Module 3

CRUD Operations

Type of Operators

Table of Contents

Modify Operators used to update

1. Comparison Query Operators:

a. $gt :

Matches values that are **greater than** the values specified in the query.

Syntax: {field :{$gt: value})

b. $gte :

Selects the documents where the value of the field is **greater than or equal** to the specified value.

Syntax: {field : {$gte : value}}

c. $in :

Select the documents where the value of a field equals any value in the specified array.

Syntax: {field: { $in : [<value1>, <value2>,…. <valueN>] })

If the field holds an array, then the $in operator selects the documents whose field holds an array that contains at least one element that matches a value in the specified array (e.g. <value1>,<value2>, etc.)

d. $lt :

Selects the documents where the value of the field is less than the specified value.

Syntax: {filed: {$lt : value}}

e. $lte :

Selects the documents where the value of the field is less than or equal to the specified value.

Syntax: {filed: {$lte : value}}

f. $ne :

Selects the documents where the value of the field is not equal (i.e. !=) to the specified value.

Syntax: {filed: {$ne : value}}

g. $nin :

Selects the documents where the field value is not in the specified array or the field does not exist. It is just opposite of $in.

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

Take an example, if you have following data and you want to perform comparison operators like $gt, $lt, $lte, $gte etc.

```
> db.coll.find().pretty()
{ "_id" : ObjectId("5479606d0f07bfe587e34782"), "name" : "sam", "age" : 10 }
{ "_id" : ObjectId("547960740f07bfe587e34783"), "name" : "mac", "age" : 11 }
{
        "_id" : ObjectId("5479607d0f07bfe587e34784"),
        "name" : "lavanya",
        "age" : 12
}
{
        "_id" : ObjectId("547960bc0f07bfe587e34785"),
        "name" : "danish",
        "age" : 13
}
{
        "_id" : ObjectId("547960dc0f07bfe587e34786"),
        "name" : "madd",
        "age" : 14
}
{
        "_id" : ObjectId("547960e70f07bfe587e34787"),
        "name" : "adam",
        "age" : 15
}
{
        "_id" : ObjectId("547960fc0f07bfe587e34788"),
        "name" : "samayra",
        "age" : 16
}
{
        "_id" : ObjectId("547961560f07bfe587e34789"),
        "name" : "henrry",
        "age" : 17
}
{
        "_id" : ObjectId("5479615f0f07bfe587e3478a"),
        "name" : "jibin",
        "age" : 18
}
{ "_id" : ObjectId("5479616d0f07bfe587e3478b"), "name" : "abc", "age" : 19 }
>
```

This shows the list whose age is greater than 15 in the “coll” collection.

```
> db.coll.find({"age" : {$gt : 15}})
{ "_id" : ObjectId("547960fc0f07bfe587e34788"), "name" : "samayra", "age" : 16 }
{ "_id" : ObjectId("547961560f07bfe587e34789"), "name" : "henrry", "age" : 17 }
{ "_id" : ObjectId("5479615f0f07bfe587e3478a"), "name" : "jibin", "age" : 18 }
{ "_id" : ObjectId("5479616d0f07bfe587e3478b"), "name" : "abc", "age" : 19 }
>
>
>
>
```

One more example shows a list of ages greater than equal to 13 and less than equal to 18 in the “coll” collection.

```
> db.coll.find({"age" : {"$gte":13, "$lte" :18}})
{ "_id" : ObjectId("547960bc0f07bfe587e34785"), "name" : "danish", "age" : 13 }
{ "_id" : ObjectId("547960dc0f07bfe587e34786"), "name" : "madd", "age" : 14 }
{ "_id" : ObjectId("547960e70f07bfe587e34787"), "name" : "adam", "age" : 15 }
{ "_id" : ObjectId("547960fc0f07bfe587e34788"), "name" : "samayra", "age" : 16 }

{ "_id" : ObjectId("547961560f07bfe587e34789"), "name" : "henrry", "age" : 17 }
{ "_id" : ObjectId("5479615f0f07bfe587e3478a"), "name" : "jibin", "age" : 18 }
>
```

2. Logical Query Operators:

a. $and :

Syntax: { $and :[{ <expression1> }, { <expression2> }, . .,{ <expression> }] }

$and performs a logical AND operation on an array of *two or more* expressions (e.g.<expression1>, <expression2>, etc.) and selects the documents that satisfy *all* the expressions in the array.

