U--->Update Operation:

students collection

Based on our requirement, we can perform update operations to reflect latest information.

eg-1: update student document with changed mobile number
eg-2: Increment all employee salaries by 1000 if salary is less than 10000

We can perform updations like

1. Overwrite existing value of a particular field with our new value
2. Add a new field for selected documents
3. Remove an existing field
4. Rename an existingfield
etc

How to perform updations:

We can perform required updations by using update methods and update operators.

update methods:

There are 3 update methods are avaialble.
1. updateOne()
2. updateMany()

3. update()

1. updateOne():

---

db.collection.updateOne(filter,update,options)

It finds the first document that matches filter criteria and perform required updation. It will perform updation for a single document.

2. updateMany():

---

db.collection.updateMany(filter,update,options)
To update all documents that match the specified filter criteria.

3. update():

---

db.collection.update(filter,update,options)

We can use this method to update either a single document or multiple documents.
Bydefault this method updates a single document only.

If we include multi:true to update all documents that match query criteria.

db.collection.update(filter,update)--->To update a single document
db.collection.update(filter,update,{multi:true})--->To update all matched documents

Case Study:

---

db.employees.insert({_id:1,eno:100,ename:"Sunny",esal:1000,eaddr:"Mumbai"})

```
db.employees.insert({_id:2,eno:200,ename:"Bunny",esal:2000,eaddr:"Hyderabad"})
db.employees.insert({_id:3,eno:300,ename:"Chinny",esal:3000,eaddr:"Mumbai"})
db.employees.insert({_id:4,eno:400,ename:"Vinny",esal:4000,eaddr:"Delhi"})
db.employees.insert({_id:5,eno:500,ename:"Pinny",esal:5000,eaddr:"Chennai"})
db.employees.insert({_id:6,eno:600,ename:"Tinny",esal:6000,eaddr:"Mumbai"})
db.employees.insert({_id:7,eno:700,ename:"Zinny",esal:7000,eaddr:"Delhi"})
```

Note: To perform updations we have to use update operators like $set, $unset, $inc etc

$set --->To set new value to the specified field

Q1. Update salary of Sunny with 9999?

```
> db.employees.updateOne({ename: "Sunny"},{$set: {esal: 9999}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

```
> db.employees.find({ename: "Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1000, "eaddr" : "Mumbai" }
> db.employees.updateOne({ename: "Sunny"},{$set: {esal: 9999}})
```

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.employees.find({ename: "Sunny"})

{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 9999, "eaddr" : "Mumbai" }

Q2. Update all Mumbai based employees salary as 7777?

```
> db.employees.updateOne({eaddr: "Mumbai"},{$set: {esal: 7777}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

It will perform updation only for first matched document.

```
> db.employees.updateOne({eaddr: "Mumbai"},{$set: {esal: 7777}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.employees.find({eaddr: "Mumbai"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3000, "eaddr" :
"Mumbai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6000, "eaddr" :
"Mumbai" }
```

Note: updateOne() will always consider only first matched document. If updation is not available then only it will perform updation.

```
> db.employees.updateOne({eaddr: "Mumbai"},{$set: {esal: 7777}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.employees.find({eaddr: "Mumbai"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3000, "eaddr" :
"Mumbai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6000, "eaddr" :
"Mumbai" }
```

```
> db.employees.updateMany({eaddr: "Mumbai"},{$set: {esal: 7777}})
```

It will update all matched documents

```
> db.employees.find({eaddr: "Mumbai"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3000, "eaddr" :
"Mumbai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6000, "eaddr" :
"Mumbai" }
> db.employees.updateMany({eaddr: "Mumbai"},{$set: {esal: 7777}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 2 }
> db.employees.find({eaddr: "Mumbai"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 7777, "eaddr" :
"Mumbai" }
```

Q3. Update all Delhi based Employees salary as 5555?

```
> db.employees.update({eaddr: "Delhi"},{$set: {esal:5555}})
```

It will perform updation for only first matched document.It is exactly same as updateOne() method.

```
> db.employees.find({eaddr: "Delhi"})
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4000, "eaddr" :
"Delhi" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7000, "eaddr" :
"Delhi" }
> db.employees.update({eaddr: "Delhi"},{$set: {esal:5555}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.employees.find({eaddr: "Delhi"})
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 5555, "eaddr" :
"Delhi" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7000, "eaddr" :
"Delhi" }


>db.employees.update({eaddr: "Delhi"},{$set: {esal:5555}},{multi: true})
  It will perform updation for all matched documents.

> db.employees.find({eaddr: "Delhi"})
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 5555, "eaddr" :
"Delhi" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7000, "eaddr" :
"Delhi" }
> db.employees.update({eaddr: "Delhi"},{$set: {esal:5555}},{multi: true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 1 })
> db.employees.update({eaddr: "Delhi"},{$set: {esal:4444}},{multi: true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
```

Update Operators:

We can use update operators to perform required updations.

1. $set
2. $unset
3. $rename
4. $inc
5. $min
6. $max
7. $mul

etc

1. $set operator:

We can use $set operator to set the value to the field in matched document.

```
> db.employees.update({ename:"Sunny"},{$set: {esal:9999}})
```

case-1:

If the specified field does not exist, $set will add a new field with provided value.

```
> db.employees.update({ename:"Sunny"},{$set: {husband: "Daniel"}})


> db.employees.find({ename:"Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" : "Mumbai" }
> db.employees.update({ename:"Sunny"},{$set: {husband: "Daniel"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.employees.find({ename:"Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" : "Mumbai", "husband" : "Daniel" }
```

Case-2: updating multiple fields at a time

We can perform updations for multiple fields at a time.

```
{$set: {field1: value1, field2:value2, ...}}

> db.employees.update({ename:"Sunny"},{$set: {esal: 1111, age: 45, origin: "Punjab"}})
```

```
> db.employees.find({ename:"Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 7777, "eaddr" :
"Mumbai", "husband" : "Daniel" }
> db.employees.update({ename:"Sunny"},{$set: {esal: 1111, age: 45,
origin: "Punjab"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.employees.find({ename:"Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1111, "eaddr" :
"Mumbai", "husband" : "Daniel", "age" : 45, "origin" : "Punjab" }
```

