LAB MANUAL

EDUNET FOUNDATION - Class Exercise Notebook

Lab 2 – CRUD Operations in MongoDB

Exercise 1: Creating database in MongoDB

Step 1: To create a database, MongoDB provides use command. Syntax of use command is:
use DATABASE_NAME
let's create our first database by typing this command in the terminal:

```
use mydb
```

Output:

```
test> use mydb
switched to db mydb
mydb>
```

Step 2: List all database in MongoDB Server
To get the list of databases in your MongoDB server, MongoDB provides show databases and show dbs commands.

```
mydb> show dbs
admin   40.00 KiB
config  72.00 KiB
local   40.00 KiB
mydb>
```

Note that if you had created a database using the use command, the above commands will not show the database unless it has some data or documents added to it.
Step 3: Creating Collection in MongoDB

MongoDB provides the db.createCollection() method to accomplish this task. Syntax is:
db.createCollection(“collection_name”)

```
mydb> db.createCollection("my_test_collection")
{ ok: 1 }
mydb>
```

Step 4: Now you can type `show dbs` to see all the databases, including the new one you just created.

```
mydb> show dbs
admin   40.00 KiB
config  96.00 KiB
local   40.00 KiB
mydb     8.00 KiB
mydb>
```

Exercise 2: Creating and dropping collection in MongoDB.
Step 1: Lets create another collection using createCollection() method.

```
mydb> db.createCollection("my_second_collection")
{ ok: 1 }
mydb>
```

Step 2: list all collections
Mongodb provides show collections command to enlist all collections inside database.

```
mydb> show collections
my_second_collection
my_test_collection
mydb>
```

We can see two collections inside mydb database.
Step 3: Dropping Collections
Existing collections can be removed with the drop method. Syntax is :

db.collection_name.drop()

```
mydb> db.my_second_collection.drop()
true
```

Step 4: Now you can type show collections to enlist all collections in mydb database.

```
mydb> show collections
my_test_collection
mydb>
```

You can see only one collection exist.

Exercise 3: Inserting Single and multiple documents into MongodB.

Inserting single document:

To insert single document into collection mongodb provides InsertOne() method. If the collection does not exist, then insertOne() method creates the collection.

Syntax of insertOne method is :

db.collection.insertOne(document, options)

```
test> use car_db
switched to db car_db
car_db> db.car.insertOne({"car_name":"Alto","company":"Maruti","Price":420000})
{
  acknowledged: true,
  insertedId: ObjectId("6476e1492e5ff6faedba39a3")
}
```

Inserting Multiple documents:

To insert Multiple document into collection mongodb provides InserMany() method.

Syntax of insertMany() method is :

db.collection.insertMany(document, options)

```
car_db> db.car.insertMany([{"car_name":"Scorpio","company":"Mahindra","Price":1600000},{"car_name"
:"Brezza","company":"Maruti","Price":1200000}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('664f1771a929ceaef3cdcdf8'),
    '1': ObjectId('664f1771a929ceaef3cdcdf9')
  }
}
car_db>
```

Exercise 4: Reading Single and multiple documents into MongodB.

Reading Single document:

MongoDB findOne() method returns only one document that satisfies the criteria entered. If the criteria entered matches for more than one document, the method returns only one document according to natural ordering.

Syntax of findOne() method is:

db.collection.findOne(<criteria>, <projection>)

```
car_db> db.car.findOne()
{
  _id: ObjectId('664f160aa929ceaef3cdcdf6'),
  car_name: 'Alto',
  company: 'Maruti',
  Price: 420000
}
car_db>
```

Reading Multiple documents:

To get all the documents in a collection, we need to use the find method with an empty parameter.

```
test> use car_db;
switched to db car_db
car_db> db.car.find()
[
  {
    _id: ObjectId("6476e1492e5ff6faedba39a3"),
    car_name: 'Alto',
    company: 'Maruti',
    Price: 420000
  },
  {
    _id: ObjectId("6476e2572e5ff6faedba39a4"),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  },
  {
    _id: ObjectId("6476e2572e5ff6faedba39a5"),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  }
]
```

Exercise 5: Filtering documents into MongodB.

You can filter query results in MongoDB by defining a specific condition that documents must adhere to in order to be included in a result set.

