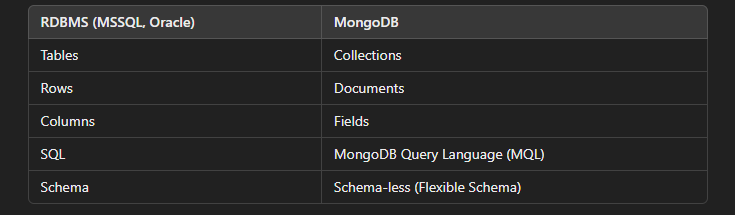
# MongoDB Learning Notes

## Introduction

MongoDB is a NoSQL database that stores data in a flexible, JSON-like format. Unlike traditional relational databases, it uses collections and documents instead of tables and rows. This learning journey covers key concepts, querying techniques, and optimization strategies.

## LESSON 1: Introduction to MongoDB

MongoDB is a document-oriented database that provides a high degree of flexibility for managing data.



MongoDB architecture : databases , collections and document

1. Database

A database in MongoDB is a container that holds collections of documents.It is similar to a schema in a relational database.

use mydb # Create or switch to a database named 'mydb'

2. Collection {Tables}

A collection is a group of related documents, similar to a table in an RDBMS.Collections are created dynamically when you insert a document, though you can also create them explicitly

db.createCollection("students") # Create a collection named 'students'

3. Document{Rows}

A document is a single record stored in a collection, similar to a row in an RDBMS.Documents are stored in BSON (Binary JSON) format, a binary-encoded version of JSON.Each document is a key-value pair structure, making it highly flexible.Unlike rows in relational databases, documents in the same collection can have different fields.

eg: {

"\_id": ObjectId("67502f28c9423970be239d3f"), // Unique identifier (automatically generated if not provided)

"name": "Alice",

"age": 21,

"course": "Mathematics"

}

**\_id:** A unique identifier for the document, similar to a primary key in RDBMS. If you don’t specify it, MongoDB generates it automatically.

KeyConcepts : BSON,JSON,CRUD  
**BSON**

BSON stands for Binary JSON.It is a binary-encoded serialization format used by MongoDB to store documents and make data transmission efficient.BSON extends JSON by adding support for more data types, such as:

- ObjectId (MongoDB's unique identifier)

- Date (stored as a 64-bit integer)

- Binary data (e.g., images, files)

- Arrays and nested objects

**JSON**

MongoDB uses JSON-like syntax for querying, but stores the data internally in BSON.

**CRUD** Operations : The fundamental database operations: Create, Read, Update, Delete.

CRUD Operations

(i) **Create** (Insert Data)

Adds new documents to a collection.

Methods:

- insertOne(): Inserts a single document.

- insertMany(): Inserts multiple documents.

// Insert a single document

db.students.insertOne({

name: "Alice",

age: 21,

course: "Mathematics"

})

// Insert multiple documents

db.students.insertMany([

{ name: "Bob", age: 22, course: "Physics" },

{ name: "Charlie", age: 23, course: "Chemistry" }

])

(ii) **Read** (Retrieve Data)

Retrieves documents from a collection.

Methods:

- find(): Retrieves documents based on a filter.

- findOne(): Retrieves a single document.

Example:

// Retrieve all documents

db.students.find()

// Retrieve documents with age > 21

db.students.find({ age: { $gt: 21 } })

(iii) Update (Modify Data)

Updates existing documents in a collection.

Methods:

- updateOne(): Updates a single document.

- updateMany(): Updates multiple documents.

Example:

// Update a single document

db.students.updateOne(

{ name: "Alice" }, // Filter

{ $set: { age: 22 } } // Update

)

// Update multiple documents

db.students.updateMany(

{ age: { $lt: 23 } }, // Filter

{ $set: { course: "Data Science" } } // Update

)

(iv) Delete (Remove Data)

Deletes documents from a collection.

Methods:

- deleteOne(): Deletes a single document.

- deleteMany(): Deletes multiple documents.

Example:

// Delete a single document

db.students.deleteOne({ name: "Bob" })

// Delete multiple documents

db.students.deleteMany({ age: { $lt: 23 } })

## LESSON 2: Querying and Aggregations

1. Querying:

Querying is the process of retrieving specific data from a MongoDB database based on certain conditions

It’s similar to the SELECT statement in SQL.

db.collectionName.find(

<filter>, // Filter to select documents

<projection> // Fields to include or exclude in the result

)

Components of Querying:  
**Filtering:** Used operators like $gt, $lt, $eq to retrieve documents based on conditions.  
**Projection:** Selected specific fields to include or exclude in results.  
**Sorting** : Orders the results in ascending or descending order.

