MANGODB-LAB2 - CRUD OPERATIONS

- CRUD operation is one of the essential concepts of a database system.
- Inserting data in the database comes under one of the CRUD operations.
- If you do not insert data in your database, you will not be able to continue with other activities within your document.
- We learn about the different concepts and methods that are related to the insert operation in MongoDB.

MANGODB-LAB2 - CRUD OPERATIONS

- The insert operation is one of the crucial operations in the database system. MongoDB supports the below mentioned three methods to insert document data in your database:
- insert()
- insertOne()
- insertMany()

MANGODB-LAB2 - CRUD OPERATIONS

- The insert() method is used to insert one or multiple documents in a collection.
- The collection name is associated with the insert() method and the parameters.

db.movie.insert({"name":"Avengers: Endgame"}) db.movie.find()

MANGODB-LAB2 - CRUD OPERATIONS

- The syntax to insert a single document is shown below:
- Syntax :
- db.collection_Name.insert(JSON document)
- In the above syntax, the document will consist of { name: "data_value" }.

MANGODB-LAB2 - CRUD OPERATIONS

- As it is a JSON document, these documents will consist of the data as name-value pairs, surrounded by curly braces, i.e. {}
- db.movie.insert({"name":"Avengers: Endgame"}) db.movie.find()

MANGODB-LAB2 - CRUD OPERATIONS

- The **_id** which is provided by MongoDB is a 12-byte value of ObjectId which is prepared from the following values:

```
WriteResult({ "nInserted" : 1 })
> db.movie.find()
{ "_id" : ObjectId("5d10f858b0f4e87f32b2b2e8"), "name" :
"Avengers: Endgame" }
>
```

4-byte value denoting the seconds as Unix epoch,

3-byte device identifier value,

2-byte processing id,

3 byte counter which is a random value.

CREATING MULTIPLE DOCUMENTS USING THE SINGLE INSERT() COMMAND

- It is also possible for you to insert multiple document values in a particular insert() method.
- Let us take an example where you can insert multiple documents at a time
- Example :
- db.movie.insert(
- [
- { name: "Avengers: Infinity War" },
- { name: "Avengers: Endgame" }
-]
-)

CREATING MULTIPLE DOCUMENTS USING THE SINGLE INSERT() COMMAND

- It is to be noted that the documents are supplied in the form of an array. Document values are packed or enclosed in square brackets [] and separated by commas.
- Executing the above statements will pop up with messages something like this:

Output:

```
> db.movie.insert([{name:"Avengers: Infinity War"}
,{name:"Avengers: Endgame"}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
>
```

EMBEDDED DOCUMENTS

- MongoDB also allow users to create document containing other documents, arrays of values, as well as arrays of documents

Example:

```
db.writer.insert({
  writername: "Stan Lee",
  comics: [
    { comics: "DC Comics", year: 2004, name: "Superman" },
    { project: "DC Comics", year: 2001, level: "Batman" },
    { project: "Marvel Comics", year: 1968, level: "Captain America" }
  ]
})
```

EMBEDDED DOCUMENTS

- MongoDB also allow users to create document containing other documents, arrays of values, as well as arrays of documents

Output:

```
> db.writer.insert({
...   writername: "Stan Lee",
...   comics: [
...     { comics: "DC Comics", year: 2004, name: "Superman" },
...     { project: "DC Comics", year: 2001, level: "Batman" },
...     { project: "Marvel Comics", year: 1968, level: "Captain America" }
...   ]
... })
WriteResult({ "nInserted" : 1 })
> db.writer.find()
{ "_id" : ObjectId("5d1101a0b0f4e87f32b2b2f4"), "writername" : "Stan Lee", "comics" : [ {
 "comics" : "DC Comics", "year" : 2004, "name" : "Superman" }, { "project" : "DC Comics",
 "year" : 2001, "level" : "Batman" }, { "project" : "Marvel Comics", "year" : 1968, "leve
l" : "Captain America" } ] }
>
```

THE INSERTONE() METHOD

- Another way to insert documents is by using the insertOne() method for a single document in a collection:

Example:

```
db.movie.insertOne({ _id: 2, writername: "Stan Lee", name: "Aquaman" })
```

In this case, you have a particular non-existent collection of data. In the case of the insert() method, a precise collection will get produced in case it does not exist previously.

