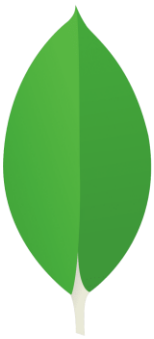
MongoDB Notes with Commands



mongoDB®

Muhammad Zohaib Irshad

Table of Contents

1. Introduction of MongoDB:

MongoDB is a NoSQL (Non-Relational) database that stores data in a JSON-like document format**.** Unlike traditional SQL databases (such as MySQL and PostgreSQL), MongoDB does not use tables and rows. Instead, it uses collections and documents**,** making it more flexible and scalable for modern applications.

2. Key Features of MongoDB:

NoSQL Database:

- No tables or rows like SQL databases.
- Uses JSON-like BSON (Binary JSON) format for storing data.

Schema-less:

- No predefined structure like SQL databases.
- Can store different types of data in the same collection.

Scalability:

- Supports horizontal scaling (sharding).
- Suitable for large-scale applications.

High Performance:

- Faster read and write operations compared to relational databases.
- Indexing improves query performance.

Flexible Data Model:

- Easily store complex, nested data structures.
- No need for complex JOIN operations.

Aggregation Framework:

- Provides powerful analytics and reporting tools.
- Uses pipelines to process large datasets efficiently.

Built-in Replication & High Availability:

- Data can be replicated across multiple servers.
- Ensures failover and redundancy.

3. MongoDB Architecture:

Databases

- Similar to a SQL database.
- Stores multiple collections.

Collections

- Similar to SQL tables.
- Stores multiple documents.

Documents

- Similar to rows in SQL.
- Stored in JSON-like BSON format.

4. Example of MongoDB Document (Record):

```
{
    "_id": ObjectId("64b5c9e9d4f7b1234b9f4c1a"),
    "name": "John Doe",
    "age": 30,
    "disease": "Diabetes",
    "medications": ["Metformin", "Insulin"],
    "doctor": { "name": "Dr. Smith", "specialization":
"Endocrinology" }
  }
```

Same Data in SQL Format (Table-Based):

id	Name	age	Disease	medications	doctor_name	doctor_specialization
1	John	38	Diabetes	Metformin, Insulin	Dr. Smith	Endocrinology

5. How MongoDB works:

Document-Oriented Storage:

- Data is stored in flexible JSON-like BSON format.
- Documents inside a collection can have different structures.

Indexing:

- MongoDB automatically creates an index on the `_id` field.
- You can manually add indexes for faster query performance.

Replication:

- Uses Replica Sets for high availability.
- Multiple copies of data on different servers.

Sharding:

- Distributes data across multiple servers.
- Ensures horizontal scaling for large datasets.

Aggregation Framework:

- Performs complex queries, analytics, and transformations.
- Uses aggregation pipelines.

6. Use Cases of MongoDB:

- **Medical Records Management**:

  Store patient records, appointments, and medical history.

- **E-Commerce Platforms**:

  Product catalogs, customer orders, and reviews.

- **Real-Time Analytics**:

  Process large amounts of data quickly.

- **Content Management Systems (CMS)**:

Manage articles, blogs, and media content.

- **IoT & Sensor Data**:

  Store and analyze sensor readings efficiently.

7. MongoDB Editions:

MongoDB Community Edition (Free):

- Open-source, self-hosted database.
- Suitable for small to medium applications.

MongoDB Atlas (Cloud-based):

- Fully managed cloud database by MongoDB.
- Scalable and easy to deploy.
- Supports AWS, Azure, and Google Cloud.