If you want to select all documents in the coll collection where the age field value is not equal to 15 and the age field exists.
Same operation can also be done using the command like:
db.coll.find({ age : {$ne : 15, $exists: true } })

```
> db.coll.find( { $and: [{"age" : {$ne : 15 }}, {"age" : {$exists :true}}]})
{ "_id" : ObjectId("5479606d0f07bfe587e34782"), "name" : "sam", "age" : 10 }
{ "_id" : ObjectId("547960740f07bfe587e34783"), "name" : "mac", "age" : 11 }
{ "_id" : ObjectId("5479607d0f07bfe587e34784"), "name" : "lavanya", "age" : 12 }

{ "_id" : ObjectId("547960bc0f07bfe587e34785"), "name" : "danish", "age" : 13 }
{ "_id" : ObjectId("547960dc0f07bfe587e34786"), "name" : "madd", "age" : 14 }
{ "_id" : ObjectId("547960fc0f07bfe587e34788"), "name" : "samayra", "age" : 16 }

{ "_id" : ObjectId("547961560f07bfe587e34789"), "name" : "henrry", "age" : 17 }
{ "_id" : ObjectId("5479615f0f07bfe587e3478a"), "name" : "jibin", "age" : 18 }
{ "_id" : ObjectId("5479616d0f07bfe587e3478b"), "name" : "abc", "age" : 19 }
>
>
```

b. $nor :

Syntax: { $nor :[{ <expression1>}, {<expression2>}, . . {<expression}] }

$nor performs a logical NOR operation on an array of one or more query expression and selects the documents that **fail** all the query expressions in the array.

c. $not :

Syntax: { field: { $not : {<operator-expression> } } }

$not performs a logical NOT operation on the specified <operator-expression> and selects the documents that do *not* match the <operator-expression>. This includes documents that do not contain the field.

d. $or :

Syntax: { $or : [{ <expression1> }, {<expression2>}, . ., {<expression>}]}

The $or operator performs a logical OR operation on an array of *two or more* <expressions> and selects the documents that satisfy *at least* one of the <expressions>.

If you want to get a list with age less than 16 or name as “madd”, then perform the following query i.e.

```
> db.coll.find({$or: [{"age": {$lt : 16}}, {"name" : "madd"}]}).pretty()
{ "_id" : ObjectId("5479606d0f07bfe587e34782"), "name" : "sam", "age" : 10 }
{ "_id" : ObjectId("547960740f07bfe587e34783"), "name" : "mac", "age" : 11 }
{
        "_id" : ObjectId("5479607d0f07bfe587e34784"),
        "name" : "lavanya",
        "age" : 12
}
{
        "_id" : ObjectId("547960bc0f07bfe587e34785"),
        "name" : "danish",
        "age" : 13
}
{
        "_id" : ObjectId("547960dc0f07bfe587e34786"),
        "name" : "madd",
        "age" : 14
}
{
        "_id" : ObjectId("547960e70f07bfe587e34787"),
        "name" : "adam",
        "age" : 15
}
```

3. Element Query Operators:

a. $exists: Matches documents that have the specified field.

Syntax: { field: { $exists: <boolean> } }

When <boolean> is true, **$exists** matches the documents that contain the field, including documents where the field value is null. If <boolean> is false, the query returns only the documents that do not contain the field.

b. $type: Selects documents if a field is of the specified type.

