Questions:

Update document

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,

   }
)
```

Name	Description
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp.
$inc	Increments the value of the field by the specified amount.
$min	Only updates the field if the specified value is less than the existing field value.
$max	Only updates the field if the specified value is greater than the existing field value.
$mul	Multiplies the value of the field by the specified amount.
$rename	Renames a field.
$set	Sets the value of a field in a document.
$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.
$unset	Removes the specified field from a document.

Array

Operators

Name	Description
$	Acts as a placeholder to update the first element that matches the query condition.
$[]	Acts as a placeholder to update all elements in an array for the documents that match the query condition.
$[<identifier>]	Acts as a placeholder to update all elements that match the `arrayFilters` condition for the documents that match the query condition.
$addToSet	Adds elements to an array only if they do not already exist in the set.
$pop	Removes the first or last item of an array.
$pull	Removes all array elements that match a specified query.

Name	Description
$push	Adds an item to an array.
$pullAll	Removes all matching values from an array.
Name	Description
$each	Modifies the $push and $addToSet operators to append multiple items for array updates.
$position	Modifies the $push operator to specify the position in the array to add elements.
$slice	Modifies the $push operator to limit the size of updated arrays.
$sort	Modifies the $push operator to reorder documents stored in an array.

db.users.insert({ _id: 1, status: "a", lastModified: ISODate("2013-10-02T01:11:18.965Z") })

1. To update one field

```
db.Employee.update(
{"Employeeid" : 1},
{$set: { "EmployeeName" : "NewMartin"}});
```

2. To update multiple value

```
db.Employee.update
     (
             {
                     Employeeid : 1
             },
             {
                     $set :
                     {
                     "EmployeeName" : "NewMartin"
                     "Employeeid" : 22
                     }
             },{multi : true}

     )
```

3.

```
db.users.update(
  { _id: 1 },
  {
   $currentDate: {
     lastModified: true,
     "cancellation.date": { $type: "timestamp" }
   },
   $set: {
     status: "D",
     "cancellation.reason": "user request"
   }
  }
)
```

Inventory examples

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status:
"A" },
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status:
"A" },
   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status:
"A" },
   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" },
status: "P" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status:
"P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status:
"D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },
status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },
status: "A" },
   { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" },
status: "A" },
   { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" },
status: "A" }
] );
```

Using updateone

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

```
Using updatemany
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

```
To replace first document
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse:
"B", qty: 40 } ] }
)
```

To remove value of field
db.movie.update({name:'padmavat'},{$unset:{'rating':""}}) ---will delete rating only for 1st record

To delete from all documents
db.movie.update({name:'padmavat'},{$unset:{'rating':""}},{multi:True)

To add data in the array
db.movie.update({name:'padmavat'},{$push:{actors:"raza murad"}})

db.movie.update({name:'padmavat'},{$push:{actors:{$each:["raza murad",”aditi rao”]}}})

------will add at the beginning because given position is 0
db.movie.update({name:'padmavat'},{$push:{actors:{$each:["raza murad",”aditi rao”],$position: 0}}})

```
db.movie.updateOne(
   { _id: 1, actors: ‘raza murad’ },
   { $set: { "actors.$" : ‘xxx’ } }
)
```

```
{
  _id: 4,
  grades: [
     { grade: 80, mean: 75, std: 8 },
     { grade: 85, mean: 90, std: 5 },
     { grade: 85, mean: 85, std: 8 }
  ]
}
```

$ indicates the matched record
db.students.updateOne(
  { _id: 4, "grades.grade": 85 },
  { $set: { "grades.$.std" : 6 } }
)

$[] – all the values in the array $inc ---increament values

```
{ "_id" : 1, "grades" : [ 85, 82, 80 ] }
{ "_id" : 2, "grades" : [ 88, 90, 92 ] }
{ "_id" : 3, "grades" : [ 85, 100, 90 ] }
```

To increase all the vauses in grade array by 10

```
db.students.update(
   { },
   { $inc: { "grades.$[]": 10 } },
   { multi: true }
)
```

```
{
   "_id" : 1,
   "grades" : [
      { "grade" : 80, "mean" : 75, "std" : 8 },
      { "grade" : 85, "mean" : 90, "std" : 6 },
      { "grade" : 85, "mean" : 85, "std" : 8 }
   ]
}
{
```

```
   "_id" : 2,
   "grades" : [
      { "grade" : 90, "mean" : 75, "std" : 8 },
      { "grade" : 87, "mean" : 90, "std" : 5 },
      { "grade" : 85, "mean" : 85, "std" : 6 }
   ]
}
```

To decrease all values of stat

```
db.students2.update(
   { },
   { $inc: { "grades.$[].std" : -2 } },
   { multi: true }
)
```

$pop – delete last element