Q3. Add a new field named with friend with value Guest where esal value is >= 4000?

```
> db.employees.updateMany({esal:{$gte: 4000}},{$set: {friend:"Guest"}})
> db.employees.update({esal:{$gte: 4000}},{$set:
{friend:"Guest"}},{multi:true})


> db.employees.find({esal:{$gte: 4000}})
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4444, "eaddr" :
"Delhi" }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 7777, "eaddr" :
"Mumbai" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 4444, "eaddr" :
"Delhi" }
> db.employees.updateMany({esal:{$gte: 4000}},{$set: {friend:"Guest"}})
{ "acknowledged" : true, "matchedCount" : 5, "modifiedCount" : 5 }
```

```
> db.employees.find({esal:{$gte: 4000}})
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 7777, "eaddr" :
"Mumbai", "friend" : "Guest" }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4444, "eaddr" :
"Delhi", "friend" : "Guest" }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai", "friend" : "Guest" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 7777, "eaddr" :
"Mumbai", "friend" : "Guest" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 4444, "eaddr" :
"Delhi", "friend" : "Guest" }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1111, "eaddr" :
"Mumbai", "husband" : "Daniel", "age" : 45, "origin" : "Punjab" }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2000, "eaddr" :
"Hyderabad" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 7777, "eaddr" :
"Mumbai", "friend" : "Guest" }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4444, "eaddr" :
"Delhi", "friend" : "Guest" }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai", "friend" : "Guest" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 7777, "eaddr" :
"Mumbai", "friend" : "Guest" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 4444, "eaddr" :
"Delhi", "friend" : "Guest" }
```

case-3: Nested document updation:

```
{
        "_id" : ObjectId("5fe95428fe935cdac43627cf"),
        "title" : "Oracle in simple way",
        "isbn" : 6677,
```

```
        "downloadable" : true,
        "no_of_reviews" : 3,
        "tags" : [
                "database",
                "sql",
                "relational"
        ],
        "languages" : [
                "english",
                "hindi",
                "telugu"
        ],
        "author" : {
                "name" : "Virat Kohli",
                "callname" : "kohli",
                "profile" : {
                        "exp" : 2,
                        "courses" : 2,
                        "books" : 2
                }
        }
}
```

Q. Change call name of Virat Kohli as Virushka?

```
> db.books.update({"author.name": "Virat Kohli"},{$set:
{"author.callname": "Virushka"}})

> db.books.find({"author.name": "Virat Kohli"}).pretty()
{
        "_id" : ObjectId("5fe95428fe935cdac43627cf"),
        "title" : "Oracle in simple way",
        "isbn" : 6677,
```

```
        "downloadable" : true,
        "no_of_reviews" : 3,
        "tags" : [
                "database",
                "sql",
                "relational"
        ],
        "languages" : [
                "english",
                "hindi",
                "telugu"
        ],
        "author" : {
                "name" : "Virat Kohli",
                "callname" : "kohli",
                "profile" : {
                        "exp" : 2,
                        "courses" : 2,
                        "books" : 2
                }
        }
}
> db.books.update({"author.name": "Virat Kohli"},{$set:
{"author.callname": "Virushka"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.books.find({"author.name": "Virat Kohli"}).pretty()
{
        "_id" : ObjectId("5fe95428fe935cdac43627cf"),
        "title" : "Oracle in simple way",
        "isbn" : 6677,
        "downloadable" : true,
        "no_of_reviews" : 3,
        "tags" : [
```

```
            "database",
            "sql",
            "relational"
        ],
        "languages" : [
            "english",
            "hindi",
            "telugu"
        ],
        "author" : {
            "name" : "Virat Kohli",
            "callname" : "Virushka",
            "profile" : {
                "exp" : 2,
                "courses" : 2,
                "books" : 2
            }
        }
}
```

2. $unset operator:

To delete the specified field.

Syntax:

```
{$unset: {field1:"",field2:"",...}}
```

The specified value in the $unset expression (ie "") does not impact operation.

Q1. Delete esal and husband fields where ename is "Sunny"?

```
> db.employees.update({ename: "Sunny"},{$unset: {esal:0,husband:""}})
```

```
> db.employees.find({ename: "Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1000, "eaddr" :
"Mumbai", "husband" : "Daniel" }
> db.employees.update({ename: "Sunny"},{$unset: {esal:0,husband:""}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.employees.find({ename: "Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "eaddr" : "Mumbai" }
```

Note: If the specified field is not avaialble, then $unset operator won't do anything.

```
> db.employees.update({ename: "Sunny"},{$unset: {age:0}})

> db.employees.find({ename: "Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "eaddr" : "Mumbai" }
> db.employees.update({ename: "Sunny"},{$unset: {age:0}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.employees.find({ename: "Sunny"})
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "eaddr" : "Mumbai" }
```

Q. Remove fields husband and friend where esal is less than 8000?

```
> db.employees.updateMany({esal:{$lt:8000}}, {$unset:
{husband:"",friend:""}})
```

3. $rename operator:

We can use $rename operator to rename fields, ie to change name of the field.

Syntax:

```
{$rename: {field1:<newName1>, field2:<newName2>, ...} }
```

Q. Write Query to rename esal as salary and eaddr as city in employees collection?

```
> db.employees.updateMany({},{$rename: {esal:"salary", eaddr: "city"}})


> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1000, "eaddr" :
"Mumbai" }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2000, "eaddr" :
"Hyderabad" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3000, "eaddr" :
"Mumbai" }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4000, "eaddr" :
"Delhi" }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6000, "eaddr" :
"Mumbai" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7000, "eaddr" :
"Delhi" }
> db.employees.updateMany({},{$rename: {esal:"salary", eaddr: "city"}})
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 7 }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "city" : "Mumbai", "salary" :
1000 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "city" : "Hyderabad", "salary"
: 2000 }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "city" : "Mumbai", "salary" :
3000 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "city" : "Delhi", "salary" :
4000 }
```

{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "city" : "Chennai", "salary" : 5000 }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "city" : "Mumbai", "salary" : 6000 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "city" : "Delhi", "salary" : 7000 }

Note:

---

1. The $rename operator internally performs $unset of both old name and new name and then performs $set with new name. Hence it won't preserve order of fields.

