Questions:

CRUd - create, read , update, delete

Update document

update, updateMany, updateOne

```
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.
`$push`	Adds an item to an array.
`$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}

   )

------update price to 400 and rating to 5 for
All movies whose name contains a at 2 nd
position

Db.movie.update({name:/^.a/},
{$set:{price:400,rating:5}},
{multi:true})



----update rating of kahani movie
to 5
>db.movie.update({name:'kahani'},{$set:{ratin
g:5}},{multi:true})
```

---- update price of movie sholey to 300

- Db.movie.update({name:’Sholey’},{$set:{price:300}},{multi:true,upsert:true})

3. Users

- {_id:1,
- status:’D’,cancellation:{date: ISODate(2020-09-27),reason:’user requested’} }

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

--------to remove the rating key

```
>db.movie.update({name:”kahani”},{$unset:{rating:””}},{multi:true,upsert:true})
```

------ increase the price by 100 for kahani movie
>db.movie.update({name:’kahani’},{$inc:{price:100}},{multi:true})

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" },
     $currentDate: { lastModified: true }
   }
)
-----update rating of all movies to 5 if the
price > 300
```
>db.movie.update({**price:{$gt:300}**},{$set:{rating:5}},{upsert:true,multi: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 } ] }
)
```

```
Increase the ticket_no by 20 for all movies
db.movie.update({},{$max:{ticket_no:500}},{mult
i:true})                              200
```

Using $min
Use $min to Compare Numbers
Consider the following document in the
collection scores:

```
{ _id: 1, highScore: 800, lowScore: 200 }
```

The lowScore for the document currently has the value 200. The following operation uses $min to compare 200 to the specified value 150 and updates the value of lowScore to 150 since 150 is less than 200:

```
db.scores.update( { _id: 1 }, { $min: { 
lowScore: 150 } } )
```

The scores collection now contains the following modified document:

```
{ _id: 1, highScore: 800, lowScore: 150 }
```

The next operation has no effect since the current value of the field lowScore, i.e 150, is less than 250:

```
db.scores.update( { _id: 1 }, { $min: { 
lowScore: 250 } } )
```

The document remains unchanged in the scores collection:

```
{ _id: 1, highScore: 800, lowScore: 150 }
```

To remove value of field

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

To delete from all matching documents

db.movie.update({name:'padmavat'},{$unset:{'rating':""}},{multi:True})

---------To modify name of the column use $rename
{id:123,name:’jhfjhdfkjh’,addr:’Aundh}

db.student.update({},{$rename:{addr:’address’}})

----------To multiply by a number
Db.movie.update({},{$mul:{rating:2}})

To add data in the array

db.movie.update({name:'mission mangal'},{$push:{actor:"Tapsee"}})

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

---

student :
{_id:1,name:”revati”,hobbies:[“reading”,”swimming”]}
db.student.update({name:”revati”},{$push:{hobbies:{$each:[”drawing”],$position:0}}},{multi:true})

db.student.update({name:”revati”},{$push:{hobbies:{$each:[“d
rawing”,”riding”,”reading novels”],$position:2}}},{multi:true})

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

-------write query to add grade (“B”,21-06-2018,89)object in
grades
array for all documents for cuisine is America or Chinese

>db.restaurants.update({“cuisine”:{$in:[“America”,”Chinese”]}},
{ $push:{grades:”{grade:”A”,score:”89”,date:ISODate(“2018-
06-21”)}”}},{multi:true})

---------add new contact for Rajan mobile and number is 33333
Db.friends.update({name:’Rajan’},{$push:{contact:’{type:’mob’,
num:33333}’}},{multi:true,upsert:true})

>db.restarants.update({“cuisine”:{$in:[“America”,”Chinese”]},{
$push:{grades:{$each:”[
{grade:”A”,score:”89”,date:ISODate(“2018-06-21”)},
{grade:”A+”,score:”99”,date:ISODate(“2018-08-
21”)}]”,$position:2} }} },{multi:true,upsert:true})

db.movie.update({name:”kahani”},{$push:{ actors:{ $each:[“aaaa”,”bbbb”],$position:0 }}},{multi:true})

<u>$ operator will update only the first matched value.</u>

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

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

$ indicates update the first matched record

```
db.students.update(
   { _id: 4, "grades.grade": 85 },
   { $set: { "grades.$.std" : 6 } }
)
```

----- to increase all std value by 2

Db.mygrade.update({},{$inc:{‘grades.std’:2}}

Coordinate:[34.5555,35.666]

{coordinate.0:12.0000}

---- increase the salary by 10000 for all employees.

```
>db.employee.update({},{$inc:{sal:10000}})

>db.movie.update({name:"padmavat"},{$inc:{rating:-2}})
```

$[] – 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 values in grade array by 10

