Database Connectivity in JS

SQL vs NoSQL

Feature	SQL (Relational Databases)	NoSQL (Non-Relational Databases)
Structure	Table-based (rows & columns)	Document, Key-Value, Column-Family, or Graph-based
Schema	Fixed schema (predefined structure)	Flexible schema (dynamic structure)
Scalability	Vertically scalable (adding CPU/RAM)	Horizontally scalable (adding more servers)
ACID Compliance	Follows ACID (Atomicity, Consistency, Isolation, Durability)	May sacrifice some ACID properties for speed (follows BASE)
Query Language	SQL (Structured Query Language)	NoSQL-specific queries (MongoDB uses JSON-like queries)
Best For	Structured data (banking, ERP, CRM)	Unstructured/semi-structured data (social media, big data, real-time apps)
Examples	MySQL, PostgreSQL, Oracle, SQL Server	MongoDB, Cassandra, Firebase, Redis

Using MongoClient

Step 1: Install MongoDB Driver

npm install mongodb

Step 2: Connect to MongoDB (`db.js`)

Step 3: Insert Data (`insertUser.js`)

Step 4: Fetch Data (`fetchUsers.js`)

NOSQL Database Connectivity

Connect to MongoDB (db.js)

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

const MONGO_URI = "mongodb://localhost:27017";
const DB_NAME = "myappDB";

const client = new MongoClient(MONGO_URI);

async function connectDB() {
  try {
    await client.connect();
    console.log("MongoDB Connected Successfully!");
    return client.db(DB_NAME);
  } catch (error) {
    console.error("MongoDB Connection Error:", error);
    process.exit(1);
  }
}

module.exports = connectDB;
```

Insert Data (`insertUser.js`)

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

async function insertUser() {
  try {
    const db = await connectDB();
    const usersCollection = db.collection("users");

    const newUser = { name: "Ashish Bajpai", email: "ab@gmail.com", age: 42 };

    const result = await usersCollection.insertOne(newUser);
    console.log("User Inserted:", result.insertedId);
  } catch (error) {
    console.error("Error inserting user:", error);
  }
}

insertUser();
```

Fetch Data (fetchUsers.js)

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

async function fetchUsers() {
  try {
    const db = await connectDB();
    const usersCollection = db.collection("users");

    const users = await usersCollection.find().toArray();
    console.log("Users in Database:", users);
  } catch (error) {
    console.error("Error fetching users:", error);
  }
}

fetchUsers();
```

Using ODM (mongoose)

ODM (Object-Document Mapping) is a technique used in **NoSQL databases** (like **MongoDB**) to map **JavaScript objects** to **database documents**. It works similarly to ORM (Object-Relational Mapping) but for **document-based databases** instead of relational ones.

STEP-1: install Dependencies (mongoose)

```
npm install mongoose
```

STEP-2 Connect to MongoDB(db.js)

STEP-3 Define a Model (models/User.js)

STEP-4 Insert Data (insertUser.js)

STEP-5 Fetch Data (fetchUser.js)

Connect to MongoDB(db.js)

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

const MONGO_URI = "mongodb://localhost:27017/myappDB";


mongoose.connect(MONGO_URI)
  .then(() => console.log("MongoDB Connected"))
  .catch((err) => console.error("MongoDB Connection Error:", err));

module.exports = mongoose;
```

Define a Model (models/User.js)

```
const mongoose = require("../db");

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, unique: true, required: true },
  age: { type: Number },
});


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

module.exports = User;
```

Insert Data (insertUser.js)

```
const User = require("./models/User");

const insertData=async() => {
  try {
    const newUser = new User({
      name: "John Doe",
      email: "john@example.com",
      age: 25,
    });

    await newUser.save();
    console.log("User Added:", newUser);
  } catch (error) {
    console.error("Error inserting user:", error);
  }
}

insertData();
```

Fetch Data (`fetchUsers.js`)

```
const User = require("./models/User");

const fetchdata=async () => {
  try {
    const users = await User.find();
    console.log("Users:", users);
  } catch (error) {
    console.error("Error fetching users:", error);
  }
}

fetchdata();
```

SQL Database Connectivity using PostgreSQL

Using Native Driver

Step-1: Install PostgreSQL Client for Node.js

```
npm install pg
```

Step-2: Setup PostgreSQL Connection (`db.js`)

Step-3: Create a Table (`createTable.js`)

Step-4: Insert Data (`insertUser.js`)

Step-5: Fetch Data (`fetchUsers.js`)