Syntax: { field : { $type : <BSON type> } }

It selects the documents where the value of the field is the specified numeric BSON type. BSON type can be double, string, object, array, undefined, Boolean, data, null, min key, max key etc.

```
> db.test.find()
{ "_id" : ObjectId("54b39b7cc043981617cd33dd"), "x" : 3 }
{ "_id" : ObjectId("54b39b7cc043981617cd33de"), "x" : 2.9 }
{ "_id" : ObjectId("54b39b7cc043981617cd33df"), "x" : ISODate("2015-01-12T10:01:
32.579Z") }
{ "_id" : ObjectId("54b39b7cc043981617cd33e0"), "x" : true }
{ "_id" : ObjectId("54b39b7cc043981617cd33e1"), "x" : { "$maxKey" : 1 } }
{ "_id" : ObjectId("54b39b7cc043981617cd33e2"), "x" : { "$minKey" : 1 } }
{ "_id" : ObjectId("54b3bac7c043981617cd33e4"), "x" : "abc" }
>
>
> db.test.find({"x" : {$type : 9 } } );
{ "_id" : ObjectId("54b39b7cc043981617cd33df"), "x" : ISODate("2015-01-12T10:01:
32.579Z") }
> db.test.find({"x" : {$type : 9 } } );
{ "_id" : ObjectId("54b39b7cc043981617cd33df"), "x" : ISODate("2015-01-12T10:01:
32.579Z") }
>
>
> db.test.find({"x" : {$type : 2 } } );
{ "_id" : ObjectId("54b3bac7c043981617cd33e4"), "x" : "abc" }
>
> db.test.find().sort({"x" : -1})
{ "_id" : ObjectId("54b39b7cc043981617cd33e1"), "x" : { "$maxKey" : 1 } }
{ "_id" : ObjectId("54b39b7cc043981617cd33df"), "x" : ISODate("2015-01-12T10:01:
32.579Z") }
{ "_id" : ObjectId("54b39b7cc043981617cd33e0"), "x" : true }
{ "_id" : ObjectId("54b3bac7c043981617cd33e4"), "x" : "abc" }
{ "_id" : ObjectId("54b39b7cc043981617cd33dd"), "x" : 3 }
{ "_id" : ObjectId("54b39b7cc043981617cd33de"), "x" : 2.9 }
{ "_id" : ObjectId("54b39b7cc043981617cd33e2"), "x" : { "$minKey" : 1 } }
>
>
> db.test.find().sort({"x" : 1})
{ "_id" : ObjectId("54b39b7cc043981617cd33e2"), "x" : { "$minKey" : 1 } }
{ "_id" : ObjectId("54b39b7cc043981617cd33de"), "x" : 2.9 }
{ "_id" : ObjectId("54b39b7cc043981617cd33dd"), "x" : 3 }
{ "_id" : ObjectId("54b3bac7c043981617cd33e4"), "x" : "abc" }
{ "_id" : ObjectId("54b39b7cc043981617cd33e0"), "x" : true }
{ "_id" : ObjectId("54b39b7cc043981617cd33df"), "x" : ISODate("2015-01-12T10:01:
32.579Z") }
{ "_id" : ObjectId("54b39b7cc043981617cd33e1"), "x" : { "$maxKey" : 1 } }
>
```

4. Evaluation Query Operators:

a. $mod:

Syntax: {field: { $mod : [divisor, remainder] } }

Select documents where the value of a field divided by a divisor has the specified remainder.

List of documents whose age are divisible by 4 and remainder is 0.

```
> db.coll.find({"age" : {$mod : [4,0]}})
{ "_id" : ObjectId("5479607d0f07bfe587e34784"), "name" : "lavanya", "age" : 12 }
{ "_id" : ObjectId("547960fc0f07bfe587e34788"), "name" : "samayra", "age" : 16 }
>
```

b. $regex:

Syntax :{ <field> : { $regex: /pattern/ , $options: '<options>' } }

Selects documents where values match a specified regular expression.

Matches documents with name field value that start with "sam",

```
> db.coll.find( { "name" : {$regex : /^sam/}}).pretty()
{ "_id" : ObjectId("5479606d0f07bfe587e34782"), "name" : "sam", "age" : 10 }
{
        "_id" : ObjectId("547960fc0f07bfe587e34788"),
        "name" : "samayra",
        "age" : 16
}
>
```

c. $where:

Matches documents that satisfy a JavaScript expression**.** Use the $where operator to pass either a string containing a JavaScript expression or a full JavaScript function to the query system. The $where provides greater flexibility, but requires that the database processes the JavaScript expression or function for *each* document in the collection.

5. Query Operator Array:

a. $all:

The $all operator selects the documents where the value of a field is an array that contains all the specified elements.

b. $elemMatch:

The $elemMatch operator matches documents in a collection that contain an array field with at least one element that matches all the specified query criteria.

c. $size:

The $size operator matches any array with the number of elements specified by the argument.

6. Projection Operators:

a. $: Projects the first element in an array that matches the query condition.

b. $slice: Limits the number of elements projected from an array. Supports skip and limit slices.

7. Update Operators: The following modifiers are available for use in update operations; **e.g. in db.collection.update()**

a. $currentDate : Sets the value of a field to current date, either as a Date or a Timestamp.
b. $inc : Increments the value of the field by the specified amount.
c. $max: Only updates the field if the specified value is greater than the existing field value.
d. $min: Only updates the field if the specified value is less than the existing field value
e. $mul: Multiplies the value of the field by the specified amount.
f. $rename: Renames a field.

Here the name field is renamed with a new field "Name" in the document where name is abc.

```
> db.coll.update( { "name" : "abc" }, { $rename: { "name" : "Name"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
>
```

g. setOnInsert: Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
h. $set: Sets the value of a field in a document.
i. $unset: Removes the specified field from a document.

8. Array Update Operators:

a. $: Acts as a placeholder to update the first element that matches the query condition in an update.
b. $addToSet: Adds elements to an array only if they do not already exist in the set.
c. $pop: Removes the first or last item of an array.