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

------write a query to increase only 85 values
by 5
for all documents
>db.students.update({grades:85},{$inc:{"grades.
$":5}},{multi:true})



db.students.update(
   { grades:85},
   { $inc: { "grades.$": 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 std

```
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 deleting last element

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

---- **delete last value of actors array for movie kahani**

```
>db.movie.update({name:"Kahani"},{$pop:{actor:1
}},{multi:true})
```

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 }
)
```

```
>db.movie.update({name:/^s/},{$pull:{actor:["vi
dya balan"]}},{multi:true})
```

------ **write a query to delete last object of grades array**
from all documents

```
>db.restaurants.update({},{$pop:{grades:1},{mul
ti:true})
```

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

or
db.movie.createIndex({rating:-1})

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

db.emp.ensureIndex({sal:1})

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

To view indexes
db.movie.getIndexes()

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

-----delete all documents from employee collection whose
salary is < 10000

db.emp.remove({sal:{$lt:10000}})

Nested Queries

```
SELECT * FROM books WHERE id IN (
  SELECT bookId FROM likes WHERE userId=100
)
```

------list all movies with rating same as kahani;

```
Select *
From movie
Where rating=(select rating from movie where name='kahani')
```

```
Var
ratings=db.movie.find({name:’kahani’}).map(func
tion(mv){return mv.rating;});

Db.movie.find({rating:{$in: ratings }})
```

```
var bookIds =
db.likes.find({userId:100}).map(function(like)
{
  return like.bookId;
});
var books = db.books.find({_id:{$in:bookIds}});

var
name=db.movie.find({name:’kahani’}).map(functio
n(mv){
  return mv.rating;
})
db.movie.find({rating:name})
```

Aggregate function

$project	to keep only required keys, or find derived columns
$group	To use all aggregate function ($sum,$min,$max,$avg)
$match	Keep documents which satisfies the condition
$unwind	It unwinds the array

	If array contains 3 objects it will convert one document onto 3 separate Documents by unwinding it
$sort	Sorts the data
$skip	Skips the given records
$limit	Restrict number of records

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

----------to display count of movies based on rating only if count is > 3 and display it in descending order of rating

```
db.movie.aggregate([
{$project:{rating:1}},
{$group:{_id:”$rating”,count:{$sum:1}}},
{$match:{count:{$gt:3}}}
{$sort:{_id:-1}}
])
```

-----------to display name,rating, price and addition of price and rating
For all movies with price > 200,arrange it in descending order
Of addition and display 2 nd movie

```
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()
```

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]
```

Price*1.10

```
$multiply:[“$price”,1.10]
Db.movie.aggregate([{$project:{name:1,rating:1,price:1,”increa
sed price”: {$multiply:[“$price”,1.10]}
}}])
```

------display increased rating by 2

```
Db.movie.aggregate([{$project:{name:1,rating:1,nrating:{$add:[
“$rating”,2]}}}}])
```

Price+rating-ticket_no

```
$add:[‘$price’,’$rating’]
```

```
Addition:{$subtract:[ {$add:[‘$price’,’$rating’]},’$ticket_no]}
Db.movie.aggregate([{$project:{name:1, Addition:{$subtract:[
{$add:[‘$price’,’$rating’]},’$ticket_no]}
}}])
```

----------String related function

$substr : [expr,start offset,number of character to return]

```
Db.movie.aggregate([
{$project:{name:1,subname:{$substr:['$name',0,3]}
}])
```

$concat: [expr 1[,expr 2,expr 3,expr 4, ……….,exprn]
$toLower : expr
$toUpper : expr

1. To display all names in lowercase for all movies with rating > 3

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

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


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

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

}}}])
```

-----------------date related function

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

---

2. To display number of years of experience

```
Db.emp.aggregate([{$project:{name:1,”year_of_joining”:{$year
:’$hiredate’  }}])
{$year:’$hierdate’}
{$year:’new Date()’}
Experience:{$subtract:[ {$year:new Date()}
, {$year:’$hierdate’}
]}
```

Select rating,name,price,price+rating addition

```
{$project:{empno:1,ename:1,updatedsal:{$divide[$multiply:[‘$s
al’,1.10,’$percentage’]}}},10]}
{$project:{rating:1,name:1,price:1,calculate:{}}

A={$multiply:[‘$price’,’$rating’]}
{$divide:[‘$price’,’$rating’]}
(price*rating) /3        {$divide:[{
$multiply:[‘$price’,’$rating]},3]}

Db.movie.aggregate({$project:{name:1,price:1,rating:1,newpric
e: {$divide:[{ $multiply:[‘$price’,’$rating]},3]}}})

$divide:[$multiply:[‘$price’,’$rating’],3]

((price+ticketno)-rating)/price
```