2. If the document already has a field with newName then $rename operator removes that field and renames specified field with newName.

eg:

```
>
db.employees.insert({_id:8,eno:800,esal:8000,eaddr:"Hyderabad",city:"Mumbai"})
> db.employees.find()
```

Q. rename eaddr as city?

```
> db.employees.update({_id:8},{$rename: {eaddr:"city"}})


> db.employees.find({_id:8})
{ "_id" : 8, "eno" : 800, "esal" : 8000, "eaddr" : "Hyderabad", "city" : "Mumbai" }
> db.employees.update({_id:8},{$rename: {eaddr:"city"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.employees.find({_id:8})
```

{ "_id" : 8, "eno" : 800, "esal" : 8000, "city" : "Hyderabad" }

3. If the field to rename does not exist in the document then $rename won't do anything.

```
> db.employees.update({_id:8},{$rename: {age:"totalage"}})


> db.employees.find({_id:8})
{ "_id" : 8, "eno" : 800, "esal" : 8000, "city" : "Hyderabad" }
> db.employees.update({_id:8},{$rename: {age:"totalage"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.employees.find({_id:8})
{ "_id" : 8, "eno" : 800, "esal" : 8000, "city" : "Hyderabad" }
```

4. $inc operator:

---

inc means increment.

We can use $inc to increment or decrement value of the field with specified amount.

salary=salary+1000

salary=salary-1000

Syntax:

{$inc: {field1:amount1,field2:amount,..}}

$inc can take both positive and negative values.

positive value for increment operation

negative value for decrement operation

case study:

```
db.employees.insert({_id:1,eno:100,ename:"Sunny",esal:1000,eaddr:"Mumbai"})
db.employees.insert({_id:2,eno:200,ename:"Bunny",esal:2000,eaddr:"Hyderabad"})
db.employees.insert({_id:3,eno:300,ename:"Chinny",esal:3000,eaddr:"Mumbai"})
db.employees.insert({_id:4,eno:400,ename:"Vinny",esal:4000,eaddr:"Delhi"})
db.employees.insert({_id:5,eno:500,ename:"Pinny",esal:5000,eaddr:"Chennai"})
db.employees.insert({_id:6,eno:600,ename:"Tinny",esal:6000,eaddr:"Mumbai"})
db.employees.insert({_id:7,eno:700,ename:"Zinny",esal:7000,eaddr:"Delhi"})
```

Q1. Increment all employee salary by 500.

```
> db.employees.updateMany({},{$inc: {esal:500}})
```

```
> db.employees.updateMany({},{$inc: {esal:500}})
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 7 }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1500, "eaddr" : "Mumbai" }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2500, "eaddr" : "Hyderabad" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3500, "eaddr" : "Mumbai" }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" : "Delhi" }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5500, "eaddr" : "Chennai" }
```

```
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6500, "eaddr" :
"Mumbai" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7500, "eaddr" :
"Delhi" }
```

Q2. Decrement Employee salary by Rs 1 where esal is > 4700?

```
> db.employees.updateMany({esal:{$gt: 4700}},{$inc: {esal:-1}})

> db.employees.updateMany({esal:{$gt: 4700}},{$inc: {esal:-1}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1500, "eaddr" :
"Mumbai" }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2500, "eaddr" :
"Hyderabad" }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3500, "eaddr" :
"Mumbai" }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" :
"Delhi" }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5499, "eaddr" :
"Chennai" }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6499, "eaddr" :
"Mumbai" }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7499, "eaddr" :
"Delhi" }
```

Note:

1. If the specified field does not exist, $inc creates that field and sets that field to the specified value.

```
> db.employees.updateMany({},{$inc: {age:2}})
```

```
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 7 }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1500, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2500, "eaddr" :
"Hyderabad", "age" : 2 }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3500, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" :
"Delhi", "age" : 2 }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5499, "eaddr" :
"Chennai", "age" : 2 }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6499, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7499, "eaddr" :
"Delhi", "age" : 2 }
```

2. We cannot perform multiple updates on the same field at a time, otherwise we will get error.

```
> db.employees.updateMany({},{$inc:{esal:500}, $set:{esal:5000}})
 "errmsg" : "Updating the path 'esal' would create a conflict at 'esal'"
```

$set--->To set a new value to the specified field/To create a new field with provided value
$unset--->To delete specified field
$rename--->To rename the specified field
$inc--->To perform increment and decrement operations on the field value

Basic understanding purpose:

```
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1500, "eaddr" : "Mumbai", "age" : 2 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2500, "eaddr" : "Hyderabad", "age" : 2 }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3500, "eaddr" : "Mumbai", "age" : 2 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" : "Delhi", "age" : 2 }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5499, "eaddr" : "Chennai", "age" : 2 }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 6499, "eaddr" : "Mumbai", "age" : 2 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 7499, "eaddr" : "Delhi", "age" : 2 }
```

Q1. update 5000 as minimum salary for every employee--->$max operator

Q2. update 5000 as maximum salary for every employee--->$min operator

Q3. Increment Every employee salary by 10% --->$mul operator esal*1.1

Q4. Double Every employee salary as Covid Offer --->$mul operator esal*2

4. $min operator:

It only updates field value if the specified value is less than current field value .

minimum value of(provided value,current value)

Consider only the value which is minimum among given and current

Syntax: {$min: {field1:value1, field2:value2,...}}

Q. To make maximum salary of every employee as 5000. If any employee salary greater than 5000 then assign 5000?

```
> db.employees.updateMany({},{$min: {esal:5000}})
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 3 }


> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 1500, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 2500, "eaddr" :
"Hyderabad", "age" : 2 }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 3500, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" :
"Delhi", "age" : 2 }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai", "age" : 2 }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 5000, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 5000, "eaddr" :
"Delhi", "age" : 2 }
```

Note: If the specified field does not exist, then $min operator creates that field and assign with provided value.

```
> db.employees.updateMany({},{$min:{marks: 99}})
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 7 }
```

6. $max operator:

$max operator updates the value of the field to the specified value iff specified value is greater than current value.

i.e maximum of (provided value,current value)