Suppose we have following document:

```
{
        "_id" : ObjectId("547982f10f07bfe587e3478d"),
        "name" : "john",
        "scores" : [
                8,
                9,
                10,
                5,
                7,
                2,
                9
        ]
}
{
        "_id" : ObjectId("547983020f07bfe587e3478e"),
        "name" : "maaan",
        "scores" : [
                8,
                9,
                10,
                2,
                9
        ]
}
```

And we want to pop an element from one document means to remove an element from an array, use:

```
> db.mycoll.update({"name" : "maaan"}, {$pop : {"scores": -1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

And when you will see again the documents the first element from the array i.e. 8 will be removed from that array. If you want to remove the last element from the array use {“score” : 1} instead of -1.

```
{
        "_id" : ObjectId("547982f10f07bfe587e3478d"),
        "name" : "john",
        "scores" : [
                8,
                9,
                10,
                5,
                7,
                2,
                9
        ]
}
{
        "_id" : ObjectId("547983020f07bfe587e3478e"),
        "name" : "maaan",
        "scores" : [
                9,
                10,
                2,
                9
        ]
}
```

d. $pullAll: Removes all matching values from an array. This command will remove all instances of the value 9 and 2 from the scores array.

```
> db.mycoll.update({"name" : "maaan"},{$pullAll : {"scores": [9,2]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

And now you can see that value 9 and 2 does not exist in the scores array.

```
{
        "_id" : ObjectId("547982f10f07bfe587e3478d"),
        "name" : "john",
        "scores" : [
                8,
                9,
                10,
                5,
                7,
                2,
                9
        ]
}
{
        "_id" : ObjectId("547983020f07bfe587e3478e"),
        "name" : "maaan",
        "scores" : [
                10
        ]
}
```

e. $pull: Removes all array elements that match a specified query.

```
> db.mycoll.update( { "scores" : 10}, {$pull : {"scores":10}},{multi:true})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
>
```

```
{
        "_id" : ObjectId("547982f10f07bfe587e3478d"),
        "name" : "john",
        "scores" : [
                8,
                9,
                5,
                7,
                2,
                9
        ]
}
{
        "_id" : ObjectId("547983020f07bfe587e3478e"),
        "name" : "maaan",
        "scores" : [ ]
}
```

f. $pushAll: Adds several items to an array.

g. $push: Adds an item to an array.

```
> db.mycoll.update( { "name" : "maaan"}, {$push : {"scores":[10,1,5]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

h. $each: Modifies the $push and $addToSet operators to append multiple items for array updates.

```
> db.mycoll.update( { "name" : "john"}, {$push: {"scores" : {$each : [93,82,35]
} } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

$push operator with $each operator. Here the following example appends each element of [93,82, 35] to the scores array for the document where the name field equals john.

```
{
        "_id" : ObjectId("547982f10f07bfe587e3478d"),
        "name" : "john",
        "scores" : [
                8,
                9,
                5,
                7,
                2,
                9,
                93,
                82,
                35
        ]
}
```

i. $position: Modifies the $push operator to specify the position in the array to add elements.

Here the operation updates the scores field to add the elements 20 and 30 at the array index of 2.

```
> db.mycoll.update( { "name" : "john"}, { $push: {"scores" : {$each :[20,30], $p
osition:2}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

You can see that elements 20 and 30 at the array index of 2 that is after 9.

```
{
        "_id" : ObjectId("547982f10f07bfe587e3478d"),
        "name" : "john",
        "scores" : [
                8,
                9,
                20,
                30,
                5,
                7,
                2,
                9,
                93,
                82,
                35
        ]
}
```

j. $slice: Modifies the $push operator to limit the size of updated arrays.

```
> db.mycoll.update({"name" : "john"},{$push :{ "scores" : {$each: [100,20], $sli
ce : 3}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

k. $sort: Modifies the $push operator to reorder documents stored in an array.

Here elements 40 and 60 will be pushed to the score field and then the elements will be sorted in the ascending order.

```
>
> db.mycoll.update(
... {"name" : "john"},
... {$push : { "scores" : {$each: [40,60], $sort : 1}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
>
>
```

Aggregation Commands

Name	Description
Aggregate	Performs aggregation tasks such as group using the aggregation framework.
Count	Counts the number of documents in a collection.
Distinct	Displays the distinct values found for a specified key in a collection.
Group	Groups documents in a collection by the specified key and performs simple aggregation.

mapReduce	Performs map-reduce aggregation for large data sets.
skip	Skips over the specified number of *documents* that pass into the stage and passes the remaining documents to the next stage in the *pipeline*.

