MongoDb CRUD Operations

CRUD operations create, read, update, and delete operations on documents

Create Operations – to create a new document

db.collection.insertOne()

db.collection.insertMany()

Read Operations – to retrieve the document form collection

db.collection.find()

db.collection.findOne()

MongoDb CRUD Operations

Update Operations – to modify existing documents in a collection.

db.collection.updateOne()

db.collection.updateMany()

db.collection.replaceOne()

Delete Operations – to remove documents from a collection.

db.collection.deleteOne()

db.collection.deleteMany()

Inserting Documents

db.collection.save() can also be used to insert document into a collection.

If you don't specify **_id** in the document then **save()** method will work same as **insert()** method. If you specify _id then it will replace whole data of document containing _id as specified in save() method.

```
> db.stu2.save({"rno" : 7, "name" : "inderjeet singh", "age" : 15});

> db.stu2.save({"_id" :
                        ObjectId("608fb168ea90e1f913d52714"),
                        "rno" : 8,
                        "name" : "Inder kumar singh",
                        "age" : 16}
                );
```

Logical Operator in MongoDB

To query documents based on the AND condition, we need to use $and keyword.

```
>db.collection.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })
```

```
> db.sal.find({
            $and:[
                  {"DEPT" :"HR"},
                  {"DESI" : "ASSOCIATE"}
                  ]
            });
```

Note: and is the default logical operator when working with multiple conditions.

```
.sal.find({"DEPT" :"HR","DESI" : "ASSOCIATE"});
```

Logical Operator in MongoDB

To query documents based on the OR condition, we need to use $or keyword.

```
>db.collection.find({ $or: [ {<key1>:<value1>}, { <key2>:<value2>} ] })
```

```
> db.sal.find(
                    {$or:[
                              {"DEPT" :"HR"},
                              {"DESI" : "ASSOCIATE"}
                              ]
                    });
```

Logical Operator in MongoDB

To query documents based on the NOT condition, we need to use $not keyword.

```
> db.salary.find(
                    {"DEPT" :
                              {$not:
                                        {$eq : "HR"}
                              }
                    });
```

Logical Operator in MongoDB

NOR is the combination of NOT and OR, we need to use $nor keyword.

```
> db.salary.find({$nor:
                    [
                              { "DESI" : "ASSOCIATE"},
                              {"DEPT" : "HR"}
                    ]
                    });
```

Comparison Operator in MongoDb

$eq: Equality Operator same as where dept = ‘hr’ in SQL

Synatx : {<key>:{$eq;<value>}}

Eg1: > db.sal.find({"DEPT" :{$eq : "HR"}});

Eg2: > db.sal.find({"DEPT" : "HR"});

$ne: Not equal to Operator same as where dept <> ‘hr’ in SQL

Synatx : {<key>:{$ne;<value>}}

E[illegible] > db.sal.find({"DEPT" :{$ne : "HR"}});

Comparison Operator in MongoDb

$lt: less than Operator same as where salary< 100000 in SQL

Syntax: {<key>:{$lt:<value>}}

Eg: > db.sal.find({"SALARY" :{$lt : 50000}});

$lte: less than or equal to Operator same as where salary<= 50000 in SQL

Syntax: {<key>:{$lte:<value>}}

Eg: > db.sal.find({"SALARY" :{$lte : 50000}});

Comparison Operator in MongoDb

$gt: greater than or equal to Operator same as where salary>= 50000 in SQL

Syntax: {<key>:{$gt:<value>}}

Eg: > db.sal.find({"SALARY" :{$gt : 50000}});

$gte: greater than or equal to Operator same as where salary >= 50000 in SQL

Syntax: {<key>:{$gte:<value>}}

Eg: > db.sal.find({"SALARY" :{$gte : 50000}});

Comparison Operator in MongoDb

$in: represents values in an array same as where dept in (‘hr’, ‘it’, ’admin’)

Syntax: {<key>:{$in:[<value1>, <value2>,……<valueN>]}}

```
Eg: > db.sal.find({"DEPT" :
                        {$in :
                                ["HR", "IT", "TEMP"]
                        }
                });
```

$nin: represents values not in an array same as where NOT dept in (‘hr’, ‘it’, ’admin’)

Syntax: {<key>:{$nin:[<value1>, <value2>,……<valueN>]}}

```
Eg: >  db.sal.find({"DEPT" :{$nin : ["HR", "IT", "TEMP“,”ADMIN”]}});
```

Operators Examples

```
> db.orders.find({"Category" : {"$eq" : "Technology"},"Sub-Category" : {"$eq" : "Phones"}}).count()

> db.salary.find({"$and" : [{"DEPT" : {"$eq" : "HR"}}, {"DESI" : {"$eq" : "ASSOCIATE"}}]})

> db.salary.find({"$and" : [{"DEPT" :  "HR"}, {"DESI" :  "ASSOCIATE"}]})

> db.salary.find({"$and" : [{"$or" : [{"DEPT" : "HR"}, {"DEPT" : "MIS"}]},{"$or" : [{"DESI" :
"MANAGER"}, {"DESI" :  "SR. ASSOCIATE"}]}]})

> db.salary.find({"SALARY" :{"$not" : {"$gt" :  100000}}})
```

Importing Data

For importing the data we need to download MongoDB Database Tools.

https://www.mongodb.com/try/download/database-tools

Download the Zip file

Unzip the downloaded file – Browse to the bin folder – Copy mongoimport application - paste it under the bin folder of MongoDB

C:\Program Files\MongoDB\Server\4.4\bin

Go to the command prompt and run mongoimport application

Importing Data

Flags used with mongoimport command.