Syntax: {$max: {field1:value1, field2:value2,...}}

Q. Make minimum salary of every employee as 4000. i.e if any employee salary is less than 4000 then assign 4000?

max of (4000, current value)
4000,4500-->4500
4000,2300--->4000

> db.employees.updateMany({},{$max: {esal:4000}})
if 4000 is greater than current value then only updation will be happend.

```
> db.employees.updateMany({},{$max: {esal:4000}})
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 3 }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 4000, "eaddr" : "Mumbai", "age" : 2 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 4000, "eaddr" : "Hyderabad", "age" : 2 }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 4000, "eaddr" : "Mumbai", "age" : 2 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" : "Delhi", "age" : 2 }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" : "Chennai", "age" : 2 }
```

```
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 5000, "eaddr" :
"Mumbai", "age" : 2 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 5000, "eaddr" :
"Delhi", "age" : 2 }
```

Note: If the specified field does not exist, then $max operator creates that field and assign with provided value.

```
> db.employees.updateMany({},{$max:{height:5.8}})

> db.employees.updateMany({},{$max:{height:5.8}})
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 7 }
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 4000, "eaddr" :
"Mumbai", "age" : 2, "marks" : 99, "height" : 5.8 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 4000, "eaddr" :
"Hyderabad", "age" : 2, "marks" : 99, "height" : 5.8 }
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 4000, "eaddr" :
"Mumbai", "age" : 2, "marks" : 99, "height" : 5.8 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 4500, "eaddr" :
"Delhi", "age" : 2, "marks" : 99, "height" : 5.8 }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai", "age" : 2, "marks" : 99, "height" : 5.8 }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 5000, "eaddr" :
"Mumbai", "age" : 2, "marks" : 99, "height" : 5.8 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 5000, "eaddr" :
"Delhi", "age" : 2, "marks" : 99, "height" : 5.8 }
```

Note:

1. If provided value is less than current value then only perform updation-->min operator.

2. If provided value is greater than current value then only perform updation-->max operator.

Q. what if we don't want min/max operator to create the new field? Select all documents where specified field exists and then perform updation.

```
> db.employees.update({age:{$exists:true}},{$set:{age:15}},{multi:true})
WriteResult({ "nMatched" : 7, "nUpserted" : 0, "nModified" : 7 })


>
db.employees.update({phone_number:{$exists:true}},{$set:{phone_num
ber:1234}},{multi:true})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
```

phone_number field not created.

7. $mul operator:

----

mul means multiplication

We can use $mul operator to multiply the value of a field by a number.

{$mul: {field: number}}
The field to update must contain numeric value.

Q1. Double all employee salary where esal is less than 4900?

```
> db.employees.updateMany({esal: {$lt: 4900}},{$mul: {esal: 2}})
{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }
```

Q2. Increment Salary by 10% for all employees belongs to Mumbai?

```
> db.employees.updateMany({eaddr: "Mumbai"},{$mul: {esal: 1.1}})
  { "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

1000--->1100

10000--->11000

Note:
If the specified field is not available then $mul creates that field and sets the value to zero.

```
> db.employees.updateMany({},{$mul: {xyz: 3,abc: 3.5}})
{ "acknowledged" : true, "matchedCount" : 8, "modifiedCount" : 8 }
> db.employees.find()
...
{ "_id" : 10, "ename" : "Durga", "eaddr" : "Hyderabad", "eno" : 1000,
"esal" : 10000, "abc" : 0, "xyz" : 0 }
```

Note:
1. $set --->To set a new value to the existing field or to create new field
2. $unset--->To unset/delete existing field
3. $rename-->To rename the value of the field.
4. $inc --->To increment or decrement field value
5. $min --->To update only if the provided value is less than current value
6. $max --->To update only if the provided value is greater than current value
7. $mul -->To multiply field value by a number

Understanding upsert property:

Whenever we are trying to perform update operation, the matched document may or may not available.If it is available then it will be updated and if it is not available then update won't be happend.

If the document not available then we can insert that document in the database automatically. For this we have to use upsert property.

upsert = update + insert

at the time two works update and insert. First update and if it is not possible then insert.

upsert will take boolean value.
If it is set to true, it will creates a new document if it is not available.
If it is set to false, then it will perform just update operation and won't create any new document.
The default value of upsert is: false.

Demo Execution:

```
>db.employees.find()
> db.employees.find()
{ "_id" : 1, "eno" : 100, "ename" : "Sunny", "esal" : 8800, "eaddr" :
"Mumbai", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0 }
{ "_id" : 2, "eno" : 200, "ename" : "Bunny", "esal" : 8000, "eaddr" :
"Hyderabad", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0
}
{ "_id" : 3, "eno" : 300, "ename" : "Chinny", "esal" : 8800, "eaddr" :
"Mumbai", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0 }
{ "_id" : 4, "eno" : 400, "ename" : "Vinny", "esal" : 9000, "eaddr" :
"Delhi", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0 }
{ "_id" : 5, "eno" : 500, "ename" : "Pinny", "esal" : 5000, "eaddr" :
"Chennai", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0 }
{ "_id" : 6, "eno" : 600, "ename" : "Tinny", "esal" : 5500, "eaddr" :
"Mumbai", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0 }
{ "_id" : 7, "eno" : 700, "ename" : "Zinny", "esal" : 5000, "eaddr" :
"Delhi", "age" : 15, "marks" : 99, "height" : 5.8, "abc" : 0, "xyz" : 0 }
{ "_id" : 10, "ename" : "Durga", "eaddr" : "Hyderabad", "eno" : 1000,
"esal" : 10000, "abc" : 0, "xyz" : 0 }
```

```
> db.employees.update({ename: "Mallika"}, {$set:
{_id:11,esal:9999,eaddr:"Mumabi"}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })


> db.employees.update({ename: "Mallika"}, {$set:
{_id:11,esal:9999,eaddr:"Mumabi"}}, {upsert: false})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })

> db.employees.update({ename: "Mallika"}, {$set:
{_id:11,esal:9999,eaddr:"Mumabi"}}, {upsert: true})
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 11
})

> db.employees.find()
...
{ "_id" : 11, "ename" : "Mallika", "eaddr" : "Mumabi", "esal" : 9999 }
```

Array Update Operators:

---

1. $ 2. $[] 3. $[<identifier>]

1. Updating First Matched element by using $:

$ acts as a placeholder to update first matched element based on query condition.

Syntax:

```
db.collection.update(query,{update_operator:{"<array>.$" : value}})
```

The array field must appear as the part of query condition.

Case Study:

```
db.students.insertOne({_id:1, marks: [70,87,90,30,40]})
db.students.insertOne({_id:2, marks: [90,88,92,110,45]})
db.students.insertOne({_id:3, marks: [85,100,90,76,58]})
db.students.insertOne({_id:4, marks: [79,85,80,89,56]})
db.students.insertOne({_id:5, marks: [88,88,92,45,23]})
db.students.insertOne({_id:6, marks: [95,90,96,92,95]})
```

Q1. Update the first matched element 90 in marks array to 999 where _id:1?