Suppose we want to retrieve details of car Scorpio. Then we to have to query inside find() or findOne() method.

```
car_db> db.car.find({"car_name":"Scorpio"})
[
  {
    _id: ObjectId("6476e2572e5ff6faedba39a4"),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  }
]
```

Exercise 6: Selecting documents Fields Using Projection Queries

MongoDB Projection is a special feature allowing you to select only the necessary data rather than selecting the whole set of data from the document. For Example, If a Document contains 10 fields and only 5 fields are to be shown the same can be achieved using the Projections MongoDB projection operator will increase the performance of the database server.

A sample Syntax that you can use to retrieve the limited amount of data using Projection in MongoDB can be written as follows:

db.DATA_COLLECTION_NAME.find({}, {YOUR_FIELD_KEY: BOOLEAN})

Let us break down this syntax to understand everything in detail.

- Here "DATA_COLLECTION_NAME" is the name of the table/Collection from where the records are to be retrieved for processing.
- YOUR_FIELD_KEY is the name of the column or data item that you want to process from the Collection
- BOOLEAN is the check to show or hide the Field value.

```
car_db> db.car.find({},{"car_name":1})
[
  { _id: ObjectId('664f160aa929ceaef3cdcdf6'), car_name: 'Alto' },
  { _id: ObjectId('664f1771a929ceaef3cdcdf8'), car_name: 'Scorpio' },
  { _id: ObjectId('664f1771a929ceaef3cdcdf9'), car_name: 'Brezza' }
]
car_db>
```

Suppose if you pass “car_name”:0 then it will exclude car_name field.

```
car_db> db.car.find({},{"car_name":0})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    company: 'Mahindra',
    Price: 1600000
  },
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    company: 'Maruti',
    Price: 1200000
  },
  {
    _id: ObjectId('664f5630a929ceaef3cdcdfb'),
    company: 'Maruti',
    Price: 420000
  }
]
```

Exercise 7: Update documents in MongoDB

MongoDB provides the updateOne() and updatemany() method to update the documents of a collection. To update only the specific documents, you need to add a condition to the update statement so that only selected documents are updated.

Syntax: db.collection.update(query, update, options)

Where:

query: Specify the condition to be used to update the document. In the below example, we need to update the document which has car_name is Alto.

Use the $set operator to update the values

Choose the Field Name that needs to be updated and enter the new values accordingly

```
car_db> db.car.updateOne({"car_name":"Alto"},{$set:{"price":450000}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

```
car_db> db.car.find({"car_name":"Alto"})
[
  {
    _id: ObjectId('664f5064a929ceaef3cdcdfa'),
    car_name: 'Alto',
    company: 'Maruti',
    Price: 450000
  }
]
car_db>
```

Exercise 8: Deleting Documents in MongoDB

To delete documents from mongodb database, MongoDB provides deleteOne() and deleteMany().

deleteOne(): This method deletes or removes the single document from the collection that matches the given query clause. It deletes only one document even if conditions match more than one document.

Syntax is : db.collection.deleteOne({ <field>:<value> })

```
car_db> db.car.deleteOne({"car_name":"Alto"})
{ acknowledged: true, deletedCount: 1 }
car_db>
```

Now you can type again find() method to query all documents.

```
car_db> db.car.find()
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  },
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  }
]
car_db>
```

You can see that document is removed from collections.

The deleteMany() method

This method deletes or removes the multiple documents from the collection that matches the given query clause. If we want to delete all the documents, we use the filter parameter deleteMany()without query condition.

Exercise 9: Demonstrate different types of data types in MongoDB.

Step1: Switch to new database by use command

```
car_db> use employee
switched to db employee
```

```
employee>
db.employee_details.insertOne({"employee_name":"Vishal","employee_salary":60000,"skills":["
PHP","Python","Java"],"address":{"house_no":1001,"city":"Ahmedabad","state":"Gujarat"},"emp
loyee_status":true})
```

Step 2: Insert details of one employee using insert one() method

```
employee> db.employee_details.insertOne({"employee_name":"Vishal","employee_salary":60000,"
skills":["PHP","Python","Java"],"address":{"house_no":1001,"city":"Ahmedabad","state":"Guja
rat"},"employee_status":true})
```

In the above insert query employee_name field storing string data, employee_salary storing integer, skills field store an array, address field storing an object or nested document and employee_status storing boolean data.

Exercise 10: Different types of Comparison operators in MongoDB.

The following operators can be used in queries to compare values:

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

The $eq operator

The $eq stands for "equal." As the name implies, it is used to find the value that is equal to the specified value.

```
car_db> db.car.find({"Price":{"$eq":1200000}})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  }
]
```

The $ne operator

The $ne stands for "not equal." As the name implies, it is used to find all values that are not equal to a specified value.

