**MongoDB Basic Commands**

**Database Commands**

* **Show Databases**: show dbs
* **Switch/Create Database**: use <database\_name>
* **Show Current Database**: db

**Collection Commands**

* **Show Collections**: show collections
* **Create Collection**: db.createCollection("<collection\_name>")
* **Drop Collection**: db.<collection\_name>.drop()

**CRUD Operations**

* **Insert Documents**:
  + Single Document: db.<collection\_name>.insertOne({ <document> })
  + Multiple Documents: db.<collection\_name>.insertMany([{ <document1> }, { <document2> }])
* **Read Documents**:
  + All Documents: db.<collection\_name>.find()
  + With Filter: db.<collection\_name>.find({ <filter> })
  + Pretty Print: db.<collection\_name>.find().pretty()
* **Update Documents**:
  + Single Document: db.<collection\_name>.updateOne({ <filter> }, { $set: { <field>: <value> } })
  + Multiple Documents: db.<collection\_name>.updateMany({ <filter> }, { $set: { <field>: <value> } })
* **Delete Documents**:
  + Single Document: db.<collection\_name>.deleteOne({ <filter> })
  + Multiple Documents: db.<collection\_name>.deleteMany({ <filter> })

**Indexing**

* **Create Index**: db.<collection\_name>.createIndex({ <field>: 1 })
* **View Indexes**: db.<collection\_name>.getIndexes()
* **Drop Index**: db.<collection\_name>.dropIndex("<index\_name>")

**Aggregation**

* **Basic Aggregation**: db.<collection\_name>.aggregate([{ <pipeline\_stage> }])

**Database Commands**

1. **Show All Databases**:

show dbs

Output:

admin 0.000GB

local 0.000GB

test 0.000GB

1. **Switch or Create a Database**:

use my\_database

Output:

switched to db my\_database

1. **Show Current Database**:

db

Output:

my\_database

**Collection Commands**

1. **Show All Collections in the Database**:

show collections

1. **Create a Collection**:

db.createCollection("users")

Output:

{ "ok" : 1 }

1. **Drop a Collection**:

db.users.drop()

Output:

true

**CRUD Operations**

**Insert Operations**

1. Insert a Single Document:

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

Output:

{

"acknowledged" : true,

"insertedId" : ObjectId("...")

}

1. Insert Multiple Documents:

db.users.insertMany([

{ name: "Bob", age: 30, city: "Chicago" },

{ name: "Charlie", age: 35, city: "San Francisco" }

])

**Read Operations**

1. Find All Documents:

db.users.find()

1. Find Documents with Filter:

db.users.find({ city: "Chicago" })

1. Pretty Print Documents:

db.users.find().pretty()

**Update Operations**

1. Update One Document:

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

1. Update Multiple Documents:

db.users.updateMany({ city: "New York" }, { $set: { city: "Brooklyn" } })

**Delete Operations**

1. Delete One Document:

db.users.deleteOne({ name: "Charlie" })

1. Delete Multiple Documents:

db.users.deleteMany({ city: "Brooklyn" })

**Indexing**

1. **Create an Index**:

db.users.createIndex({ name: 1 })

1. **View All Indexes**:

db.users.getIndexes()

1. **Drop an Index**:

db.users.dropIndex("name\_1")

**Aggregation**

1. **Basic Aggregation Example**:

db.users.aggregate([

{ $match: { age: { $gte: 30 } } },

{ $group: { \_id: "$city", averageAge: { $avg: "$age" } } }

])

### 1. ****Basic Update****

Use the updateOne() or updateMany() methods to modify documents. Here's how they work:

#### Example: Updating a single document

db.collection.updateOne(

{ filterCondition },

{ $set: { fieldName: "newValue" } }

);

#### Example: Updating multiple documents

db.collection.updateMany(

{ filterCondition },

{ $set: { fieldName: "newValue" } }

);

### 2. ****Operators for Updates****

MongoDB provides a range of operators to update documents. Some commonly used ones are:

#### ****$set****

Updates the value of a field.

{ $set: { fieldName: "newValue" } }

#### ****$unset****

Removes a field from a document.

{ $unset: { fieldName: "" } }

#### ****$inc****

Increments the value of a field by a specified amount.

{ $inc: { fieldName: 1 } }

#### ****$push****

Adds an element to an array.

{ $push: { arrayField: "newElement" } }

#### ****$pull****

Removes elements from an array that match a condition.

{ $pull: { arrayField: "valueToRemove" } }

#### ****$rename****

Renames a field.

{ $rename: { oldFieldName: "newFieldName" } }

### 3. ****Replace Document****

You can replace an entire document using replaceOne().