```
> db.students.update({_id:1,marks:90},{$set: {"marks.$": 999}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.students.find()
{ "_id" : 1, "marks" : [ 70, 87, 90, 30, 40 ] }
{ "_id" : 2, "marks" : [ 90, 88, 92, 110, 45 ] }
{ "_id" : 3, "marks" : [ 85, 100, 90, 76, 58 ] }
{ "_id" : 4, "marks" : [ 79, 85, 80, 89, 56 ] }
{ "_id" : 5, "marks" : [ 88, 88, 92, 45, 23 ] }
{ "_id" : 6, "marks" : [ 95, 90, 96, 92, 95 ] }
> db.students.update({_id:1,marks:90},{$set: {"marks.$": 999}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.students.find()
{ "_id" : 1, "marks" : [ 70, 87, 999, 30, 40 ] }
{ "_id" : 2, "marks" : [ 90, 88, 92, 110, 45 ] }
{ "_id" : 3, "marks" : [ 85, 100, 90, 76, 58 ] }
{ "_id" : 4, "marks" : [ 79, 85, 80, 89, 56 ] }
{ "_id" : 5, "marks" : [ 88, 88, 92, 45, 23 ] }
{ "_id" : 6, "marks" : [ 95, 90, 96, 92, 95 ] }
```

Q2. Update the first matched element in marks array which is less than 90 with 90 in every document?

```
> db.students.updateMany({marks:{$elemMatch:{$lt:90}}},{$set:
{"marks.$": 90}})


> db.students.find()
{ "_id" : 1, "marks" : [ 70, 87, 999, 30, 40 ] }
{ "_id" : 2, "marks" : [ 90, 88, 92, 110, 45 ] }
{ "_id" : 3, "marks" : [ 85, 100, 90, 76, 58 ] }
{ "_id" : 4, "marks" : [ 79, 85, 80, 89, 56 ] }
{ "_id" : 5, "marks" : [ 88, 88, 92, 45, 23 ] }
{ "_id" : 6, "marks" : [ 95, 90, 96, 92, 95 ] }
> db.students.updateMany({marks:{$elemMatch:{$lt:90}}},{$set:
{"marks.$": 90}})
{ "acknowledged" : true, "matchedCount" : 5, "modifiedCount" : 5 }
> db.students.find()
{ "_id" : 1, "marks" : [ 90, 87, 999, 30, 40 ] }
{ "_id" : 2, "marks" : [ 90, 90, 92, 110, 45 ] }
{ "_id" : 3, "marks" : [ 90, 100, 90, 76, 58 ] }
{ "_id" : 4, "marks" : [ 90, 85, 80, 89, 56 ] }
{ "_id" : 5, "marks" : [ 90, 88, 92, 45, 23 ] }
{ "_id" : 6, "marks" : [ 95, 90, 96, 92, 95 ] }
```

2. Updating all array elements by using $[]:

$[] acts as placeholder to update all elements in the array for the matched documents based on query condition.

Syntax:

```
db.collection.update(query,{update_operator:{"<array>.$[]" : value}})
```

Q1. To increment every element of marks array by 10?

```
> db.students.updateMany({},{$inc: {"marks.$[]": 10}})
> db.students.find()
{ "_id" : 1, "marks" : [ 90, 87, 999, 30, 40 ] }
{ "_id" : 2, "marks" : [ 90, 90, 92, 110, 45 ] }
{ "_id" : 3, "marks" : [ 90, 100, 90, 76, 58 ] }
{ "_id" : 4, "marks" : [ 90, 85, 80, 89, 56 ] }
{ "_id" : 5, "marks" : [ 90, 88, 92, 45, 23 ] }
{ "_id" : 6, "marks" : [ 95, 90, 96, 92, 95 ] }
> db.students.updateMany({},{$inc: {"marks.$[]": 10}})
{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 6 }
> db.students.find()
{ "_id" : 1, "marks" : [ 100, 97, 1009, 40, 50 ] }
{ "_id" : 2, "marks" : [ 100, 100, 102, 120, 55 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 86, 68 ] }
{ "_id" : 4, "marks" : [ 100, 95, 90, 99, 66 ] }
{ "_id" : 5, "marks" : [ 100, 98, 102, 55, 33 ] }
{ "_id" : 6, "marks" : [ 105, 100, 106, 102, 105 ] }
```

Q2. Update every element of marks array as 1000 if array contains atleat one element which is greater than or equal to 1000?

```
> db.students.updateMany({marks:{$elemMatch:{$gte:1000}}},{$set:
{"marks.$[]": 1000}})


> db.students.find()
{ "_id" : 1, "marks" : [ 100, 97, 1009, 40, 50 ] }
{ "_id" : 2, "marks" : [ 100, 100, 102, 120, 55 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 86, 68 ] }
```

```
{ "_id" : 4, "marks" : [ 100, 95, 90, 99, 66 ] }
{ "_id" : 5, "marks" : [ 100, 98, 102, 55, 33 ] }
{ "_id" : 6, "marks" : [ 105, 100, 106, 102, 105 ] }
> db.students.updateMany({marks:{$elemMatch:{$gte:1000}}},{$set:
{"marks.$[]": 1000}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.students.find()
{ "_id" : 1, "marks" : [ 1000, 1000, 1000, 1000, 1000 ] }
{ "_id" : 2, "marks" : [ 100, 100, 102, 120, 55 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 86, 68 ] }
{ "_id" : 4, "marks" : [ 100, 95, 90, 99, 66 ] }
{ "_id" : 5, "marks" : [ 100, 98, 102, 55, 33 ] }
{ "_id" : 6, "marks" : [ 105, 100, 106, 102, 105 ] }
```

3. Updating specific array elements by using $[identifier]:

Instead of updating only first matched element or all elements of the array, we can update only required array elements. For this we have to use $[identifier]

$[identifier] --->Acts as placeholder to update all elements that match the arrayFilters condition for the documents that match query condition.

In this case updation is based on arrayFilters condition but not based on query condition.

Syntax:

Q. Update all marks array elements which are less than 100 as 100?