Here you will observe that the output appeared to be different in format than that of insert() method:

Output:

```
> db.movie.insertOne({ _id: 2, writername: "Stan Lee", name: "Aquaman" })
{ "acknowledged" : true, "insertedId" : 2 }
>
```

insertMany() Method

- As the name is explaining its working, is used for inserting multiple documents:

Example:

```
db.developers.insertMany(
[
  { _id: 20, devname: "John Wick", tools: "Visual Studio", born: 1948 },
  { _id: 21, devname: "Ganesh Roy", tools: "Net Beans", born: 1945 },
  { _id: 22, devname: "Deeksha Raul", tools: "Unity 3D", born: 1954 }
]
)
```

Output:

```
> db.developers.insertMany(
... [
...   { _id: 20, devname: "John Wick", tools: "Visual Studio", born: 1948 },
...   { _id: 21, devname: "Ganesh Roy", tools: "Net Beans", born: 1945 },
...   { _id: 22, devname: "Deeksha Raul", tools: "Unity 3D", born: 1954 }
... ]
... )
{ "acknowledged" : true, "insertedIds" : [ 20, 21, 22 ] }
>
```

MONGO DB - QUERY OPERATIONS

- Queries are another essential element of a database system. When your database is having all the data fed into it, and you want to retrieve the data by executing some command, MongoDB allows you to do that.

- We will learn about the different ways of how queries can be made using MongoDB.

What Is a Database Query?

- A query in a database system is a command that is used for extracting data from a database and display it in a readable form.
- Every query associated with the database system is associated with any particular language (such as SQL for structured data, MongoDB for unstructured data).
- Queries can be explained by taking a suitable example:

What Is a Database Query?

- Let us assume a situation where your database has an employee table, and you wish to track the sales performance ID, so you have to write a query to ask your database to fetch for you the list of all the sales performance with the highest sales data in the top.
- This is where the queries of a database language become useful.

Methods for Performing Queries in MongoDB

1. **The find() method:** This method is used for querying data from a MongoDB collection.

   The basic syntax for using this method is:

   Syntax:

```
db.collection_name.find()
```

   Example:

```
db.writers.find()
```

Methods for Performing Queries in MongoDB

Various other options can be used to make the query specific. It will be discussed below.

2. **The pretty() method:** This method is used for giving a proper format to the output extracted by the query. The basic syntax for using this method is:

Syntax:

```
db.collection_name.find().pretty()
```

Example:

```
db.writers.find().pretty()
```

Methods for Performing Queries in MongoDB

Here is how they can be implemented:

Filtering Criteria in MongoDB Queries

It is also possible to filter your results by giving or adding some specific criteria in which you are interested to. For example, if you wish to see the Gaurav Mandes data, you can add a specific attribute to the find() to fetch the data of Gaurav Mandes from that particular database.

Example:

```
db.writers.find( { author: "Gaurav Mandes" } )
```

Output:

```
> db.writers.find( { author: "Gaurav Mandes" } )
{ "_id" : ObjectId("5d16f6a1e198c897a4105d0d"), "title" : "Networking", "descriptio
n" : "Networking Essentials", "author" : "Gaurav Mandes" }
{ "_id" : ObjectId("5d16f6a1e198c897a4105d0e"), "title" : "Game Engineering", "desc
ription" : "Game Engineering and Development", "author" : "Gaurav Mandes" }
>
```

MongoDB Query Which Specify "AND" Condition

MongoDB also allows you in specifying data values of the documents holding two or more specified values to be fetched from the query. Here are two examples showing the use of specifying queries using AND.

Example:

```
db.writers.find( { tools: "Visual Studio", born: 1948} )
```

MongoDB Query Which Specify "OR" Condition

MongoDB allows users to specify either one or multiple values to be true. According to this, till one of the conditions is true, the document data will get returned. Here is an example showing the use of OR condition:

Example:

```
db.musicians.find({$or: [ { instrument: "Drums" }, { born: 1945 } ] } )
```

Output:

```
> db.musicians.find({$or: [ { instrument: "Drums" }, { born: 1945 } ]})
{ "_id" : 2, "name" : "Ian Paice", "instrument" : "Drums", "born" : 1948 }
{ "_id" : 3, "name" : "Roger Glover", "instrument" : "Bass", "born" : 1945 }
{ "_id" : 7, "name" : "Jeff Burrows", "instrument" : "Drums", "born" : 1968 }
>
```

