Module 05

10/03/2025

Page 71

MongoDB

1. Explain what MongoDB is and its key feature compared to traditional relational database.

→

- MongoDB is a NOSQL database that stores data in a JSON-like document format instead of traditional tables and rows. It is schema-less, scalable and flexible, making it ideal for handling large volumes of unstructured or semi-structured data.

→ Feature	MongoDB (NOSQL)	Relational Databases (SQL)
- Data Model	Document oriented (JSON/BSON)	Table-based (row & columns)
- Schema	Schema-less, flexible	Fixed schema (structured)
- Scalability	Horizontally scalable (sharding)	Vertically scalable
- Performance	Faster for large scale, unstructured data.	Slower for large unstructure Data.
- Use case	Best for Big Data, IoT, and real-time apps	Best for structured data applications.

2. Discuss the significance of database and Collections in MongoDB and how they differ from tables in relational databases.

→

→ Databases in MongoDB :- A mongodb database is a container for collections. Each database contains multiple collections and store data in BSON format.

→ Collections in MongoDB :- A collection is a group of document that stores related data but does not enforce a strict schema.

→ Feature	MongoDB Collection	SQL Tables
- Structure	Schema-less, flexible	Fixed schema
- Data Storage	JSON-like document	Rows and Columns
- Relation-ships	Uses embedded/ nested document	Uses foreign keys and joins
- Scalability	Horizontally scalable	Mostly vertically scalable.

8. Describe the difference between dropping a database and dropping a collection in MongoDB.

→

Operations	Dropping a Database	Dropping a Collection
- Effect	Deletes the entire database, including all collections and data	Deletes only a specific collection but keeps the database intact.
- Command	db.dropDatabase()	db.collection.drop()
- Use case	used when you want to remove all data and structure of a Database	used when you only want to delete specific data while keeping the database.

4. Write a MongoDB query to create a new database named "my_database".

→

```
use my_database
```

5. Write a MongoDB query to create a new collection named "my_collection" collection with in the "my_database" database.

→ db.createCollection("my_collection");

6. Write a MongoDB query to insert a document into the "my_collection" with the fields "name" & "age".

→

```
db.my_collection.insertOne(
    {
        name: "Utsav",
        age: 20
    }
);
```

7. Write a MongoDB query to find all documents in the "my_collection" collection.

→ db.my_collection.find();

8. Write a MongoDB query to update the 'age' field of a document in the "my_collection" collection.

→ db.my_collection.updateOne(
{ name: "Priya" },
{ $set: { age: 25 } }
);

9. Write a MongoDB query to delete a document from the "my_collection" collection.

→ db.my_collection.deleteOne(
{ name: "Priya" }
);

10. Discuss the advantages and disadvantages of using MongoDB for storing large-scale data compared to traditional relationship databases.

→ Advantages :-

- Scalability :- Horizontally scales with sharding for large data.
- Flexible Schema :- No fixed structured, allowing easy modifications.
- Faster Reads / Writes :- No need for complex joins; uses embedded documents.

- Document Oriented :- Stores data in JSON-like format, making it developer-friendly.
- High Availability :- Supports replications for data redundancy.

→ Disadvantages :-

- Lack of Joins :- No built-in join support, leading to data duplications.
- Memory Usage :- Stores metadata for each document, increasing storage overhead.
- Not Ideal for Transactions :- Lacks strong ACID compliance like SQL databases.
- Learning Curve :- Developers used to SQL must adapt to NoSQL concepts.

11. Describe the process of inserting documents into a MongoDB collection and provide examples of different data structures that can be imbedded.

→ Process of Inserting Documents in MongoDB :-

1) Connect to the database :- use mydatabase_name to select or create a database.
2) Choose a collection :- store documents, similar to tables in SQL.

iii) Insert Documents :- Use insertOne() for a single document or insertMany() for multiple documents.

iv) Verify Data :- use find() to check the inserted data.

→ Example of different Data structures in.