```
> db.students.updateMany({},{$set: {"marks.$[element]":
100}},{arrayFilters:[{"element": {$lt: 100}}]})

> db.students.find()
{ "_id" : 1, "marks" : [ 1000, 1000, 1000, 1000, 1000 ] }
{ "_id" : 2, "marks" : [ 100, 100, 102, 120, 55 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 86, 68 ] }
{ "_id" : 4, "marks" : [ 100, 95, 90, 99, 66 ] }
{ "_id" : 5, "marks" : [ 100, 98, 102, 55, 33 ] }
{ "_id" : 6, "marks" : [ 105, 100, 106, 102, 105 ] }
> db.students.updateMany({},{$set: {"marks.$[element]":
100}},{arrayFilters:[{"element": {$lt: 100}}]})
{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 4 }
> db.students.find()
{ "_id" : 1, "marks" : [ 1000, 1000, 1000, 1000, 1000 ] }
{ "_id" : 2, "marks" : [ 100, 100, 102, 120, 100 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 100, 100 ] }
{ "_id" : 4, "marks" : [ 100, 100, 100, 100, 100 ] }
{ "_id" : 5, "marks" : [ 100, 100, 102, 100, 100 ] }
{ "_id" : 6, "marks" : [ 105, 100, 106, 102, 105 ] }
```

Q. If we specify both query and arrayFilters in this case what is the order?

Ans:

query condition helpful to select documents.

In those selected documents, based on arrayFilters condition array elements will be updated.

Q2. Write query to perform the following update?

If the marks in the range 101 to 110 make as 110.

```
> db.students.updateMany({},{$set: {"marks.$[e1]": 110}},{arrayFilters:
[{$and: [{"e1":{$gt:100}},{"e1":{$lte:110}}]}]})


> db.students.find()
{ "_id" : 1, "marks" : [ 1000, 1000, 1000, 1000, 1000 ] }
{ "_id" : 2, "marks" : [ 100, 100, 102, 120, 100 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 100, 100 ] }
{ "_id" : 4, "marks" : [ 100, 100, 100, 100, 100 ] }
{ "_id" : 5, "marks" : [ 100, 100, 102, 100, 100 ] }
{ "_id" : 6, "marks" : [ 105, 100, 106, 102, 105 ] }
> db.students.updateMany({},{$set: {"marks.$[e1]": 110}},{arrayFilters:
[{$and: [{"e1":{$gt:100}},{"e1":{$lte:110}}]}]})
{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 3 }
> db.students.find()
{ "_id" : 1, "marks" : [ 1000, 1000, 1000, 1000, 1000 ] }
{ "_id" : 2, "marks" : [ 100, 100, 110, 120, 100 ] }
{ "_id" : 3, "marks" : [ 100, 110, 100, 100, 100 ] }
{ "_id" : 4, "marks" : [ 100, 100, 100, 100, 100 ] }
{ "_id" : 5, "marks" : [ 100, 100, 110, 100, 100 ] }
{ "_id" : 6, "marks" : [ 110, 100, 110, 110, 110 ] }
```

Q3. Consider the students collection:

```
db.students.insertOne({_id:1,marks:[70,87,90]})
db.students.insertOne({_id:2,marks:[90,88,92]})
db.students.insertOne({_id:3,marks:[85,100,90]})
db.students.insertOne({_id:4,marks:[79,85,80]})
db.students.insertOne({_id:5,marks:[88,88,92]})
db.students.insertOne({_id:6,marks:[95,90,96]})
```

Write query to perform the following update?

If the marks in the range 71 to 80 make as 80

If the marks in the range 81 to 90 make as 90

If the marks in the range 91 to 100 make as 100

```
> db.students.updateMany({},{$set: {"marks.$[e1]": 80, "marks.$[e2]":
90, "marks.$[e3]": 100}},{arrayFilters: [ {$and:
[{"e1":{$gt:70}},{"e1":{$lte:80}}]},    {$and:
[{"e2":{$gt:80}},{"e2":{$lte:90}}]}, {$and: :{$gt:90}},{"e3":{$lte:100}}]}])



db.students.updateMany(
    {},
    {
      $set: {
          "marks.$[e1]": 80,
          "marks.$[e2]": 90,
          "marks.$[e3]": 100
         }
    },
    {
       arrayFilters: [
           {$and: [{"e1":{$gt:70}},{"e1":{$lte:80}}]},
           {$and: [{"e2":{$gt:80}},{"e2":{$lte:90}}]},
           {$and: [{"e3":{$gt:90}},{"e3":{$lte:100}}]}
           ]
    }
 )
{ "acknowledged" : true, "matchedCount" : 6, "modifiedCount" : 6 }
> db.students.find()
{ "_id" : 1, "marks" : [ 70, 90, 90 ] }
{ "_id" : 2, "marks" : [ 90, 90, 100 ] }
{ "_id" : 3, "marks" : [ 90, 100, 90 ] }
{ "_id" : 4, "marks" : [ 80, 90, 80 ] }
{ "_id" : 5, "marks" : [ 90, 90, 100 ] }
```

{ "_id" : 6, "marks" : [100, 90, 100] }

Note:
$ --->To update only first matched element of array
$[] --->To update all elements of array
$[identifier] ---> To update specific array elements

4. Adding elements to the array by using $push operator:

We can use $push operator to add elements to the array.

By default element will be added at the end, but based on our requirement we can add in our required position also.

Syntax:
db.collection.update({},{$push: {<array1>: value1,...}})

eg-1: Adding a single element:

{ "_id" : 1, "marks" : [70, 90, 90] }

> db.students.update({_id:1}, {$push: {marks: 35}})

{ "_id" : 1, "marks" : [70, 90, 90, 35] }

eg-2: Adding multiple elements:

1st way:

> db.students.update({_id:1}, {$push: {marks: 36,marks: 37, marks: 38}})
{ "_id" : 1, "marks" : [70, 90, 90, 35, 38] }

It is not added 3 elements only one element added.
Reason: In Javascript object, duplicate keys are not allowed. If we are trying to add entry with duplicate key, old value will be replaced with new value.
{marks: 36,marks: 37, marks: 38} ===>{marks: 38}

2nd way:

