UNIT- V: MONGO DB

MongoDB: SQL and NoSQL Concepts Create and Manage MongoDB

1. SQL vs NoSQL (MongoDB) Concepts

SQL (Relational DB)	MongoDB (NoSQL, Document-based)
Database	Database
Table	Collection
Row	Document
Column	Field
Primary Key	_id field (unique by default)
Joins	Embedded documents or manual references
Schemas (Strict)	Schemaless (Flexible documents)
SQL Query (SELECT)	MongoDB Query (find)

2. Creating and Managing MongoDB

A. Installation & Setup

- **Install MongoDB:**
  MongoDB Installation Docs
- **Start MongoDB Server:**

```
mongod
```

- **MongoDB Shell (CLI):**

```
mongosh
```

B. Basic Operations

1. Create a Database

```
use myDatabase;
```

2. Create a Collection

```
db.createCollection("users");
```

3. Insert Documents

```
db.users.insertOne({
  name: "John Doe",
  email: "john@example.com",
  age: 25
});
```

4. Insert Multiple

```
db.users.insertMany([
  { name: "Alice", email: "alice@example.com" },
  { name: "Bob", email: "bob@example.com" }
]);
```

5. Read (Query) Data

```
db.users.find();                    // Get all
db.users.find({ name: "Alice" });     // Find by field
```

6. Update Document

```
db.users.updateOne(
  { name: "John Doe" },
  { $set: { age: 30 } }
);
```

7. Delete Document

```
db.users.deleteOne({ name: "Bob" });
```

C. Indexing

```
db.users.createIndex({ email: 1 }); // Ascending index on email
```

D. Relationships (Embedding vs Referencing)

1. Embedding (Like JOIN-less relationship)

```
db.orders.insertOne({
  userId: "123",
  items: [
    { productId: "p1", quantity: 2 },
    { productId: "p2", quantity: 1 }
  ]
});
```

2. Referencing

```
// In users collection
{ _id: ObjectId("userId123"), name: "User A" }

// In posts collection
{ userId: ObjectId("userId123"), title: "Post Title" }
```

Tools for Managing MongoDB

Tool	**Purpose**
MongoDB Compass	GUI for managing databases
Robo 3T	Another popular MongoDB GUI
mongosh	Shell for interacting with MongoDB
Mongoose (Node.js)	ODM for defining schemas in code

UNIT- V: MONGO DB

Migration of Data into MongoDB

Migrating data into **MongoDB** means transferring your existing data (often from relational databases like MySQL, PostgreSQL, or CSV/Excel files) into MongoDB's flexible document-based structure. Here's a step-by-step guide based on the **type of source** you're migrating from:

1. Migration from SQL (MySQL/PostgreSQL) to MongoDB

A. Using MongoDB's Official Tool: mongoimport

1. **Export SQL Data to CSV or JSON**
   Example using MySQL:

```
SELECT * FROM users INTO OUTFILE '/tmp/users.csv'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. **Import CSV to MongoDB**

```
mongoimport --db mydb --collection users --type csv --file /tmp/users.csv --headerline
```

3. **Import JSON to MongoDB**
   If your SQL data is exported as JSON:

```
mongoimport --db mydb --collection users --file users.json --jsonArray
```

B. Using a Node.js Script (ETL Process)

1. **Install Required Packages**

```
npm install mysql2 mongoose
```

2. **Sample Migration Script (MySQL → MongoDB)**

```
const mysql = require('mysql2/promise');
const mongoose = require('mongoose');
```

```
async function migrate() {
  await mongoose.connect('mongodb://localhost:27017/mydb');
  const User = mongoose.model('User', new mongoose.Schema({}, { strict: false }));

  const connection = await mysql.createConnection({ host: 'localhost', user: 'root',
password: '', database: 'sql_db' });
  const [rows] = await connection.execute('SELECT * FROM users');

  for (const row of rows) {
    await User.create(row); // Dynamic schema
  }

  console.log('Migration complete');
  await connection.end();
  await mongoose.disconnect();
}

migrate();
```

2. Migration from CSV/Excel Files

A. Using mongoimport (CSV)

```
mongoimport --db mydb --collection products --type csv --file products.csv --headerline
```

B. Using Node.js + csv-parser

```
npm install csv-parser mongoose
const fs = require('fs');
const csv = require('csv-parser');
const mongoose = require('mongoose');
```