```
db.students.update( { _id: 1 }, { $pop: { scores: 1 } } )
```

To remove 1 st element specify -1 and use 1 for deleteing last element

```
db.students.update( { _id: 1 }, { $pop: { scores: -1 } } )
```

The $pull operator removes from an existing array all instances of a value or values that match a specified condition.

Remove all matching values
Removes apple and oranges from fruits and carrots from vegetables

```
db.stores.update(
    { },
    { $pull: { fruits: { $in: [ "apples", "oranges" ] }, vegetables:
"carrots" } },
    { multi: true }
)
```

To create index on rating if it does not exists
rating:1 --- ascending rating:-1 -----descending
Db.movie.ensureIndex({rating:1})

To create composit index
Db.movie.ensureIndex({rating:1,name:-1})

---All indexes are stored in system.indexes collections
----to delete index
Db.movie.dropIndex(name of index)

To view indexes
Db.movie.getIndexs()

To remove documents
Db.movie.remove() ------remove all documents
Db.movie.remove(criteria) ----remove documents that match the criteria

---

Aggregate function

```
db.articles.aggregate([{$project:{author:1},
{$group:{"_id":$author,count:{$sum:1}},
{$sort:{count:-1}},
{$limit:5}])

db.movie.aggregate([{$match:{price:{$gt:200}}},
{$project:{rating:1,name:1,price:1,addition:{$add:['$price','$rating']}}},
{$sort:{addition:-1}},
{$skip:1},
{$limit:1}

]).pretty()
```

To filter documents
$match

In $project we may want derived column then we use
---------Number function
$add :[expr1[,expr2…….]
$subtract:[expr1,expr2]
$multiply:[expr1[,expr2…….]
$divide:[expr1,expr2]
$mod : [expr1,expr2]

----------String related function
$substr : [expr,start offset,number of character to return]
$concat: [expr 1[,expr 2,expr 3,expr 4, …….,exprn]
$toLower : expr
$toUpper : expr

1. To display all names in lowercase

```
db.movie.aggregate([{$match:{rating:{$gt:3}}},

{$project:{name:{$toLower:'$name'}}}])

db.movie.aggregate([{$project:{name:{ {$toLower:{{$substr:[$name},0,2]    }}

}}}])
```

2. To display number of years of experience

```
db.employee.aggregate([{$project:{"experience":{$subtract:{[
{$year:'new Date()'},{$year:'$hierdate'}
```

]}}}}])

To display name and sum of rating and price
```
db.movie.aggregate(
  [
   { $project: { name: 1, total: { $add: [ "$rating", "$price" ] } } }
  ]
)
```

To display email address as concatenation of firstname and last name with .gamil.com

```
db.employee.aggregate([{$project:{
“email”:{$concat:[{$substr:[$firstname,0,3]},”.”,”$lastname”,”@gmail.com”]} }}])
```
---

we may us Logical expressions

$cmp:[expr1,expr2] ------- returns -ve no zero or positive number based on < = or >
$strcasecmp : [string1,string2]------only work for roman character

$eq:[expr1,expr2]
$ne,$gt,$gte,$lte,$lt ------- returns true or false

Boolean expression

$and:[expr[,expr2,expr3……]]
$or:[expr[,expr2,expr3……]]
$not:[expr]

Two control statements

$cond:[booleanExpr,truetxprs,falseexprs]
e.g
```
db.employee.aggregate([{$project:{name:1,"itemfound":{$cond:{if:{ "item: {$exists:true}
}}}}}])
```

```
db.employee.aggregate([{$project:{name:1,
"itemfound":{$cond:{if:[$eq[deptno:10],then: "ite is 10",else:"it is other"}}}}]);
```

```
db.restaurants.aggregate({$unwind:"$grades"},
{$project:{"year":{$year:"$grades.date"}}});
```

$ifNull:[expr,replacement exprn]
$ifNull:[‘$comm’,’0’,’$comm’]

```
db.employee.aggregate([
{$project:{name:1,comm:{$ifNull['$comm',0,'$comm']}}


}])
```

$group – It allows you to group the document based on certain field

------to find min and max marks for each subject in collection
db.student.aggregate([{$group:{_id:"$special-sub","min marks":{$min:'$marks'},"max marks":{$max:'$marks'} }}}])

Grouping operators are
$sum:value
$max:expr,
$avg:expr
$min:expr
$first:expr----------first value from each group
$last:expr----------last value in each group

```
Db.employee.aggregate([
{$group:{"_id":"dept.deptid","minsal":{$min:'$sal'}, "max sal":{$max:'$sal'},"count":{$sum:1},"sumsal":{$sum:'$sal'} ,{$sort:{"_id":1}}  }
])


Db.employee.aggregate([
{$group:{"_id":null,"sumsal":{$sum:'$sal'}}},
{$project:{sumsal:1}}