```
> db.students.update({_id:1}, {$push: {marks: [39,40,41]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.students.find()
{ "_id" : 1, "marks" : [ 70, 90, 90, 35, 38, [ 39, 40, 41 ] ] }
```

Total array added as single element.

We can add elements of the array individually by using $each modifier.

$each modifier:

We can use $each modifier to add multiple values to the array.
Syntax:
{ $push: { <array>: {$each: [value1,value2,...]} }

```
> db.students.update({_id:1}, {$push: {marks: {$each: [42,43,44]}}})

> db.students.update({_id:1}, {$push: {marks: {$each: [42,43,44]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.students.find()
{ "_id" : 1, "marks" : [ 70, 90, 90, 35, 38, [ 39, 40, 41 ], 42, 43, 44 ] }
```

$position modifier:

Bydefault elements will be added at the end of the array. But we can add elements in the required position. For this we have to use $position modifier.

To use $position modifier, compulsory we should use $each modifier. i.e $position without $each is always invalid.

Syntax:

```
{
  $push: {
     <array>: {
        $each: [value1,value2,value3],
         $position: <num>
     }
  }
}
```

<num> indicates the position where we have to add element. Array follows zero based index. ie index of first element is 0.

eg-1: To add element at the beginning:

We have to use <num> as 0.

```
> db.students.insert({_id:7,marks:[10,20,30,40]})

> db.students.update(
         {_id:7},
          {
             $push: {
                marks: {
                     $each:[50],
                     $position:0
                     }
```

```
            }
        }
)

{ "_id" : 7, "marks" : [ 50, 10, 20, 30, 40 ] }
```

eg-2: To add element at 3rd index place:

```
----------------------------------------
> db.students.update(
        {_id:7},
        {
          $push: {
            marks: {
                $each:[60],
                $position:3
                }
            }
        }
)

{ "_id" : 7, "marks" : [ 50, 10, 20, 60, 30, 40 ] }
```

Negative Index:

We can use negative index to add elements from the end. -1 indicates the position just before last element in the array.

eg-1:

```
> db.students.update(
        {_id:7},
        {
          $push: {
            marks: {
```

```
                $each:[70],
                $position:-1
                }
            }
        }
)
{ "_id" : 7, "marks" : [ 50, 10, 20, 60, 30, 70, 40 ] }

eg-2:
db.students.update(
        {_id:7},
        {
          $push: {
            marks: {
                $each:[80],
                $position:-2
                }
            }
        }
)
{ "_id" : 7, "marks" : [ 50, 10, 20, 60, 30, 80, 70, 40 ] }


Q. IF NUM > size of array , then will it be added at the end ?
Yes

eg-3:
db.students.update(
        {_id:7},
        {
          $push: {
            marks: {
                $each:[1,2,3],
```

```
                $position:-3
                }
            }
        }
)
{ "_id" : 7, "marks" : [ 50, 10, 20, 60, 30, 1, 2, 3, 80, 70, 40 ] }
```

Note: with a negative index position, if we specify multiple elements in the $each array, the last added element is in the specified position from the end.

$sort modifier:

We can use $sort modifier to sort elements of the array while performing push operation.

To use $sort modifier, we should use $each modifier. i.e without $each, we cannot use $sort modifier.

We can pass empty array [], to $each modifier to see effect of only $sort.

Syntax:

```
db.collection.update(
      {},
      {
        $push: {
            <array>: { $each: [value1,value2,..],
                    $sort: 1|-1
                    }
          }
```

})

1 means ascending order
-1 means descending order

eg-1: Sorting array elements according to ascending order:

{ "_id" : 7, "marks" : [10, 20, 60, 30, 1, 2, 3, 80, 70, 40, 897, 98, 99, 100, 34, 35, 36] }

> db.students.update({_id:7},{$push: {marks: {$each:[15,25,10],$sort: 1}}})

{ "_id" : 7, "marks" : [1, 2, 3, 10, 10, 15, 20, 25, 30, 34, 35, 36, 40, 60, 70, 80, 98, 99, 100, 897] }

eg-2: Sorting without adding any element:

> db.students.update({_id:7},{ $push: {marks: {$each:[],$sort: -1}}})
{ "_id" : 7, "marks" : [897, 100, 99, 98, 80, 70, 60, 40, 36, 35, 34, 30, 25, 20, 15, 10, 10, 3, 2, 1] }

$slice modifier:

The $slice modifier limits the number of array elements during $push operation.

To use $slice modifier, we should use $each modifier. i.e without $each, we cannot use $slice modifier.

We can pass empty array [], to $each modifier to see effect of only $slice modifier.

Syntax:

```
------
db.collection.update(
        {},
        {
          $push: {
              <array>: { $each: [value1,value2,..],
                        $slice: <num>
                         }
           }

        })
```

The <num> can be:

1. zero --->To update array to an empty array.
2. positive --->To update array field to contain only first <num> elements.
3. Negative --->To update array field to contain only last <num> elements.

eg-1: To update array with last 6 elements

```
----------------------------------------
{ "_id" : 7, "marks" : [ 897, 100, 99, 98, 80, 70, 60, 40, 36, 35, 34, 30, 25, 20, 15, 10, 10, 3, 2, 1 ] }

> db.students.update({_id: 7},{$push: {marks: {$each:[5,6,7],$slice: -6}}})

{ "_id" : 7, "marks" : [ 3, 2, 1, 5, 6, 7 ] }
```

eg-2: To update array with first 3 elements:

```
> db.students.update({_id: 7},{$push: {marks: {$each:[],$slice: 3}}})
{ "_id" : 7, "marks" : [ 3, 2, 1 ] }
```

eg-3: To update array with zero number of elements:

```
> db.students.update({_id: 7},{$push: {marks: {$each:[],$slice: 0}}})
{ "_id" : 7, "marks" : [ ] }
```

The effect of order of modifiers:

Order of modifiers in the query is not important and we can take in any order.

But MongoDB Server will process push operation in the following order:

1. **Update array to add elements in the correct position.**
2. **Apply sort, if specified.**
3. **slice the array, if specified.**
4. **Store the array**

eg:

```
> db.students.update(
   {_id:7},
   {
      $push: {marks: {$slice:3, $sort: -1, $each:[4,1,7,2,6,3,9,2,8,4,5]}}
   }
 )

{ "_id" : 7, "marks" : [ 9, 8, 7 ] }
```

Note:

1. If the spcified array is not already available then $push adds that array field with values as its elements.

