Database-(MongoDB)

A **database** is an organized collection of data that can be easily accessed, managed, and updated. It stores information in a structured way, making it easy to retrieve and manipulate using a database management system (DBMS).

Types of Databases

1. **Relational Databases (SQL-based)**
   - Uses tables with rows and columns.
   - Examples: MySQL, PostgreSQL, SQL Server, SQLite.
2. **NoSQL Databases**
   - Stores data in formats like key-value pairs, documents, or graphs.
   - Examples: MongoDB, Firebase, Cassandra, Redis.

The choice between **SQL (Relational Databases)** and **NoSQL (Non-Relational Databases)** depends on your project's requirements. Here's a comparison to help you decide:

Factor	SQL (Relational DBs)	NoSQL (Non-Relational DBs)
Structure	Tables with rows & columns (fixed schema)	Flexible schema (documents, key-value, graph, etc.)

Factor	SQL (Relational DBs)	NoSQL (Non-Relational DBs)
Scalability	Vertical scaling (increasing hardware power)	Horizontal scaling (adding more servers)
Performance	Better for complex queries (JOINs)	Faster for large-scale read/write operations
Data Integrity	Strong ACID compliance (reliable transactions)	Eventual consistency (depends on DB type)
Best For	Structured data (e.g., banking, e-commerce)	Unstructured or semi-structured data (e.g., social media, IoT, real-time apps)
Examples	MySQL, PostgreSQL, SQL Server	MongoDB, Firebase, Cassandra, Redis

What is MongoDB

MongoDB is a **NoSQL database** that stores data in a **document-oriented format** using **JSON-like BSON (Binary JSON)**. It is designed for **high performance, scalability, and flexibility**, making it a great choice for modern web applications, especially in the **MERN stack** (MongoDB, Express.js, React, Node.js).

What is BSON(Binary JSON)

BSON (Binary JSON) is a special format used by **MongoDB** to store data. It is similar to **JSON**, but faster and more efficient because it is stored in **binary format**.

Key Features of MongoDB

✅ **Schema-less:** No fixed table structure, allowing flexible and dynamic data storage.

✅ **Scalable:** Supports **horizontal scaling** (sharding) for handling large amounts of data.

✅ **High Performance:** Faster read/write operations compared to traditional SQL databases.

✅ **Indexing:** Uses indexes for efficient query execution.

✅ **Replication:** Ensures high availability with automatic failover and backup.

MongoDB Community Server

MongoDB **Community Server** is the **free version** of MongoDB that you can install on your computer or server. It allows you to store, manage, and retrieve data efficiently

Why Use MongoDB Community Server?

✅ **Free to Use** – No cost, open-source.

✅ **Stores Data as JSON** – Easy to use with JavaScript & MERN stack.

✅ **Fast & Scalable** – Can handle large amounts of data efficiently.

✅ **Works Offline** – Runs on your local computer, no internet needed.

What is MongoDB Shell?

MongoDB **Shell** (also called **mongosh**) is a **command-line tool** that lets you interact with your MongoDB database. You can use it to **create, read, update, and delete** data, just like a control panel for MongoDB.

Open the MongoDB Shell by typing:- mongosh to start MongoDB shell, its also allowing you to execute java script commands to interact with the **database**

What are Collections and Documents in MongoDB?

In **MongoDB,** data is stored in a structure similar to folders and files:

🗁 **Collection** → Like a **folder** that holds **multiple documents**.
🗎 **Document** → Like a **file** inside the folder, containing actual data in **JSON-like** format.

1 What is a Collection?

A **collection** in MongoDB is a group of **documents**. It is similar to a table in SQL databases, but it doesn't have a fixed structure.

✅ A **collection** can store multiple documents.
✅ Each document can have different fields (no strict schema).
✅ Collections are created automatically when you insert data.

2 What is a Document?

A **document** is an individual record inside a collection. It is stored in **BSON format (Binary JSON)** but looks like JSON when viewed.

✅ Each document is a JSON-like object.
✅ It contains key-value pairs (e.g., "name": "Alice").
✅ Unlike SQL tables, documents in the same collection can have different fields.

Basic MongoDB Shell Commands

Show all databases:- show dbs

Switch to a database (or create one):- use myDatabase

Show all collections in a database:- show collections

- ## How to Insert Data into MongoDB?

In MongoDB, **data is stored in documents** inside **collections**. You can insert data using the **MongoDB Shell (mongosh)**, **MongoDB Compass (GUI)**, or **Node.js (Mongoose for MERN apps).**

a) Insert a Single Document

Syntax:

```
Db.collectionName.insertOne({key: "value", key: "value", key: "value"});
```

Exp:

```
db.users.insertOne({ name: "Alice", age: 25, city: "New York" });
```

b) Insert Multiple Documents

```
db.users.insertMany([
  { name: "Bob", age: 30, city: "London" },
  { name: "Charlie", age: 28, email: "charlie@example.com" }
]);
```

- **<u>How to Find Data in MongoDB?</u>**

In MongoDB, you can **retrieve data** using the **find()** method. You can search for **all documents, specific documents, or filter based on conditions**.

1. **<u>Find All Documents</u>**

```
db.users.find();
```

2. **<u>Find a Single Document</u>**

```
db.users.findOne({ name: "Alice" });
```

<u>Find Documents with Conditions</u>

1. **Find users older than 25**

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

2. Find users **who live in "New York"**

```
db.users.find({ city: "New York" });
```

<u>Find Using Multiple Conditions</u>

1. **Find users older than 25 and living in "New York":**

```
db.users.find({ age: { $gt: 25 }, city: "New York" });
```

2. **Find user who live in delhi and mumbai**

```
db.users.find({city: {$ in : ['delhi', 'mumbai']}}
```

- **<u>How to Update Data in MongoDB?</u>**

In MongoDB, you can update documents using **updateOne()**, **updateMany()**, and **replaceOne()**. You can do this using **MongoDB Shell**, **Compass**, or **Mongoose (Node.js)**.

1. Update a Single Document

Change Alice's age to 26

```
db.users.updateOne(
  { name: "Alice" },  // Search condition
  { $set: { age: 26 } } // Update field
    );
```

Updates **only the first matching document**.

2. Update Multiple Documents

Increase the age of all users in "New York" by 2

```
db.users.updateMany(
  { city: "New York" },
  { $inc: { age: 2 } } // Increment age by 2
    );
```

Updates **all matching documents**.

3. Replace an Entire Document

Replace Alice's document with a new one

```
db.users.replaceOne(
  { name: "Alice" },  // Search condition
  { name: "Alice", age: 27, email: "alice@example.com" } // New
document
   );
```

Replaces the **entire document**, keeping only the specified fields.

- ## How to Delete Data in MongoDB?

In MongoDB, you can **delete documents** using **deleteOne()**, **deleteMany()**, or **drop()**. You can do this via **MongoDB Shell**, **Compass**, or **Mongoose (Node.js).**

1. Delete a Single Document

Delete one user named "Alice"

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

Deletes **only the first matching document**.

2. Delete Multiple Documents

Delete all users who live in "New York"

```
db.users.deleteMany({ city: "New York" });
```

Deletes **all matching documents**.

3. Delete All Documents from a Collection

```
db.users.deleteMany({});
```

Removes **all documents** but keeps the collection.

4. Delete an Entire Collection

```
db.users.drop();
```

Permanently deletes the collection.