```
mongoose.connect('mongodb://localhost:27017/mydb');
const Product = mongoose.model('Product', new mongoose.Schema({}, { strict: false }));

fs.createReadStream('products.csv')
  .pipe(csv())
  .on('data', async (row) => {
    await Product.create(row);
  })
  .on('end', () => {
    console.log('CSV file successfully processed');
    mongoose.disconnect();
  });
```

MongoDB with ReactJS

Integrating **MongoDB with ReactJS** involves building a **full-stack application** using:

- **Frontend:** ReactJS (handles UI)
- **Backend:** Node.js with Express (handles APIs)
- **Database:** MongoDB (stores data)

Since **ReactJS** can't connect directly to MongoDB (as it would expose credentials), we use a **Node.js backend** to act as a middle layer.

Full-Stack Architecture Overview

```
 [ ReactJS Frontend ]
        ⬇ (HTTP)
[ Express.js API Backend ]
        ⬇ (Mongoose ODM)
[ MongoDB Database ]
```

UNIT- V: MONGO DB

✅Step-by-Step Guide: MongoDB + ReactJS

1. Setup Backend (Node.js + Express + MongoDB)

A. Create a Backend Folder

```
mkdir backend && cd backend
npm init -y
npm install express mongoose cors dotenv
```

B. Create File Structure

```
backend/
│
├── models/
│   └── User.js
├── routes/
│   └── userRoutes.js
├── .env
├── server.js
```

C. .env

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/mydb
```

D. models/User.js

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
});
module.exports = mongoose.model('User', userSchema);
```

E. routes/userRoutes.js

```
const express = require('express');
const router = express.Router();
const User = require('../models/User');

router.get('/', async (req, res) => {
  const users = await User.find();
  res.json(users);
});

router.post('/', async (req, res) => {
  const newUser = new User(req.body);
  await newUser.save();
  res.status(201).json(newUser);
});

module.exports = router;
```

F. server.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');

dotenv.config();
const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGO_URI)
```

```
  .then(() => console.log('MongoDB Connected'))
  .catch((err) => console.error('MongoDB Error:', err));

app.use('/api/users', require('./routes/userRoutes'));

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

2. Setup Frontend (ReactJS)

A. Create React App

```
npx create-react-app frontend
cd frontend
npm install axios
```

B. Sample React Component (App.js)

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

function App() {
  const [users, setUsers] = useState([]);
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  const fetchUsers = async () => {
    const res = await axios.get('http://localhost:5000/api/users');
    setUsers(res.data);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    await axios.post('http://localhost:5000/api/users', { name, email });
    setName('');
    setEmail('');
    fetchUsers();
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  return (
    <div>
      <h1>User Manager</h1>
      <form onSubmit={handleSubmit}>
        <input value={name} onChange={(e) => setName(e.target.value)} placeholder="Name" />
        <input value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" />
        <button type="submit">Add User</button>
      </form>

      <ul>
        {users.map(user => (
          <li key={user._id}>{user.name} ({user.email})</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

UNIT- V: MONGO DB

✅Final Notes

- Run backend:

```
cd backend
node server.js
```

- Run frontend:

```
cd frontend
npm start
```

MongoDB with NodeJS

Integrating **MongoDB with Node.js** is a common and powerful setup for building backend applications or APIs. Below is a **complete guide** to help you set up and work with **MongoDB in a Node.js environment** using **Mongoose**, the most popular ODM (Object Data Modeling) library.

✅1. Prerequisites

- Node.js installed
- MongoDB running locally or using **MongoDB Atlas**
- Basic knowledge of JavaScript

2. Project Setup

A. Initialize the Project

```
mkdir mongodb-node-app
cd mongodb-node-app
npm init -y
npm install express mongoose dotenv
```

3. Folder Structure

```
mongodb-node-app/
│
├── models/
│   └── User.js
├── routes/
│   └── userRoutes.js
├── .env
├── server.js
```

4. .env File

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/mydb
```

Replace MONGO_URI with your **MongoDB Atlas URI** if you're using the cloud.

