MongoDB Tutorial (4th Week)

1. Switch to a Database and show collections within it

1.1 Switching or Creating a Database :

✅ The **use** command switches to the specified database. If the database doesn't exist, it switches to it **without creating it**. The database is only created when data is inserted.

```
use db_name
```

Where **db_name** is the name of the database you want to use.

- If **db_name** exists ➡️ switch to it.
- If **db_name** doesn't exist ➡️ switch to it, but create only when you insert data (collection) into it.

1.2 Show all available databases :

```
show dbs
```

✅ Displays the list of available databases.

Example output :

```
admin     40.00 KiB
config    70.00 KiB
local      2.66 MiB
```

✅ These databases are **system-created databases** that came by default when we installed MongoDB. They are part of MongoDB's internal structure, which is why they were created automatically during the installation.

Note: Dropping these databases can lead to loss of user authentication, sharding failures, and replication issues, which could make MongoDB unstable or unusable. Which is why **you should not drop** the system databases as they are critical for MongoDB's operation.

1.3 Show all collections in the current database:

```
show collections
```

✅ Displays the list of available collections in the currently selected database.

Example :

```
> use myDatabase
< switched to db myDatabase

myDatabase > show collections
```

✅ The code attempts to switch to a database named **myDatabase** and then **lists the collections** within that database.

2. Creating Collections and Inserting documents.

2.1 Creating a collection :

```
db.createCollection("myCollection")
```

✅ **myCollection** ➡️ Name of the collection you want to create.
✅ Creates a **new collection** named **"myCollection"** in the current database.

2.2 Insert a single document :

```
db.collectionName.insertOne({...})
```

collectionName ➡️ Name of the collection.

✅ Inserts a **single document** into the collection.
✅ If the collection does not exist, MongoDB will **automatically create** it when you insert the document using **insertOne()**.

Example code :

```
db.myCollection.insertOne(
   {
      "_id": "1001",
      "name": "Ram Sharma",
      "age": 21,
      "gender": "Male",
      "grades": [
         { "subject": "Computer Science", "score": 85 },
         { "subject": "Statistics", "score": 90 },
         { "subject": "Economics", "score": 78 }
      ],
      "address": {
         "permanent_address": "Kathmandu",
         "temporary_address": "Butwal"
      }
   }
)
```

The above code does the following :
✅ Uses **insertOne()** to insert a single document into the myCollection collection.
✅ If myCollection **does not exist**, MongoDB **will automatically create** the collection with the specified fields alongside their corresponding values.

2.3 Insert multiple documents :

```
db.collectionName.insertMany( [ { … }, { … }, { … } ] )
```

✅ Uses **insertMany()** to insert **multiple documents** into the collection.
✅ Inserts each document inside the array {...}, {...} into the collection.
✅ Each document should have a unique _id (if provided), otherwise MongoDB will generate one automatically.

Example code:

```
db.myCollection.insertMany([
 {
  "_id": "1002",
  "name": "Hari Adhikari",
  "age": 22,
  "gender": "Male",
  "grades": [
   { "subject": "Computer Engineering", "score": 92 },
   { "subject": "Data Structures", "score": 88 },
   { "subject": "Software Development", "score": 80 }
  ],
  "address": {
   "permanent_address": "Biratnagar",
   "temporary_address": "Butwal"
  }
 },
 {
  "_id": "1003",
  "name": "Shyam Thapa",
  "age": 23,
  "gender": "Male",
  "grades": [
   { "subject": "Application Software Development", "score": 95 },
   { "subject": "Data Science", "score": 85 },
   { "subject": "Networking", "score": 79 }
```

```
    ],
    "address": {
      "permanent_address": "Pokhara",
      "temporary_address": "Butwal"
    }
  },
  {
    "_id": "1004",
    "name": "Sita Rana",
    "age": 20,
    "gender": "Female",
    "grades": [
      { "subject": "Computer Networking", "score": 88 },
      { "subject": "Database Basics", "score": 82 },
      { "subject": "Web Development", "score": 90 }
    ],
    "address": {
      "permanent_address": "Butwal",
      "temporary_address": "Kathmandu"
    }}
])
```

The above code does the following :
✅ Uses **insertMany()** to insert multiple documents into the myCollection collection at once.
✅ If myCollection **does not exist**, MongoDB **will automatically create** the collection with the specified fields alongside their corresponding given values.

3. Read Operations

3.1 Find all documents :

```
db.myCollection.find({})
```

✅ The **empty {}** inside the find() method means no filter is applied, so it will retrieve all the documents in the collection named myCollection.
✅ It returns all the documents in the collection (myCollection) as a result.

3.2 Find documents with a specific condition :

- Find all female students :

```
db.myCollection.find({ gender: "Female" }).pretty()
```

✅ The **{ gender: "Female" } filter** inside the find() method means that it will retrieve all documents where the gender field is equal to **"Female"**.
✅ The **pretty()** method formats the output in a more readable, human-friendly format, making it easier to view documents with nested fields or arrays.

- Find students younger than 22 :

```
db.myCollection.find({ age: { $lt: 22 } })
```

✅ The **filter { age: { $lt: 22 } }** means it will retrieve all documents where the age field is less than 22.
✅ It will return all documents where the age value is strictly less than 22.

Note : **$ (dollar sign) is** used for query operators ($and, $gte, $lte, $or,$in), aggregation operator, update operator($set).

4. Find using OR and AND operators :

4.1 Find students older than 30 **OR** male students :

```
db.myCollection.find({
  $or: [
    { age: { $gt: 30 } },
    { gender: "Male" }
  ]
})
```

✅ **$or: [...]** ➡️ This is a logical operator used to match documents that satisfy either of the conditions inside the array.
✅ It will **return all documents** where the **age is greater than 30**, **or the gender**
is Male.
✅ So, even if the document doesn't meet the age > 30 condition, if it matches the gender condition, it will still be included.