$in operator

The $in operator is another special operator used in queries for providing a list of values in the query. When your document holds any of those provided values, it gets returned. Here is an example:

Example:

```
db.musicians.find( { "instrument": { $in: [ "Keyboards", "Bass" ] } } )
```

Output:

```
> db.musicians.find( { "instrument": { $in: [ "Keyboards", "Bass" ] } } )
{ "_id" : 3, "name" : "Roger Glover", "instrument" : "Bass", "born" : 1945 }
{ "_id" : 5, "name" : "Don Airey", "instrument" : "Keyboards", "born" : 1948 }
>
```

MongoDB Projection Queries

- Various other features can be used to filter out our queries in a more precise manner.
- MongoDB allows its users to add more features to a query for extracting data from the Mongo database.
- We learn about MongoDB Projection Queries which can be used for an additional purpose.

Projection Queries

- Protection queries are a particular type of MongoDB queries where you can specify fields you want to get in the output.
- MongoDB allows you to perform a query for a collection by using the db.collection.
- find() method, where you have to mention the field that needs to be explicitly returned.

Projection Queries

- This can be done by explicitly incorporating the field names in your query, and adding a 1 or 0 with them for specifying whether this needs to be returned or not.
- Such kinds of parameters are called projection parameter.
- When a projection parameter is associated with a value 1, it will show the value according to the query and hide when the projection parameter has a value 0

Projection Queries

Without Projection

Here is an example where the projection parameter is not used:

Example:

```
db.writers.find()
```

Output:

```
> db.writers.find().pretty()
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0d"),
        "title" : "Networking",
        "description" : "Networking Essentials",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0e"),
        "title" : "Game Engineering",
        "description" : "Game Engineering and Development",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0f"),
        "title" : "PHP",
        "description" : "PHP as a General-Purpose",
        "author" : "Gautam"
}
>
```

Projection Queries

With Projection

Another example where the projection parameter is used:

Example:

```
db.writers.find( { "author": "Gaurav Mandes" }, { _id:0,  author:1, title:1 } )
```

Output:

```
> db.writers.find( { "author": "Gaurav Mandes" }, { _id:0,  author:1, title:1 } )
{ "title" : "Networking", "author" : "Gaurav Mandes" }
{ "title" : "Game Engineering", "author" : "Gaurav Mandes" }
>
```

In the example above, the **_id** field is excluded, which automatically gets added, and the **title** and **author** fields are displayed.

MongoDB – Limiting Query result

- MongoDB allows you to specify the maximum number of documents to return by making use of the limit() method which will return only the number of documents you need.
- And as soon as you prepare a MongoDB query for the collection with the help of db.collection.
- find() method, you can add on the limit() method for specifying the limit.

MongoDB – Limiting Query result

Without Limit

Here is an example where the Limit method is not used:

Example:

```
db.writers.find().pretty()
```

Output:

```
> db.writers.find().pretty()
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0d"),
        "title" : "Networking",
        "description" : "Networking Essentials",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0e"),
        "title" : "Game Engineering",
        "description" : "Game Engineering and Development",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0f"),
        "title" : "PHP",
        "description" : "PHP as a General-Purpose",
        "author" : "Gautam"
}
>
```

In the above example, you can see that three results are showing as output.

With Limit

Another example where the Limit method is used:

Example:

```
db.writers.find().pretty().limit(2)
```

Output:

```
> db.writers.find().pretty().limit(2)
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0d"),
        "title" : "Networking",
        "description" : "Networking Essentials",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0e"),
        "title" : "Game Engineering",
        "description" : "Game Engineering and Development",
        "author" : "Gaurav Mandes"
}
>
```

In the above example where the limit() method is used, you can see that only **two results** are seen in the form of output because we have passed the parameter in the limit() method to display only two records.

Skipping Documents

It is also possible to skip some documents from a MongoDB database. You can perform such operations using the **skip()** method of MongoDB. In other words, it can be said that users have the power to manage or regulate where MongoDB begins returning the query results.

Example:

```
db.writers.find().pretty().skip(1)
```

Output:

```
> db.writers.find().pretty().skip(1)
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0e"),
        "title" : "Game Engineering",
        "description" : "Game Engineering and Development",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0f"),
        "title" : "PHP",
        "description" : "PHP as a General-Purpose",
        "author" : "Gautam"
}
>
```

In the above example where the skip() method is used, you can see that only two results are seen in the form of output because we have skipped the first result.