Count: this command counts the number of documents in a collection. The command returns a document that contains the count as well as the command status.

Syntax: {count : <collection>, query : <query>, limit : <limit>, skip : <skip> }

Here the fields like **count** takes the name of collection to count, **query** specifies the selection query to determine which document in the collection to count, **limit** specifies the limit for documents matching the selection query and **skip** specifies the number of matching documents to skip. Query, limit and skip field are optional here.

Count the number of all documents in the mycoll collection. Here, "n" represents the count that is 7 documents in the mycoll collection.

```
> db.runCommand( { count : 'mycoll'})
{ "n" : 7, "ok" : 1 }
>
>
>
```

Count the number of the documents in the coll collection with the field "age" greater than 15 skipping the first 3 matching records.

```
>
> db.runCommand( { count : 'coll', query: { "age" : {$gt : 15}}, skip : 3})
{ "n" : 1, "ok" : 1 }
>
>
>
```

Skip: here the starting 4 documents will be skipped and then rest of the documents will be displayed.

```
> db.coll.aggregate({$skip : 4})
{ "_id" : ObjectId("547960dc0f07bfe587e34786"), "name" : "madd", "age" : 14 }
{ "_id" : ObjectId("547960e70f07bfe587e34787"), "name" : "adam", "age" : 15 }
{ "_id" : ObjectId("547960fc0f07bfe587e34788"), "name" : "samayra", "age" : 16 }

{ "_id" : ObjectId("547961560f07bfe587e34789"), "name" : "henrry", "age" : 17 }
{ "_id" : ObjectId("5479615f0f07bfe587e3478a"), "name" : "jibin", "age" : 18 }
{ "_id" : ObjectId("5479616d0f07bfe587e3478b"), "age" : 19, "Name" : "abc" }
>
>
```

Distinct: The distinct command finds the distinct values for a specified field across a single collection. The command returns a document that contains an array of the distinct values as well as the query plan and status.

Here the command return an array of distinct values of the field name from the documents in the "coll" collection where the age is greater than 13.

```
> db.runCommand ( {distinct : 'coll', key: "name", query : {"age" : {$gt : 13}}})
>
{
        "values" : [
                "madd",
                "adam",
                "samayra",
                "henrry",
                "jibin"
        ],
        "stats" : {
                "n" : 6,
                "nscanned" : 10,
                "nscannedObjects" : 10,
                "timems" : 0,
                "cursor" : "BasicCursor"
        },
        "ok" : 1
}
>
```

Queries Comparison

The following table represents the various SQL statements related to table-level actions and the

Corresponding MongoDB statements.

Create, Insert, Select, Update and Delete:

SQL statements	**MongoDB statements**
CREATE TABLE users(name VARCHAR (128), age NUMBER)	db.createCollection("users")
INSERT INTO users values ('Bob', 12)	db.users.insert({"name" : "Bob", "age"

	: 12})
SELECT * FROM users	db.users.find()
SELECT name, age FROM users	db.users.find({}, {“name” :1, “age” : 1, _id :0})
SELECT name, age FROM users WHERE age =33	db.users.find({“age” : 33}, {“name” : 1, “age” : 1, “_id” : 0})
SELECT*FROM users WHERE age>33	db.users.find({“age” : {$gt : 33}})
SELECT*FROM users WHERE age<=33	db.users.find({age : {$lte : 33}})
SELECT*FROM users WHERE age = 33 OR name=Bob	db.users.find({$or : [{“age” : 33}, {“name” : “Bob”}]})
SELECT*FROM users WHERE name like ‘Joe%’	db.users.find({“name”:/Joe/})
SELECT*FROM users LIMIT 10 SKIP 20	db.users.find().skip(20).limit(10)
SELECT COUNT (AGE) from users	db.users.find({“age” : {$exists : true}}).count()
Update users SET age = 33 WHERE name=’Bob’ **(here multi documents will be affected by this query)**	db.users.update({“name” :”Bob”}, {$set : {age :33}}, {multi : true}) **(manipulating multiple documents at the same time)**
Create Index on users(name ASC)	db.users.ensureIndex({“name” :1})
Create Index on users(name ASC, age DESC)	db.users.ensureIndex({name : 1, age : -1})

Explain SELECT*FROM users WHERE age=32	db.users.find({“age” : 32}).explain()
UPDATE users SET age = age+3 WHERE status = “A”	db.users.update({status : “A”}, { $inc: { age : 3}}, {multi : true })
DELETE FROM users WHERE status = “D”	db.users.remove({status : “D”})
DELETE FROM users	db.users.remove({})