5. Create a Mongoose Model (models/User.js)

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

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('User', userSchema);
```

6. Create Routes (routes/userRoutes.js)

```
const express = require('express');
const router = express.Router();
const User = require('../models/User');

// Get all users
router.get('/', async (req, res) => {
  const users = await User.find();
  res.json(users);
});

// Add a new user
router.post('/', async (req, res) => {
  try {
    const newUser = await User.create(req.body);
    res.status(201).json(newUser);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});

module.exports = router;
```

7. Setup Express Server (server.js)

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const userRoutes = require('./routes/userRoutes');

dotenv.config();
const app = express();
app.use(express.json());

mongoose.connect(process.env.MONGO_URI)
```

```
  .then(() => console.log('✅MongoDB connected'))
  .catch((err) => console.error('❌MongoDB connection error:', err));

app.use('/api/users', userRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`□ Server running on http://localhost:${PORT}`));
```

8. Test API with Postman or Curl

➕Add User

```
POST http://localhost:5000/api/users
Content-Type: application/json

{
  "name": "Alice",
  "email": "alice@example.com"
}
```

Get All Users

```
GET http://localhost:5000/api/users
```

Services Offered by MongoDB

MongoDB offers a variety of **services and tools** designed to make modern application development faster, more scalable, and secure. Below is a categorized list of the **key services offered by MongoDB**:

1. MongoDB Atlas (Fully Managed Cloud Database)

MongoDB Atlas is the **flagship service** — a fully-managed cloud database service.

Key Features:

- Automated deployment on **AWS, Azure, GCP**
- Auto-scaling, backups, monitoring
- Built-in **security** and **compliance** (SOC 2, HIPAA, etc.)
- Global clusters for low-latency apps
- Performance optimization tools

2. Database Services

Service	Description
MongoDB Community Edition	Free, open-source version for local/self-hosted deployments
MongoDB Enterprise Edition	Advanced security, in-memory storage, and commercial support
MongoDB Atlas	Cloud-native managed database with built-in automation & security
Atlas for Government	FedRAMP-authorized Atlas version for U.S. government use

3. Developer Services

Tool/Service	Description
MongoDB Compass	GUI tool to explore, visualize, and query MongoDB data
Atlas CLI	Command-line interface to manage Atlas resources
MongoDB Shell	Powerful modern shell (mongosh) to interact with MongoDB
MongoDB Charts	Data visualization tool natively integrated with MongoDB Atlas
MongoDB VS Code Plugin	Browse collections, run queries, and manage databases from VS Code

4. Data Services

Feature/Service	Purpose
Atlas Search	Full-text search using Lucene, built into Atlas
Atlas Data Federation	Query across multiple data sources (e.g., S3, other clusters)
Atlas Data Lake	Run analytical queries on archived data in cloud storage
Change Streams	Real-time data change notifications for event-driven architectures
Triggers	Serverless functions triggered by database changes

5. Application Services (Backend as a Service)

Feature	Description
Atlas App Services	Serverless backend with authentication, triggers, and GraphQL APIs
Authentication Providers	Built-in support for OAuth, Email/Password, JWT, and Custom auth
Functions	Run serverless functions inside MongoDB Atlas
GraphQL API	Auto-generated GraphQL API layer over your MongoDB collections
Data Sync	Sync data between client apps (mobile/web) and Atlas backend in real time

6. Mobile Services

Service	Description
Realm Database	Local embedded database for iOS/Android with sync capabilities
Realm Sync	Real-time sync between MongoDB Atlas and mobile apps
Device Sync	Bi-directional syncing and offline-first capabilities for mobile apps

7. Security & Compliance Services

Feature	Description
Encryption at Rest & TLS	Built-in encryption for stored data and data in transit
Role-Based Access Control (RBAC)	Granular access control for users and applications
Auditing	Track access and operations for compliance
Backup & Restore	Continuous backups and point-in-time recovery
VPC Peering & PrivateLink	Secure private networking in cloud deployments

8. Monitoring & Analytics

Tool	Purpose
Atlas Monitoring	Real-time dashboards for performance and metrics
Performance Advisor	Query optimization suggestions
Slow Query Analyzer	Analyze inefficient queries

9. Migration Services

Tool/Service	Description
MongoMirror	Migrate from on-prem MongoDB to Atlas
Live Migration Service	Move data from another cloud provider to Atlas with minimal downtime
MongoDB Compass	Import/export data through CSV/JSON and run migration queries

10. Support & Training

Service	Details
MongoDB University	Free online training courses on MongoDB (Beginner to Advanced)
Enterprise Support	Dedicated technical support for enterprise customers
Documentation & SDKs	Extensive docs + SDKs for Node.js, Python, Java, C#, Go, Rust, etc.