**MongoDB Notes for Beginners**

**What is MongoDB?**

MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called BSON (Binary JSON). This means you can easily change the structure of your data without needing to define it in advance, making it perfect for modern applications.

**Key Features:**

* **Flexible Schema**: You can store different types of data in the same collection.
* **Scalable**: Can handle large amounts of data across multiple servers.
* **Fast Performance**: Designed for quick data retrieval.

**Key Concepts**

**1. Database**

A **database** is like a folder that contains your collections. You can have multiple databases in MongoDB.

**2. Collection**

A **collection** is like a folder inside the database that holds related documents. For example, you might have a collection called "Users" to store user information.

**3. Document**

A **document** is a single record in a collection. It’s similar to a row in a spreadsheet. Here’s what a user document might look like:

*#Json code*

{

"\_id": "1",

"name": "Alice",

"age": 15,

"email": "alice@example.com"

}

**4. Unique Identifier (\_id)**

Each document has a unique identifier called **\_id**. If you don’t specify it, MongoDB will create one for you.

**Basic Operations**

**Installing MongoDB**

To start using MongoDB, you need to install it on your computer. You can download it from the [official MongoDB website](https://www.mongodb.com/try/download/community).

**Starting MongoDB**

Once installed, you start the MongoDB server, usually by running the command mongod in the terminal.

**Connecting to MongoDB**

You can interact with your database using the MongoDB shell, which you access by typing mongo in the terminal.

**Creating a Database and Collections**

**Create or Switch to a Database**

To create a new database or switch to an existing one, you use:

*Javascript- code*

use myDatabase

**Create a Collection**

To create a collection within that database, you can type:

*Javascript- code*

db.createCollection("Users")

**Inserting Documents**

**Adding Users**

You can add a new user to the "Users" collection using the insertOne method:

*Javascript- code*

db.Users.insertOne({

name: "Alice",

age: 15,

email: "alice@example.com"

})

If you want to add multiple users at once, use insertMany:

*Javascript- code*

db.Users.insertMany([

{ name: "Bob", age: 16, email: "bob@example.com" },

{ name: "Charlie", age: 14, email: "charlie@example.com" }

])

**Querying Documents**

**Find All Users**

To see all the documents in the "Users" collection, use:

*Javascript- code*

db.Users.find()

**Query with Conditions**

To find users who are older than 15, you can do:

*Javascript- code*

db.Users.find({ age: { $gt: 15 } })

**Updating Documents**

**Update a User's Age**

If Alice turns 16, you can update her age like this:

*Javascript- code*

db.Users.updateOne(

{ name: "Alice" },

{ $set: { age: 16 } }

)

**Update Multiple Users**

You can also update multiple users at once. For example, to mark all users aged 15 as "young":

*Javascript- code*

db.Users.updateMany(

{ age: 15 },

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

)

**Deleting Documents**

**Remove a User**

To delete Alice from the "Users" collection, you would use:

*Javascript- code*

db.Users.deleteOne({ name: "Alice" })

**Remove Multiple Users**

If you want to delete all users under 15:

*Javascript- code*

db.Users.deleteMany({ age: { $lt: 15 } })

**Extra Features**

**Indexing**

Indexes help speed up searches in the database. You can create an index on the name field like this:

*Javascript- code*

db.Users.createIndex({ name: 1 }) // 1 for ascending order

**Aggregation**

The aggregation framework lets you perform operations like counting or grouping data. For example, to count how many users are over 15:

*Javascript- code*

db.Users.aggregate([

{ $match: { age: { $gt: 15 } } },

{ $group: { \_id: null, count: { $sum: 1 } } }

])

**Replication and Sharding**

* **Replication** keeps copies of your data across different servers to ensure it’s always available.
* **Sharding** splits your data across multiple servers to handle larger datasets more efficiently.

**Using MongoDB with Node.js**

**What is Node.js?**

Node.js is a JavaScript runtime that allows you to run JavaScript code on the server side, which is great for web applications.

**Connecting MongoDB and Node.js**

To connect to MongoDB from a Node.js application, you can use a library called **Mongoose**.

1. **Install Mongoose**:

bash

Copy code

npm install mongoose

1. **Basic Example**: Here’s how you can create a simple Node.js application to add a user:

*Javascript- code*

const mongoose = require('mongoose');

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/myDatabase', {

useNewUrlParser: true,

useUnifiedTopology: true

});

// Define a User Schema

const userSchema = new mongoose.Schema({

name: String,

age: Number,

email: String

});

// Create a User Model

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

// Create and Save a User

const createUser = async () => {

const user = new User({ name: 'Alice', age: 15, email: 'alice@example.com' });

await user.save();

console.log('User saved:', user);

};

// Call the function to create a user

createUser();

**Learning Resources**

* **Official Documentation**: The best place to learn more is the [MongoDB Documentation](https://docs.mongodb.com/).
* **Online Courses**: Websites like Udemy and Coursera offer beginner-friendly courses on MongoDB.