MongoDB Enterprise Edition (Paid):

- Advanced security and monitoring features.
- Suitable for large-scale enterprise applications**.**

8. MongoDB Commands:

a) For creating new document:

Command	Impact on database
db.users.insertOne({ name: “zohaib” })	it will add a new document with a name of zohaib in the user collection.
db.users.insertMany([{ age: 18 }, { age: 20 }])	it will add two new documents with age of 18 and 20 in the users collection.

b) For finding document:

Command	Impact on database
db.users.find()	It will get all users
db.users.find({ name: “zohaib” })	It will get all users with the name zohaib
db.users.findOne({ name: “zohaib” })	It will get the first user with the name zohaib
db.users.countDocuments({ name: “zohaib” })	It will give the number of users with the name zohaib
db.users.find({}, { age: 0 })	It will get all users and return all columns except for age

c) For updating document:

Command	Impact on database
db.users.updateOne({ age: 40 }, { $set: { age: 24 } })	It will update the first user with an age of 40 to the age of 24
db.users.updateMany({ age: 12 }, { $inc: { age: 3 } })	It will update all users with an age of 12 by adding 3 to their age
db.users.replaceOne({ age: 12 }, { age: 13 })	It will replace the first user with an age of 12 with an object that has the age of 13 as its only field.

d) For deleting document:

Command	Impact on database
db.users.deleteOne({ age: 40 })	It will delete the first user with an age of 40
db.users.deleteMany({ age: 30 })	It will delete all users with an age of 12

e) Complex Commands:

i) For filtering object:

Command	Impact on database
db.users.find({ name: { $eq: “Uzaif” } })	It will give all users with the name Uzaif.
db.users.find({ name: { $ne: “ali” } })	It will give all users with a name other than ali.
db.users.find({ age: { $gt: 30 } })	It will give all users with an age greater than 30
db.users.find({ age: { $gte: 15 } })	It will give all users with an age greater than or equal to 15
db.users.find({ age: { $lt: 12 } })	It will give all users with an age less than 12
db.users.find({ age: { $lte: 15 } })	It will give all users with an age less than or equal to 15
db.users.find({ name: { $in: [“ali”, “zohaib”] } })	It will give all users with a name of ali and zohaib
db.users.find({ name: { $nin: [“ali”, “zohaib”] } })	It will give all users that do not have the name ali or zohaib
db.users.find({ $and: [{ age: 12 }, { name: “zohaib” }] })	It will give all users that have an age of 12 and the name zohaib
db.users.find({ age: 12, name: “zohaib” })	This is an alternative way to do the same thing.
db.users.find({ $or: [{ age: 12 }, { name: “zohaib” }] })	It will get all users with a name of zohaib or an age of 12

db.users.find({ name: { $not: { $eq: “zohaib” } } })	It will give all users with a name other than zohaib
db.users.find({ name: { $exists: true } })	It will give all users that have a name field
db.users.find({ $expr: { $gt: [“$balance”, “$debt”] } })	It will give all users that have a balance that is greater than their debt

ii) For updating object:

Command	Impact on database
db.users.updateOne({ age: 12 }, { $set: { name: “newName” } })	This command will update the name of the first user with the age of 12 to the value newname.
db.users.updateOne({ age: 12 }, { $inc: { age: 5 } })	It will add 5 to the age of the first user with the age of 12
db.users.updateMany({}, { $rename: { age: “old” } })	It will rename the field age to old for all users
db.users.updateOne({ age: 12 }, { $unset: { age: “” } })	It will remove the age field from the first user with an age of 12
db.users.updateMany({}, { $push: { friends: “zohaib” } })	It will add zohaib to the friends array for all users
db.users.updateMany({}, { $pull: { friends: “ali” } })	It will remove ali from the friends array for all users

f) Basic Commands:

Command	Impact on database
db.users.find().sort({ name: 1, age: -1 })	Get all users sorted by name in alphabetical order.
db.users.find().limit(2)	Only return the first 2 users
db.users.find().skip(4)	Skip the first 4 users when returning results.
Cls	It will clear the mongo shell
Mongosh	It will open the connection to your local mongodb instance.
show dbs	It will show all the present databases
use databaseName	It will use the named database
Db	It will show current database name.
show collections	It will show all the collections in the current database.
db.dropDatabase()	It will delete the current database
exit	It will exit the current mongo shell session.