])
```

$unwind operator turns each field of array into separate document
e.g
```
>db.blog.post.findOne()
{_id:Objectid(--------),author:"a",post:"hello"
Comment:[
    {author :"abc" ,    date :ISODATE(2018-04-30T17:52:04.148z),text:"nicepost"},
    {author :"pqr" ,    date :ISODATE(2018-04-30T17:52:04.148z),text:"goodone post"}

]
```

>db.blog.post.aggregate([{
$unwind:”comments”
}])
{result :
{_id:Object id(------),author :”abc” , date :ISODATE(2018-04-30T17:52:04.148z),text:”nicepost”},
{_id:Object id(------),author :”pqr” , date :ISODATE(2018-04-30T17:52:04.148z),text:”goodone post”}

Ok:1}

Will display 2 documents as array contains 2 documents
------array will get converted into documents

$sort operator
----to display addition of salary and bonus under the heading compensation in descending order of compensation and name
Db.emp.aggregate([
{$project:{“compensasion”:{ “$add”:{“$salary”,”$bonus”}}},name:1}’
{$sort:{“compensation”:-1,”name”:-1}}
])

$limit ---- takes number n and returns first n documents
$skip ----takes a number n as i/p and discards those many columns

-----skip is not efficient for large skip
--------as it finds all of the matching records that must be skipped

---

Date operator
$dayOfMonth,$dayOfWeek,$dayOfYear,$hour,$milisecond,$minute,$month,$second,$week,$year

---

{ "_id" : 1, "item" : "ABC1", sizes: ["S", "M", "L"] }
1.
db.inventory.aggregate([{ $unwind : "$sizes" }])

{ "_id" : 1, "item" : "ABC1", "sizes" : "S" }
{ "_id" : 1, "item" : "ABC1", "sizes" : "M" }
{ "_id" : 1, "item" : "ABC1", "sizes" : "L" }

2.

```
{ "_id" : 1, "item" : "ABC", "sizes": [ "S", "M", "L"] }
{ "_id" : 2, "item" : "EFG", "sizes" : [ ] }
{ "_id" : 3, "item" : "IJK", "sizes": "M" }
{ "_id" : 4, "item" : "LMN" }
{ "_id" : 5, "item" : "XYZ", "sizes" : null }
```

```
db.inventory.aggregate( [ { $unwind: "$sizes" } ] )
db.inventory.aggregate( [ { $unwind: { path: "$sizes" } } ] )
```

```
{ "_id" : 1, "item" : "ABC", "sizes" : "S" }
{ "_id" : 1, "item" : "ABC", "sizes" : "M" }
{ "_id" : 1, "item" : "ABC", "sizes" : "L" }
{ "_id" : 3, "item" : "IJK", "sizes" : "M" }
```

This excludes null values and empty array

```
db.inventory.aggregate( [ { $unwind: { path: "$sizes", includeArrayIndex:
"arrayIndex" } } ] )
```

The operation unwinds the sizes array and includes the array index of the array index in the new arrayIndex field. If the sizes field does not resolve to an array but is not missing, null, or an empty array, the arrayIndex field is null

```
{ "_id" : 1, "item" : "ABC", "sizes" : "S", "arrayIndex" : NumberLong(0) }
{ "_id" : 1, "item" : "ABC", "sizes" : "M", "arrayIndex" : NumberLong(1) }
{ "_id" : 1, "item" : "ABC", "sizes" : "L", "arrayIndex" : NumberLong(2) }
{ "_id" : 3, "item" : "IJK", "sizes" : "M", "arrayIndex" : null }
```

```
db.inventory.aggregate( [
   { $unwind: { path: "$sizes", preserveNullAndEmptyArrays: true } }
] )
```

```
{ "_id" : 1, "item" : "ABC", "sizes" : "S" }
{ "_id" : 1, "item" : "ABC", "sizes" : "M" }
{ "_id" : 1, "item" : "ABC", "sizes" : "L" }
{ "_id" : 2, "item" : "EFG" }
{ "_id" : 3, "item" : "IJK", "sizes" : "M" }
{ "_id" : 4, "item" : "LMN" }
{ "_id" : 5, "item" : "XYZ", "sizes" : null }
```

Similar to outer join

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

```
SELECT *, <output array field>
FROM collection
WHERE <output array field> IN (SELECT *
```