Setup PostgreSQL Connection (`db.js`)

```
const { Client } = require("pg");

const client = new Client({
  user: "postgres",
  host: "localhost",
  database: "mydatabase",
  password: "your_password",
  port: 5432
});


client.connect()
  .then(() => console.log("Connected to PostgreSQL!"))
  .catch((err) => console.error("Connection error", err.stack));

module.exports = client;
```

Create a Table

```
const client = require("./db");

async function createTable() {
  try {
    await client.query(`
      CREATE TABLE IF NOT EXISTS users (
        id SERIAL PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        email VARCHAR(100) UNIQUE NOT NULL
      );
    `);
    console.log("Users table created!");
  } catch (error) {
    console.error("Error creating table:", error);
  } finally {
    client.end();
  }
}

createTable();
```

Insert Data (`insertUser.js`)

```
const client = require("./db");

async function insertUser() {
  try {
    const res = await client.query(
      "INSERT INTO users (name, email) VALUES ($1, $2) RETURNING *",
      ["Ashish Bajpai", "zb@gmail.com"]
    );
    console.log("User Inserted:", res.rows[0]);
  } catch (error) {
    console.error("Error inserting user:", error);
  } finally {
    client.end();
  }
}

insertUser();
```

Fetch Data (`fetchUsers.js`)

```
const client = require("./db");

async function fetchUsers() {
  try {
    const res = await client.query("SELECT * FROM users");
    console.log("Users:", res.rows);
  } catch (error) {
    console.error("Error fetching users:", error);
  } finally {
    client.end();
  }
}

fetchUsers();
```

Using ORM

ODM vs ORM

Feature	ODM (Object-Document Mapping)	ORM (Object-Relational Mapping)
Database Type	NoSQL (MongoDB, CouchDB)	SQL (PostgreSQL, MySQL)
Data Structure	JSON-like Documents	Tables (Rows & Columns)
Schema Flexibility	Dynamic Schema (Flexible)	Strict Schema (Fixed)
Query Language	Query Methods (Mongoose.find())	SQL-based Queries
Relationships	Embedded/Nested Data	Foreign Keys & Joins

Using ORM

Sequelize is an **ORM (Object Relational Mapper)** that makes it easier to work with PostgreSQL in **Node.js**. Here's how to connect **PostgreSQL (PgAdmin Database) with Sequelize**.

Step-1: Install PostgreSQL Client for Node.js

npm install sequelize pg pg-hstore

Step-2: Setup Database Connection (`db.js`)

Step-3: Define a Model (`models/User.js`)

Step-4: Insert Data (`insertUser.js`)

Step-5: Fetch Data (`fetchUsers.js`)

PostgreSQL Connection Setup

```
const { Sequelize } = require("sequelize");

const sequelize = new Sequelize("myappDB", "postgres", "abes@123", {
  host: "localhost",
  dialect: "postgres",
  logging: false,
});


(async () => {
  try {
    await sequelize.authenticate();
    console.log("Connected to PostgreSQL!");
  } catch (error) {
    console.error("Connection Error:", error);
  }
})();

module.exports = sequelize;
```

User Model setup

```
const { DataTypes } = require("sequelize");
const sequelize = require("../db");


const User = sequelize.define("User", {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true,
  },
  name: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  email: {
    type: DataTypes.STRING,
    unique: true,
    allowNull: false,
  },
});


(async () => {
  await sequelize.sync({ force: false });
  console.log("Users table is ready!");
})();

module.exports = User;
```

Insert Data (`insertUser.js`)

Add User

```
const User = require("./models/User");

(async () => {
  try {
    const newUser = await User.create({
      name: "Ashish Bajpai",
      email: "ab@gmail.com",
    });

    console.log("New User Added:", newUser.toJSON());
  } catch (error) {
    console.error("Error inserting user:", error);
  }
})();
```

Fetch Data (`fetchUsers.js`)

Fetch Users from Database

```
const User = require("./models/User");

(async () => {
  try {
    const users = await User.findAll();
    console.log("Users in Database:", users.map(user => user.toJSON()));
  } catch (error) {
    console.error("Error fetching users:", error);
  }
})();
```

Update User Data (updateUser.js)

Update User Details

```
const User = require("./models/User");

(async () => {
  try {
    const userId = 1;
    const updatedData = { name: "Aatif Jamshed", email: "aj@abes.ac.in" };


    const user = await User.findByPk(userId);
    if (!user) {
      console.log("User not found!");
      return;
    }


    await user.update(updatedData);
    console.log("User Updated:", user.toJSON());
  } catch (error) {
    console.error("Error updating user:", error);
  }
})();
```

Thank You