- Simple Document (Flat Structure) :-

```
db.user.insertOne({
    name: 'utsav',
    age: 20
    city: 'Bhavnagar'
});
```

- Embedded Document (Nested structure) :-

```
db.users.insertOne({
    name: "Bob",
    age: 35,
    address: { street: "123 Main st",
               city: "Los Angeles",
               zip: "90001"
             }
});
```

- Array of Values :-

```
db.users.insertOne({
    name: "charlie",
    hobies: ["reading", "traveling", "gaming"]
});
```

- Insert Multiple Documents :-

```
db.users.insertMany([
    {name: "Jil", age: 23},
    {name: "Krupa", age: 22}
]);
```

12. Write a Node.js script using the MongoDB Node.js driver to connect to a MongoDB database and insert multiple documents into a collection.

→

```
const {MongoClient} = require('mongodb');
const url = "mongodb://localhost:2115";
const client = new MongoClient(url);
const dbName = "my_database";

async function run() {
  try {
    await client.connect();
    console.log("Connected to MongoDB");

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

    const result = await collection.insertMany
    ([ {name: "utsav", age: 22},
       {name: "Priya", age: 20},
       {name: "Sujal", age: 814}
    ]);

    console.log(`${result.insertedCount}
        documents inserted`);
```

```
} finally {
    await client.close();
}
}
run().catch(console.error);
```

13. Develop a Node.js application with Express.js that Implements CRUD operations for a MongoDB collection (Create, Read, Update, Delete).

→

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

const app = express();
app.use(express.json());
const url = "mongodb://localhost:2115";
const client = new MongoClient(url);
const dbName = "my_database";

async function connectDB() {
    await client.connect();
    console.log("Connected to MongoDB");
}
connectDB();

const db = client.db(dbName);
const collection = db.collection("users");
```

```
//CREATE : Insert a New user
app.post("/users", async (req, res) => {
    const result = await collection.insert
        One(req.body);
    res.send(result);
});

//READ
app.get("/users", async (req, res) => {
    const users = await collection.find().
        toArray();
    res.send(users);
});

//UPDATE
app.put("/users/:id", async (req, res) => {
    const result = await collection.updateOne
    ( {_id: new ObjectId(req.params.id)},
      { $set : req.body }
    );
    res.send(result);
});

//DELETE
app.delete("/users/:id", async (req, res) => {
    const result = await collection.
        deleteOne({_id: new ObjectId(
            req.params.id)
    });
    res.send(result);
});
```

```
app.listen(3000, () => {
    console.log("Server is running on
                port 3000");
});
```

14. Build a full-stack e-commerce application using Node.js, Express.js, React.js and MongoDB for managing product listings, user accounts, and orders.

→

step-1 Set up a Backend (Node.js + Express.js + MongoDB).

```
npm init -y
npm install express mongoose cors
            body-parser
```

→ server.js

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

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

mongoose.connect("mongodb://localhost:
    2115/ecommerce", {
    useNewURLParser: true,
    useUnifiedTopology: true
});
```

```
const ProductSchema = new mongoose.
    Schema({
      name: String,
      price: Number,
      description: String
});

const Product = mongoose.model("Product",
      productSchema);

// create Product
app.post("/products", async (req, res) => {
    const product = new Product(req.body);
    await product.save();
    res.send(product);
});

//READ Products
app.get("/products", async (req, res) => {
    const products = await Product.find();
    res.send(products);
});

app.listen(5000, () => console.log("Server running
        on port 5000"));
```

Step-2 Setup the Frontend (React.js) :-

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

→ App.js

```
import React, {useEffect, useState} from "React";
import axios from "axios";

function App(){
  const [products, setProducts] = useState([]);
  useEffect(() => {
    axios.get("http://localhost:5000/products").then((response) => {
      setProducts(response.data);
    });
  }, []);

  return (
    <div>
      <h1>Product List</h1>
      <ul>
        {products.map((product) => (
          <li key={product._id}>
            {product.name} - {product.price}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```