4.2 Find students older than 22 **AND** male students :

```
db.myCollection.find({
  $and: [
    { age: { $gt: 22 } },
    { gender: "Male" }
  ]
})
```

✅ **$and:** [...] ➡️ This is a logical operator used to match documents that satisfy all of the conditions inside the array.
✅ It will **return all documents** where the **age is greater than 22**, **and the gender is Male**.
✅ So, the document must meet the age > 22 condition as well as the gender condition, to be included.

5. Find specific fields only :

5.1 Include specific fields **(exclude _id)** :

```
db.myCollection.find({ }, { name: 1, age: 1, _id: 0 })
```

—-- OR —--

```
db.myCollection.find({ }, { name: true, age: true, _id: false })
```

✅ Both queries are functionally the same and will give the same result. The choice of syntax depends on your preference.
✅ First argument **{ }** ➡️ An empty query filter, meaning it will match all documents in the collection.
✅ Second argument **{ name: 1, age: 1, _id: 0 }** ➡️ This is the projection :

- **name: 1** ➡️ means to **include the name** field.
- **age: 1** ➡️ means to **include the age** field.
- **_id: 0** ➡️ means to **exclude the _id** field (since MongoDB includes _id by default).

> ***Note:*** Only the specified fields will be included in the results. In other words, all other fields not explicitly listed will be **excluded** by default.

5.2 Include specific fields **(include _id)** :

```
db.myCollection.find( { }, { name: 1, age: 1 })
```

✅ Returns the **name and age fields along with _id.**
✅ By default, MongoDB includes _id, so no need to explicitly set it to 1.

6. Find using $in operator

$in is used to match documents where a field's value matches any value in a specified array.

Syntax :

```
db.collectionName.find( { field: { $in: [ value1, value2, value3 ] } })
```

✅ Use case ➡️ When you want to find documents where a specific field matches one of several possible values.

Example code :

```
db.myCollection.find({ age: { $in: [ 21, 23 ] } })
```

✅ The query will return all documents where the age field is **either 21 or 23**.
✅ Difference between **$in** and **$and** :

$in	$or
$in checks if a field's value matches any of the values in an array.	$or checks if at least one of multiple conditions is true.
Use $in when you're comparing a field to multiple possible values.	Use $or when you need to match multiple conditions on the same or different fields.

7. Update Operations

7.1 Update One Document :

✅ **updateOne()** ➡️ This method updates **a single document** that matches the specified filter criteria.

```
db.myCollection.updateOne(
  { age: 23 },
  { $set: { name: "John" } }
)
```

✅ First argument **{ age: 23 }** ➡️ The filter specifies that the operation should target a document where the age field is equal to 23.
✅ Second argument **{ $set: { name: "John" } }** ➡️ The update operation:

- **$set** is an **update operator** that sets the value of a specified field.
- **{ name: "John" }** updates the name field of the matched document to **"John"**.
- If the **name** field does not exist in the document, MongoDB will create it.

Note : If there are **multiple documents** where age = 23, only the first matching document (based on natural order) will be updated.

7.2 Update Multiple Documents :

```
db.myCollection.updateMany(
  { gender: "Male" },
  { $set: { "address.country": "UK" } }
)
```

✅ **updateMany()** ➡️ This method updates **all documents** that matches the specified filter criteria.
✅ First argument **{ gender: "Male" }** ➡️ The filter specifies that the operation should target all documents where the gender field is equal to **"Male"**.
✅ Second argument **{ $set: { "address.country": "UK" } }** ➡️ The update operation:

- **$set** is an **update operator** that sets the value of a specified field.
- **{"address.country" : "UK"}** updates (or creates if it doesn't exist) the country field inside the nested address object to "UK".
- If the **address** field itself doesn't exist, MongoDB will create the entire path (address.country).

7.3 Replace a document :

```
db.myCollection.replaceOne(
  { name: "John" },
  { name: "JOHN" }
)
```

✅ **replaceOne()** ➡️ method in MongoDB is used to replace a single document
in a collection that matches a specified filter.
✅ **{ name: "John" }** ➡️ This is the filter condition that finds the document where the name field equals "John".
✅ **{ name: "JOHN" }** ➡️ This is the **replacement document** that will completely replace the existing document that matches the filter.

Key Points to Remember :

✅ **replaceOne()** replaces the entire document (except for the _id field, which remains unchanged).
✅ The replacement document will only have the fields specified in the new object. Any fields not included will be **removed**.
✅ If no document matches the filter, nothing will be replaced.

8. Delete Operations

8.1 Delete One Document :

```
db.myCollection.deleteOne({ "_id": "1004" })
```

✅ **deleteOne()** ➡️ This method deletes a **single document** that matches the specified filter criteria.
✅ **Filter { "_id": "1004" }** ➡️ The filter specifies that the operation should target a document where the _id field is **"1004".**
✅ If a document with _id = "1004" exists, MongoDB will delete the **first matching document**.
✅ If **multiple documents match,** only the **first one** will be deleted.

8.2 Delete Multiple Documents :

```
db.myCollection.deleteMany({ age: { $lt: 20 } })
```

✅ **deleteMany()** ➡️ This method deletes **all documents that** matches the specified filter criteria.
✅ **Filter { age: { $lt: 20 } }** ➡️ The filter specifies that the operation should target documents where age field value is **less than 20** ($lt = "less than").

Note :

✅ When deleting a specific document, it's recommended to use the **_id field** as the filter because the _id field is **unique for each document**, so using it ensures that you are targeting exactly one document.

✅ This prevents the accidental deletion of multiple documents due to matching other fields like **name** or **age** which may not be unique.