What is MongoDB?

MongoDB is a **NoSQL** (non-relational) **document database** designed for scalability, performance, and high availability. Instead of storing data in traditional **tables and rows** like relational databases (e.g., MySQL, PostgreSQL),

Note:-

MongoDB stores data in **JSON-like documents** (called **BSON** – Binary JSON).

Each document is a flexible, schema-less structure that can contain arrays and nested documents, making MongoDB ideal for modern applications.

Key Features of MongoDB:

- **Document-Oriented**: Stores data in BSON documents, which can have different structures.
- **Schema-Less**: Each document can have its own unique structure (no predefined schema required).
- **Scalable**: Supports horizontal scaling through **sharding**.
- **High Performance**: Optimized for high throughput and low latency.
- **Indexing**: Supports secondary indexes, geospatial indexing, text search, etc.
- **Aggregation Framework**: Powerful data processing capabilities (like SQL's GROUP BY).

Use of MongoDB (When and Why to Use It)

MongoDB is used when:

1. **You need fast development cycles**:
    - Schema-less structure allows developers to iterate quickly.
2. **Data structures are complex and evolving**:
    - Suitable for hierarchical data (like JSON APIs, user profiles, etc.).
3. **You need to handle large volumes of data**:
    - Efficient for big data applications and analytics.
4. **Scalability is important**:
    - Easily scales horizontally by adding more servers (sharding).
5. **Real-time analytics and high write loads**:
    - Great for logging, analytics, and real-time dashboards.

Common Use Cases:

- Content management systems (CMS)
- E-commerce product catalogs
- IoT applications
- Mobile and web apps
- Real-time analytics platforms
- User and session data storage
- Chat/messaging applications

MongoDB vs SQL Databases:

Feature	MongoDB	SQL (e.g., MySQL, PostgreSQL)
Data Model	Document (JSON/BSON)	Table (Rows and Columns)
Schema	Flexible (Schema-less)	Fixed Schema
Joins	Limited	Powerful (JOINs supported)
Scalability	Horizontally (Sharding)	Vertically (Scaling up)
Best Use Case	Unstructured/complex data	Structured data

What is a Database and a Collection in MongoDB?

- **Database**: A container for collections. Think of it like a folder.
- **Collection**: A group of MongoDB **documents**. Similar to a table in SQL, but schema-less.
- **Document**: A single record in a collection, stored in **BSON** (binary JSON).

🧠 Example hierarchy:

```
Database: myDB
  └── Collection: users
        └── Document: { name: "Alice", age: 25 }
```

Here's a quick and clear guide to **basic MongoDB queries** for:

- Insert
- Update
- Display (find)
- Delete
- Filter

🔷 1. Insert Data

➤ Insert a Single Document

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

➤ Insert Multiple Documents

```
db.users.insertMany([
  { name: "Bob", age: 30, city: "London" },
  { name: "Charlie", age: 28, city: "Paris" }
]);
```

🔷 2. Display Data (Find)

➤ Display All Documents

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

➤ Pretty Print the Result (in shell)

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

🔷 3. Filter Documents (Find with Condition)

➤ Find One Matching Document

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

➤ Find All Where Age > 25

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

➤ Find Where City is "London"

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

➤ Multiple Conditions (AND)

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

➤ OR Condition

```
db.users.find({ $or: [ { city: "London" }, { city: "Paris" } ] });
```

🔷 4. Update Data

➤ Update One Document (change age of Alice)

```
db.users.updateOne(
  { name: "Alice" },
  { $set: { age: 26 } }
);
```

➤ Update Multiple Documents (set city to 'Berlin' where age > 28)

```
db.users.updateMany(
  { age: { $gt: 28 } },
  { $set: { city: "Berlin" } }
);
```

🔷 5. Delete Data

➤ Delete One Document

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

➤ Delete Multiple Documents

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

🔷 6. Additional: Sorting & Limiting

➤ Sort by Age (ascending)

```
javascript
CopyEdit
db.users.find().sort({ age: 1 });
```

➤ Sort by Age (descending)

```
db.users.find().sort({ age: -1 });
```

➤ Limit Results

```
db.users.find().limit(2);
```

📝 Notes:

- `db.collectionName` — Replace `collectionName` with your actual collection (e.g., `users`).
- Operators like `$gt`, `$lt`, `$or`, `$set`, etc., are part of MongoDB's **query language**.
- All examples assume you're using the **Mongo shell**, **MongoDB Compass**, or a MongoDB client like **Node.js**, **Python (PyMongo)**, etc.