#### Example:

db.collection.replaceOne(

{ filterCondition },

{ field1: "value1", field2: "value2" }

);

### 4. ****Example Use Case****

#### Update a user's email in a users collection:

db.users.updateOne(

{ username: "johndoe" },

{ $set: { email: "john.doe@example.com" } }

);

#### Increment a product's stock count:

db.products.updateMany(

{ category: "electronics" },

{ $inc: { stock: 10 } }

);

#### Add a tag to a blog post:

db.blogs.updateOne(

{ title: "MongoDB Basics" },

{ $push: { tags: "database" } }

);

### 5. ****Upsert Option****

Use upsert: true to insert a document if no matching document is found.

#### Example:

db.collection.updateOne(

{ uniqueField: "value" },

{ $set: { fieldName: "newValue" } },

{ upsert: true }

);

To get the count of documents in a MongoDB collection using **mongosh**, you can use the following methods:

### ****1. Using**** countDocuments() ****Method****

This method provides the count of documents that match a specific filter.

#### Example: Count all documents

db.collection.countDocuments();

#### Example: Count documents with a condition

db.collection.countDocuments({ fieldName: "value" });

### ****2. Using**** estimatedDocumentCount() ****Method****

This method gives an **approximate count** of all documents in a collection. It is faster but may not be 100% accurate, especially during ongoing writes.

#### Example:

db.collection.estimatedDocumentCount();

### ****3. Deprecated**** count() ****Method****

The older count() method is still available but is not recommended in newer versions of MongoDB. Use it only if working with legacy code.

#### Example:

db.collection.count({ fieldName: "value" });

### Use Case Examples

#### Count all users in a users collection:

db.users.countDocuments();

#### Count users with the role "admin":

db.users.countDocuments({ role: "admin" });

#### Estimate the number of documents in a large logs collection:

db.logs.estimatedDocumentCount();

**Delete Document**

To delete documents in a MongoDB collection using **mongosh**, you can use the following methods:

### ****1. Delete a Single Document****

Use the deleteOne() method to delete the first document that matches the filter criteria.

#### Syntax:

db.collection.deleteOne({ filterCondition });

#### Example:

Delete a user with the username "johndoe":

db.users.deleteOne({ username: "johndoe" });

### ****2. Delete Multiple Documents****

Use the deleteMany() method to delete all documents that match the filter criteria.

#### Syntax:

db.collection.deleteMany({ filterCondition });

#### Example:

Delete all users with the role "guest":

db.users.deleteMany({ role: "guest" });

### ****3. Delete All Documents****

To delete all documents in a collection, pass an empty filter {} to deleteMany().

#### Example:

db.collection.deleteMany({});

### ****4. Drop the Entire Collection****

If you want to completely remove the collection, including its schema and indexes, use the drop() method.

#### Example:

db.collection.drop();

### Use Case Examples

#### Remove all products in the electronics category:

db.products.deleteMany({ category: "electronics" });

#### Remove a single order with ID 12345:

db.orders.deleteOne({ orderId: 12345 });

#### Clear all documents from a logs collection:

db.logs.deleteMany({});

**MongoDB Compass**

MongoDB Compass is the official graphical user interface (GUI) for MongoDB, designed to help developers and database administrators visualize, manage, and query their data effectively.

**Key Features:**

* **Data Visualization:** Provides an intuitive view of collections, documents, and relationships.
* **Schema Analysis:** Automatically detects and displays the schema of your collections.
* **Query Building:** Allows users to create and execute queries using a visual query builder or raw MongoDB query syntax.
* **Aggregation Framework:** Simplifies the creation of aggregation pipelines with a user-friendly interface.
* **Performance Insights:** Helps analyze and optimize queries with detailed execution statistics.
* **CRUD Operations:** Easily create, read, update, and delete documents.

**MongoDB GUI**

MongoDB GUI refers to third-party or alternative tools designed for managing MongoDB databases visually. While Compass is the official GUI, other popular GUIs include **Studio 3T**, **Robo 3T**, and **NoSQLBooster**.

**Popular MongoDB GUIs:**

1. **Studio 3T:**
   * Advanced query builder.
   * Built-in SQL-to-MongoDB translator.
   * Migration tools and IntelliShell for developers.
2. **Robo 3T:**
   * Lightweight, open-source MongoDB client.
   * Supports shell integration for running queries.
   * Suitable for quick and simple tasks.
3. **NoSQLBooster:**
   * Provides IntelliSense for query editing.
   * Cross-platform support.
   * Supports SQL-like queries.

**Comparison:**

| **Feature** | **Compass** | **Studio 3T** | **Robo 3T** |
| --- | --- | --- | --- |
| Official Support | Yes | No | No |
| Schema Analysis | Yes | Yes | Limited |
| Aggregation Builder | Yes | Yes | No |
| Pricing | Free | Paid/Free Tier | Free/Open-Source |

**Summary:**

* **MongoDB Compass** is the go-to GUI for MongoDB users looking for robust and official support.
* Alternative GUIs like **Studio 3T** and **Robo 3T** offer unique features that may cater to different use cases or preferences.