```
                                FROM <collection to join>
                                WHERE <foreignField>=
<collection.localField>);


db.orders.insert([
   { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
   { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
   { "_id" : 3  }
])



db.inventory.insert([
   { "_id" : 1, "sku" : "almonds", description: "product 1", "instock" :
120 },
   { "_id" : 2, "sku" : "bread", description: "product 2", "instock" : 80
},
   { "_id" : 3, "sku" : "cashews", description: "product 3", "instock" : 60
},
   { "_id" : 4, "sku" : "pecans", description: "product 4", "instock" : 70
},
   { "_id" : 5, "sku": null, description: "Incomplete" },
   { "_id" : 6 }
])
```

The following aggregation operation on the orders collection joins the documents from orders with the documents from the inventory collection using the fields item from the orders collection and the sku field from the inventory collection:

```
db.orders.aggregate([
   {
     $lookup:
       {
         from: "inventory",
         localField: "item",
         foreignField: "sku",
         as: "inventory_docs"
       }
  }
])



{
   "_id" : 1,
   "item" : "almonds",
   "price" : 12,
   "quantity" : 2,
   "inventory_docs" : [
      { "_id" : 1, "sku" : "almonds", "description" : "product 1",
"instock" : 120 }
   ]
}
{
   "_id" : 2,
   "item" : "pecans",
   "price" : 20,
   "quantity" : 1,
```

```
   "inventory_docs" : [
      { "_id" : 4, "sku" : "pecans", "description" : "product 4", "instock"
: 70 }
   ]
}
{
   "_id" : 3,
   "inventory_docs" : [
      { "_id" : 5, "sku" : null, "description" : "Incomplete" },
      { "_id" : 6 }
   ]
}
```

```
Orders
{ "_id" : 1, "item" : "MON1003", "price" : 350, "quantity" : 2, "specs" :
[ "27 inch", "Retina display", "1920x1080" ], "type" : "Monitor" }

Inventory
{ "_id" : 1, "sku" : "MON1003", "type" : "Monitor", "instock" : 120,
"size" : "27 inch", "resolution" : "1920x1080" }
{ "_id" : 2, "sku" : "MON1012", "type" : "Monitor", "instock" : 85,
"size" : "23 inch", "resolution" : "1280x800" }
{ "_id" : 3, "sku" : "MON1031", "type" : "Monitor", "instock" : 60,
"size" : "23 inch", "display_type" : "LED" }



db.orders.aggregate([
   {
      $unwind: "$specs"
   },
   {
      $lookup:
         {
            from: "inventory",
            localField: "specs",
            foreignField: "size",
            as: "inventory_docs"
         }
   },
   {
      $match: { "inventory_docs": { $ne: [] } }
   }
])


{
   "_id" : 1,
   "item" : "MON1003",
   "price" : 350,
   "quantity" : 2,
   "specs" : "27 inch",
   "type" : "Monitor",
   "inventory_docs" : [
      {
         "_id" : 1,
         "sku" : "MON1003",
         "type" : "Monitor",
         "instock" : 120,
         "size" : "27 inch",
```

```
         "resolution" : "1920x1080"
      }
   ]
}
```

Mapreduce function

```
{
     _id: ObjectId("50a8240b927d5d8b5891743c"),
     cust_id: "abc123",
     ord_date: new Date("Oct 04, 2012"),
     status: 'A',
     price: 25,
     items: [ { sku: "mmm", qty: 5, price: 2.5 },
              { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

```
var mapFunction1 = function() {
                       emit(this.cust_id, this.price);
                   };

var reduceFunction1 = function(keyCustId, valuesPrices) {
                          return Array.sum(valuesPrices);
                      };


db.orders.mapReduce(
                     mapFunction1,
                     reduceFunction1,
                     { out: "map_reduce_example" }
                   )
```

Calculate Order and Total Quantity with Average Quantity Per Item

```
var mapFunction2 = function() {
            for (var idx = 0; idx < this.items.length; idx++) {
               var key = this.items[idx].sku;
               var value = {
                        count: 1,
                        qty: this.items[idx].qty
                       };
               emit(key, value);
             }
           };


var reduceFunction2 = function(keySKU, countObjVals) {
            reducedVal = { count: 0, qty: 0 };
```

```
    for (var idx = 0; idx < countObjVals.length; idx++) {
       reducedVal.count += countObjVals[idx].count;
       reducedVal.qty += countObjVals[idx].qty;
    }

    return reducedVal;
  };



var finalizeFunction2 = function (key, reducedVal) {

    reducedVal.avg = reducedVal.qty/reducedVal.count;

    return reducedVal;

  };



db.orders.mapReduce( mapFunction2,
    reduceFunction2,
    {
     out: { merge: "map_reduce_example" },
     query: { ord_date:
              { $gt: new Date('01/01/2012') }
            },
     finalize: finalizeFunction2
    }
  )
```

This operation uses the `query` field to select only those documents with `ord_date` greater than `new Date(01/01/2012)`. Then it output the results to a collection `map_reduce_example`. If the `map_reduce_example` collection already exists, the operation will merge the existing contents with the results of this map-reduce operation.