db.cars.find().sort({ price: -1 })

2. Aggregation Framework:  
**$match:** Filters documents based on specified conditions.

db.cars.aggregate([

{ $group: { \_id: "$make", totalPrice: { $sum: "$price" } } }

])

**$group:** Groups documents and applies aggregations like $sum and $avg.

db.cars.aggregate([

{ $group: { \_id: "$make", totalPrice: { $sum: "$price" } } }

])

**$project:** Shapes the output by including or excluding fields.

db.cars.aggregate([

{ $project: { make: 1, model: 1, price: 1, \_id: 0 } }

])

**$limit** **& $Skip**: Limits the number of documents returned and skips a specified number of documents.

db.orders.aggregate([

{$skip: 5 }, // Skip the first 5 documents.

{$limit: 5 } // Limit the result to the next 5 documents.

])

3. Joining Collections:  
Used **$lookup** to perform an inner join between collections.

db.orders.aggregate([{$lookup:{

from:"customers",

localField:"customerId",

foreignField : "\_id",

as:"CustomerDetails"}

}

])

## LESSON 3: Indexing and Performance

Indexing is a critical feature for optimizing query performance in MongoDB.

* To find the executiontime and stats:

db.cars.find({Make:'Ford'}).explain("executionStats")

1. To creating index :

db.cars.createIndex({Make:1})

1. To get the indexes of Collection :

db.cars.getIndexes()

[

{ v: 2, key: { \_id: 1 }, name: '\_id\_' },

{ v: 2, key: { Make: 1 }, name: 'Make\_1' }

]

1. To use text index to search:

db.cars.createIndex({ Make: "text" })

//Make\_text index created

db.cars.find({ $text: { $search: "Ford" } })

1. To drop index:

db.cars.dropIndex("Make\_1")

1. To create Index in MongoDB compass:

1.go to index tab

2.Click Create index

3.Select field and and type

Explain()- COLLSCAN',means no index used and IXSCAN means index used.

Indexes are special data structures that store a small portion of the collection's data set in an easy-to-traverse form. MongoDB indexes use a B-tree data structure.

Without indexes, MongoDB would have to scan all the documents in a collection to fulfill a query, which can be very slow for large collections.

**Types of Indexes in MongoDB**

1. **Single-Field Indexes:** These indexes are created on a single field in a document.

Use Case: Use for queries that filter or sort based on a single field.

db.cars.createIndex({ make: 1 }) // Ascending order index on the 'make' field

1. **Compound Indexes**: These indexes are created on multiple fields. They are useful when you frequently query using multiple fields. **Use Case**: Use for queries that filter or sort by multiple fields. db.cars.createIndex({ make: 1, model: 1 }) // Compound index on 'make' and 'model'.
2. **Text Indexes**: These indexes are used for full-text search on string fields. **Use Case**: Use for text search queries where you search for specific words or phrases in strings. db.cars.createIndex({ description: "text" }) // Text index on 'description' field

**Positive Impacts**:

* Faster read operations and query execution.
* Better performance for sorting and aggregations.
* Improved efficiency in finding documents based on indexed fields.

**Negative Impacts**:

* Increased disk space usage for storing indexes.
* Slower write operations (insert, update, delete) due to index updates.

To check query performance analysis can be checked by explain() in shell or gui of mongoDB compass.

## LESSON 4: Backup and Security

Backup : To restore the data from the backup and the backed up regular intervals.

Creates a BSON dump of your data.

1.mongodump --db sampleAuctionDB "C:\Users\anvar\Desktop\Mongo\_DB\MONGODB\sampleAuctionDB"

2.db.orders.drop()

3.db.orders.find().pretty() // no result will be there

4.do mongo restore particular collection

5.mongorestore --db sampleAuctionDB --collection orders "C:\Users\anvar\Desktop\Mongo\_DB\MONGODB\sampleAuctionDB\orders.bson"

6.db.orders.find().pretty() //then,we will get the value

**Authentication**

Authentication ensures that only **legitimate users** can connect to your database. It involves verifying a user's identity before granting them access to the database.

**To Enable Authentication**  
Authentication is disabled by default in MongoDB. To enable it, modify the MongoDB configuration file:

* Edit the MongoDB configuration file (mongod.conf) and add:

security:

authorization: enabled

* **Restart MongoDB** Restart the MongoDB service after enabling authentication.

net stop MongoDB

net start MongoDB

* **Create MongoDB Users**

use admin

db.createUser({ user: "admin", pwd: "admin", roles: [ { role: "root", db: "admin" } ] })

* **Log in as Admin User**

mongosh -u "admin" -p "admin" --authenticationDatabase "admin" //modern-way

or

db.auth("admin","admin")