```
> db.students.update({_id:7},{$push: {marks1: {$each: [10,20,30]}}})
{ "_id" : 7, "marks" : [ 9, 8, 7 ], "marks1" : [ 10, 20, 30 ] }
```

2. If the field is not array, then $push operation will fail.

```
> db.students.update({_id:7},{$set: {name:"Durga"}})
{ "_id" : 7, "marks" : [ 9, 8, 7 ], "marks1" : [ 10, 20, 30 ], "name" :
"Durga" }
> db.students.update({_id:7},{$push: {name:{$each: [10,20,30]}}})
"errmsg" : "The field 'name' must be an array but is of type string in
document {_id: 7.0}"
```

Summary:

$push operator --->To add elements to array.

$each modifier --->To add multiple elements

$position modifier --->To add elements at specified position

$sort modifier --->To sort elements after addition

$slice modifier --->To limit the number of elements.

5. $addToSet operator:

It is exactly same as $push operator except that it won't allow duplicates.

It adds elements to the array iff array does not contain already those elements.

There is no effect on already existing duplicates.

case study:

```
db.students.insertOne({_id:1,marks:[70,87,90]})
db.students.insertOne({_id:2,marks:[90,88,92]})
db.students.insertOne({_id:3,marks:[85,100,90]})
db.students.insertOne({_id:4,marks:[79,85,80]})
db.students.insertOne({_id:5,marks:[88,88,92]})
db.students.insertOne({_id:6,marks:[95,90,96]})
```

eg-1: Adding duplicate element

```
{_id:5,marks:[88,88,92]}
> db.students.update({_id:5},{$addToSet: {marks: 88}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
{ "_id" : 5, "marks" : [ 88, 88, 92 ] }
```

In this case 88 won't be added because it is already available.

eg-2: Adding non-duplicate element:

```
> db.students.update({_id:5},{$addToSet: {marks: 90}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
{ "_id" : 5, "marks" : [ 88, 88, 92, 90 ] }
```

eg-3: Adding multiple elements:

To add multiple elements we have to use $each modifier.

```
> db.students.update({_id:5},{$addToSet: {marks: {$each:
[10,20,88,90,30]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

{ "_id" : 5, "marks" : [88, 88, 92, 90, 10, 20, 30] }

Note: In the case of $push operator, order terminology is applicable. Hence we can use $position, $sort, $slice modifiers.

But in the case of $addToSet operator, order terminology is not applicable. Hence we cannot use $position, $sort, $slice modifiers.

But $each modifier applicable for both $push and $addToSet operators.

> db.students.update({_id:5},{$addToSet: {marks: {$each: [7,8,9], $position: 2}}})

"errmsg" : "Found unexpected fields after $each in $addToSet: { $each: [7.0, 8.0, 9.0], $position: 2.0 }"

6. Removing Elements by using $pop operator:

We can use $pop operator to remove either first or last element from the array.

Syntax:

```
{
  $pop: {<array>:-1|1}
}
```

-1 --->To remove the first element

1 --->To remove the last element

eg-1: To remove first element:

```
----------------------------
{ "_id" : 5, "marks" : [ 88, 88, 92, 90, 10, 20, 30 ] }
> db.students.update({_id: 5},{$pop: {marks: -1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
{ "_id" : 5, "marks" : [ 88, 92, 90, 10, 20, 30 ] }
```

eg-2: To remove last element:

```
----------------------------
> db.students.update({_id: 5},{$pop: {marks: 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1
{ "_id" : 5, "marks" : [ 88, 92, 90, 10, 20 ] }
```

7. Remove elements by using $pull operator:

```
-----------------------------------------
```

We can use $pull operator either

1. To remove all instances of specified element.
2. To remove elements that match the given condition.

Syntax:

```
-------
{
  $pull: {<array>: <value> | <condition> }
}
```

eg-1: To delete all instances of 10

```
----------------------------------
{ "_id" : 7, "marks" : [ 10, 20, 30, 10, 20, 40, 50, 60, 70, 80, 90, 100 ] }
> db.students.update({_id:7}, {$pull: {marks: 10}})
```

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
{ "_id" : 7, "marks" : [20, 30, 20, 40, 50, 60, 70, 80, 90, 100] }

eg-2: To remove all elements which are greater than or equal to 50

> db.students.update({_id:7}, {$pull: {marks: {$gte: 50}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
{ "_id" : 7, "marks" : [20, 30, 20, 40] }

8. Removing elements by using $pullAll operator:

By using $pull, we can delete either all instances of a single element or elements based on some condition.

But by using $pullAll, we can delete all instances of given list of multiple elements.

Syntax:
{
$pullAll: {<array>: [value1, value2, value3, ...]}
}

eg:
> db.students.insert({_id:8,
marks:[10,20,10,10,20,20,10,30,30,40,50,60]})
> db.students.update({_id:8},{$pullAll: {marks: [10,20,30,40]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
{ "_id" : 8, "marks" : [50, 60] }

Summary of array update operators:
----
1. $

2. $[]
3. $[element]
4. $push operators with modifiers: $each, $position, $sort, $slice
5. $addToSet operator with $each modifier
6. $pop
7. $pull
8. $pullAll

CRUD Operations:

C --->Create|Insert
R --->Retrieve | Read
U --->Update
D --->Delete

Deleting Documents from the collection:

MongoDB provides the following methods to delete documents from the collection.

1. deleteOne()
2. deleteMany()
3. remove()

1. deleteOne():