```
car_db> db.car.find({"Price":{"$ne":1200000}})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  },
  {
    _id: ObjectId('664f5630a929ceaef3cdcdfb'),
    car_name: 'Alto',
    company: 'Maruti',
    Price: 420000
  }
]
car_db>
```

The $gt operator

The $gt stands for "greater than." As the name implies, it is used to find the value that is greater than the specified value.

```
car_db> db.car.find({"Price":{"$gt":1200000}})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  }
]
car_db>
```

The $gte operator

The $gte stands for "greater than equal." As the name implies, it is used to find the value that is greater or equal to the specified value.

```
car_db> db.car.find({"Price":{"$gte":1200000}})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  },
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  }
]
```

The $lt operator

The $lt stands for "less." As the name implies, it is used to find the value that is less than the specified value.

```
car_db> db.car.find({"Price":{"$lt":1200000}})
[
  {
    _id: ObjectId('664f5630a929ceaef3cdcdfb'),
    car_name: 'Alto',
    company: 'Maruti',
    Price: 420000
  }
]
car_db>
```

The $lte operator

The $lte stands for "less than equal." As the name implies, it is used to find the value that is less than or equal to the specified value.

```
car_db> db.car.find({"Price":{"$lte":1200000}})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  },
  {
    _id: ObjectId('664f5630a929ceaef3cdcdfb'),
    car_name: 'Alto',
    company: 'Maruti',
    Price: 420000
  }
]
```

Exercise 11: Logical Operators in MongoDB

MongoDB provides four different kinds of Logical Operators.

- $and
- $or
- $nor
- $not

The $and operator

It performs the logical AND operation on the expression or array of expressions. It combines two or more queries and returns the document that matches all the given conditions.

```
car_db> db.car.find({$and:[{"car_name":"Scorpio"},{"company":"Mahindra"}]})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  }
]
car_db>
```

The $or operator

It joins two or more query clauses with the logical OR operator. It returns all documents that match the conditions of either given query clause.

```
car_db> db.car.find({$or:[{"car_name":"Alto"},{"company":"Maruti"}]})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  },
  {
    _id: ObjectId('664f5630a929ceaef3cdcdfb'),
    car_name: 'Alto',
    company: 'Maruti',
    Price: 420000
  }
]
car_db>
```

The $nor operator

It is the opposite of the $or operator. It joins two or more query clauses with a logical OR operator. It returns all documents that don't match the conditions of either given query clause.

```
car_db> db.car.find({$nor:[{"car_name":"Alto"},{"company":"Maruti"}]})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  }
]
car_db>
```

The $not operator

It inverts the operation of a query expression. It returns the documents that fail to match the given query expression.

```
car_db> db.car.find({"car_name":{$not:{$eq:"Alto"}}})
[
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf8'),
    car_name: 'Scorpio',
    company: 'Mahindra',
    Price: 1600000
  },
  {
    _id: ObjectId('664f1771a929ceaef3cdcdf9'),
    car_name: 'Brezza',
    company: 'Maruti',
    Price: 1200000
  }
]
car_db>
```

Exercise 12: Aggregation Pipeline in MongoDB

Aggregation Pipeline:

When dealing with a database management system, any time you extract data from the database you need to execute an operation called a query. MongoDB provides find() method to retrieve data. The main limitation of find() method is that we cannot analyse data to find pattern or information. The solution to this problem is aggregation pipeline.

The MongoDB aggregation pipeline is a powerful framework for data aggregation operations, allowing you to process and transform documents in a collection. It consists of multiple stages, each performing a specific operation on the data. The stages are executed in sequence, with the output of one stage serving as the input for the next. Every stage performs operations. Below is a list of operations we can perform through an aggregation pipeline stage:

Filter: A filter operation is performed on a collection through a provided query.

Join: Records from different stages can be added to the primary collection records.

Group: The results of the previous stage can be grouped.

Sort: The results of the previous stage can be sorted.

Operations: Counting and merging records, or arithmetic operations.

Transforming the output: We can transform a document or decide which fields to return as a result of the aggregation.

Example: We want to count number of cars according to company wise.

```
car_db> db.car.aggregate([{$group:{_id:{company:"$company"},count:{$sum:1}}},])
[
  { _id: { company: 'Mahindra' }, count: 1 },
  { _id: { company: 'Maruti' }, count: 2 }
]
car_db>
```