cloud Database)**

✅ **Fully managed cloud-based MongoDB service**  
✅ Available on **AWS, Azure, and Google Cloud**  
✅ Features:

* Automatic scaling
* Data backup & recovery
* Multi-region deployment
* Built-in security & monitoring  
  ✅ **Best for:** Cloud-based applications and serverless deployments.

**2️⃣ MongoDB Enterprise Edition**

✅ **Self-hosted version with advanced features**  
✅ Includes:

* **Advanced security** (LDAP, Kerberos, Encryption)
* **Performance monitoring**
* **Automated backups**  
  ✅ **Best for:** Large-scale businesses requiring extra security and compliance.

**3️⃣ MongoDB Community Edition**

✅ **Free & open-source version** of MongoDB  
✅ Ideal for **small projects and local development**  
✅ Can be **self-hosted** on any server  
✅ Includes:

* **Basic CRUD operations**
* **Replication & sharding**
* **Indexes & aggregation**

**4️⃣ MongoDB Realm (Mobile Database)**

✅ **Offline-first database** for mobile & IoT apps  
✅ Works with **iOS, Android, and Web**  
✅ **Syncs automatically** with MongoDB Atlas  
✅ Supports **GraphQL & REST APIs**

**5️⃣ MongoDB Charts**

✅ **Visualize MongoDB data** in real-time  
✅ Create **interactive dashboards** without SQL  
✅ Supports **drag-and-drop** analytics  
✅ Works directly with **MongoDB Atlas**

**6️⃣ MongoDB Compass**

✅ **GUI tool** for managing MongoDB databases  
✅ Features:

* Visualize & edit data
* Build aggregation queries
* Monitor performance  
  ✅ **Best for:** Developers who prefer a visual interface over the command line.

**7️⃣ MongoDB Search**

✅ **Full-text search engine** powered by **Lucene**  
✅ Built into **MongoDB Atlas**  
✅ Supports:

* Text search
* Relevance ranking
* Multi-language support

**8️⃣ MongoDB Data Lake**

✅ Store & analyze **large volumes of data**  
✅ Works with **JSON, BSON, CSV, Parquet** formats  
✅ Query data **without moving it** into MongoDB  
✅ Integrates with **AWS S3, Azure Blob Storage**

**9️⃣ MongoDB Backup & Restore**

✅ Automatic **backups & point-in-time recovery**  
✅ Available in **MongoDB Atlas & Enterprise Edition**  
✅ Supports **continuous backups & snapshots**

**🔹 Summary Table**

| **MongoDB Service** | **Description** | **Best For** |
| --- | --- | --- |
| **MongoDB Atlas** | Fully managed cloud-based MongoDB | Cloud applications |
| **MongoDB Enterprise** | Advanced security, monitoring, and compliance | Large-scale businesses |
| **MongoDB Community** | Free, open-source MongoDB version | Small projects & local development |
| **MongoDB Realm** | Mobile database with offline sync | Mobile & IoT apps |
| **MongoDB Charts** | No-code data visualization tool | Business intelligence |
| **MongoDB Compass** | GUI for MongoDB database management | Developers & DBAs |
| **MongoDB Search** | Built-in full-text search engine | Text-heavy applications |
| **MongoDB Data Lake** | Query large datasets without importing | Big Data analytics |
| **Backup & Restore** | Automated backups & recovery | Data protection |

**Final Thoughts**

MongoDB offers **powerful services** to help businesses and developers **store, manage, and analyze data efficiently**.