-d: Specifies what database to use. We used the demo database.

-c: Specifies what collection to use. We used a sal collection.

--type: Specifies the type of file to import. json, csv, or tsv. We are using csv

--headerline: Specifies that the first row in our csv file should be the field names.

--drop: Specifies that we want to drop the collection before importing documents to avoid duplicate documents.

```
>mongoimport -d demo -c sal --type csv --file Salary.csv --headerline  --drop
```

```
>mongoimport -d demo2 -c sal --type csv --file C:\Users\Raj\Desktop\Salary.csv --headerline --drop
```

Delete Documents

>db.collectionname.deleteOne({}) – delete one document even though specified criteria returns multiple documents.

>db.collectionname.deleteMany({}) – delete all documents returned by the specified criteria.

>db.collectionname.remove({},Just One) – delete a single or all documents returned by the specified criteria.

Note: db.collectionname.deleteMany({}) –this will delete all the documents.

Delete Documents

```
>db.collectionname.deleteOne()
```

```
>db.temp.insertMany([
{"id" : 1, "name" : "a" , "age" : 15},
{"id" : 2, "name" : "b" , "age" : 15},
{"id" : 3, "name" : "c" , "age" : 16},
{"id" : 4, "name" : "d" , "age" : 14},
{"id" : 5, "name" : "e" , "age" : 13},
{"id" : 6, "name" : "f" , "age" : 15},
{"id" : 7, "name" : "g" , "age" : 16},
{"id" : 8, "name" : "h" , "age" : 14},
{"id" : 9, "name" : "i" , "age" : 13},
{"id" : 10, "name" : "j" , "age" : 15}])
```

```
.temp.deleteOne({"id" : 10});
```

Delete Documents

```
>db.collectionname.deleteMany()

 > db.temp.deleteMany({"age" : 14});

 > db.temp.deleteMany({});
```

Delete Documents

>db.collectionname.remove() – delete a single or all documents returned by the specified criteria.

> db.temp.remove({"age" : 15}); - remove all documents with the matching criteria

> db.temp.remove({"age" : 15},true); - remove first document with the matching criteria

> db.temp.remove({"age" : 15},1); - remove first document with the matching criteria

Projections

Projections means selecting the required data rather than selecting the entire data of document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

The second optional parameter of the find() method is the list of fields we need to retrieve.

```
>db.collectionname.find({},{key : 1})
```

1 is used to display a field & 0 is used to hide a field

```
> db.emp.find({},{"EID" :1, "NAME" :1 ,"DOJ" : 1});
> db.emp.find({},{"DOJ" : 0, "DOB" : 0, "ADDRESS" :0});
```

Limiting Documents

limit() method is used to restrict the no of documents to be retrieved. The method accepts one number type argument, which is the number of documents that you want to be displayed

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

```
> db.salary.find({}).limit(10);
```

The **skip()** method along with the limit() method is used to skip the number of documents.

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

```
> db.salary.find({}).limit(2).skip(1);
```

ASSIGNMENT

- Import the emp & salary csv files and perform the below actions on the data:
- List the documents of Managers having salary more than 100000.
- List the documents of employees from either IT or ADMIN team.
- Remove the documents of TEMP employees

Sorting Documents

sort() method is used to sort the documents. The method accepts a document containing a list of fields along with their sorting order. 1 is used for ascending order while -1 is used for descending order.

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

```
> db.salary.find().sort({“EID" :1});
> db.salary.find({}).limit(5).sort({"SALARY" : 1});
> db.salary.find({},{"EID" : 1 , "DESI" : 1}).limit(5).sort({"EID" : 1})
> db.emp3.find({},{"EID" : 1, "Fname" :1 ,"City" :1}).sort({"City" :
1,"EID" :1}).limit(10);
```

Indexing

Indexing is done for the faster retrial of data. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.

Type of Index:

Default_id – every collection contains an index on default _id field.

Single Field – An index created on a single field in ascending or descending order.

Compound Index – An index based on multiple fields (max 31)

Multi-Key Index – index created on array field

TTL Index – TTL (Total Time to Live) are created for a limited time

Unique Index – ensures the uniqueness of the field

Indexing

Create index – db.collectionname.createIndex()

Single Field – creating a single field index in ascending order on city field in emp3 collection.

```
> db.emp3.createIndex({"City" : 1}) ;
```

Compound Index – creating a compound index in ascending order on EID & City field in emp3 collection.

```
> db.emp3.createIndex({"EID" : 1, "City" : 1});
```

Indexing

TTL Index – TTL (Total Time to Live) are created by combining “expireAfterSeconds” and createIndex().

```
> db.emp3.createIndex({"EID" : 1},{expireAfterSeconds:600});
```

TTL Limitations:

Compound Indexes can not be created as TTL

Can not be created on capped collections

Cannot be created on the field on which other index exists

Indexing

Unique Index – to create a unique index we need to set the unique option of createIndex() method to true

```
db.collectionname.createIndex({key: 1},{unique:true})
```

```
>db.emp3.createIndex({"EID" : 1},{unique:true});
```

A unique index allows 1 null value.

Indexing

Other Index methods

Find index – db.collectionname.getIndexes()

Drop index – db.collectionname.dropIndex()

Drop All index - db.collectionname.dropIndexes();

```
> db.emp.getIndexes();
> db.emp.dropIndex({"EID" : 1});
> db.emp.dropIndexes();
```

Note: A collection can have maximum 64 indexes.