$divide:[$ subtract:[$add:[‘$price’,$ticket_no’] ,’$rating’],’$price’]

-------divide price by 2 and check whether reminder is 0
If price %2==0

{price:{$mod:[2,0]}}

Addition:$add:[{$divide:[
{$multiply:[‘$price’,’$rating’]},2]},’$rating’]

db.employee.aggregate([{$project:{“experience”:{$subtract:{[
{$year:’new Date()’},{$year:’$hiredate’}

]}}}}])

{$project:{exp_date:{$add:[{$year:’$mfg_date’},3]}}}

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 .gmail.com
Abc.pqr

```
$concat:['$fname',".",'$lname','@gmail.com']
```

```
db.employee.aggregate([{$project:{
"email":{$concat:[{$substr:['$fname',0,3]},".","$lastname","@g
mail.com"]} }}])
```

----

we may us Logical expressions
{price:{$gt:23}}
$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
a>b?a:b
2 different syntax for $cond

{ $cond: { if: <boolean-expression>, then: <true-case>, else: <false-case-> } }

$cond:[booleanExpr,truetxprs,falseexprs]

e.g

note : the way we check condition---- syntax is different

$exists doenot work in $cond in that case use $ifnull

syntax 1-------------

-----display rating if > 2 otherwise display 'not found'

Rating>2>rating:"notfound"

{$cond:{if:{$gt:["$rating",2]},then:"$rating",else:"not found"}}

db.movie.aggregate([{$project:{name: 1,rating:1,discount:{$cond: [{ $gte: ["$rating", 2] }, '$rating', 'not found']}}}])

```
db.orders.aggregate( [
  { $project: { status: {
    $cond: { if: { $gt: ["$feedback", null] },
    then: 'finished', else: {
      $cond: { if: { $gt: ["$received", null] },
      then: 'received', else: {
        $cond: { if: { $gt: ["$shipped", null] },
        then: 'shipped', else: {
          $cond: { if: { $gt: ["$payment", null] },
            then: 'payment received', else: 'created' }
        } }
      } }
    } }
  } } },
  { $match: { } }
] )
```

syntax 2--------------------------

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

```
Db.movie.aggregate([{$project:{name:1,rating:1,status:{$cond:
{if:{$gt:[‘$rating’:4]} ,then:’good’, else:’ok’ }}}}])
```

$arrayElemAt to retrieve value at index position

```
> db.restaurants.aggregate([
{$project:{name:1,_id:0,odate:{$arrayElemAt:['$grades.date',0]
}}},
{$project:{name:1,oyear:{$year:'$odate'}}},
{$match:{oyear:{$gt:2013,$lte:2015}}},
{$group:{_id:null,count:{$sum:1}}}])
output: {"_id" : null, "count" : 24260 }
```

```
> db.restaurants.aggregate([
{$project:{name:1,_id:0,odate:{$arrayElemAt:['$grades.date',0]
}}},
{$project:{name:1,oyear:{$year:'$odate'}}},
{$match:{oyear:{$gt:2013,$lte:2015}}},
{$group:{_id:'$oyear',count:{$sum:1}}}])
```

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

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

```
{name:”jashha”,rating:null}
{$project:{name:1,rating:{$ifNull:[‘$rating’,1]}}}
{name: :”jashha”,rating:1}


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

$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’}
,”average”:{‘$avg’,’$marks’}  }}}])

db.movie.aggregate({$group:{_id:’$rating’,min_price:{$min:’$price’}}}})

db.movie.aggregate([{$group:{_id:”$rating”,minp:{$min:”$price”},maxp:{$max:”$price”},avg:{$avg:”$price”},cnt:{$sum:1}}}])
```

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}}

])
```

---------find sum of prices of all movies whose rating is same

```
Db.movie.aggregate([{
$group :{_id:”$rating”,”sum of
prices”:{$sum:’$price’},minprice:{$min:’$price’}}

}])
```

$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”}

]

{_id:Objectid(--------),author:”a”,post:”hello” {author :”abc” ,
date :ISODATE(2018-04-30T17:52:04.148z),text:”nicepost”}}
{_id:Objectid(--------),author:”a”,post:”hello”,      {author :”pqr”
,    date :ISODATE(2018-04-30T17:52:04.148z),text:”goodone
post”}
```

```
>db.blog.post.aggregate([{
      $unwind:”$comment”
}])
{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 comments
------array will get converted into documents

--------count how many actor in each movie

```
db.movie.aggregate([{$match:{name:"kahani"}},
{$unwind:"$actor"},
{$group:{_id:"$name",count:{$sum:1}}}
])
```

------Count each actor has acted in how many movies.

$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

---

```
{ "_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 } }
] )
```

```
db.blog.post.aggregate({$unwind:{path:'$comment
s',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 }
```
