Release: 20230414

Topics we cover

Array Updates

Expressive Updates

Upsert Operations

FindOneAndUpdate

Updating Arrays

There are different ways to update specific members of arrays

Example (kept very minimalist for clarity):

- Database with information about the number of hours a person (Tom) works
- The array "hrs" has 5 members representing the days of the week
- Each array member contains the number of hours to work on that day

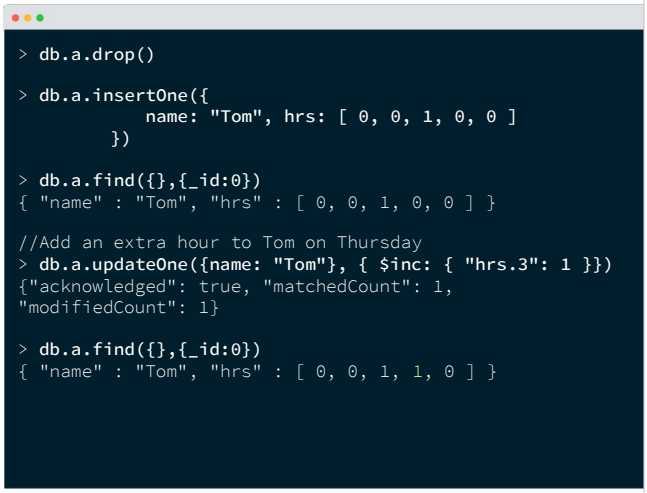
In the next set of slides we look at different ways to update specific members of arrays.
Imagine our database contains information about the number of hours a person "Tom" will work so the array "hrs" of 5 members represents days of the week, each is how many hours they need to work that day.
This example is kept very minimalist for clarity.

Arrays - Update Specific Element

Update a **specific array element** using a **numeric index**

The index starts at 0

Arrays grow as required



```
> db.a.drop()

> db.a.insertOne({
            name: "Tom", hrs: [ 0, 0, 1, 0, 0 ]
          })

> db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 0, 0, 1, 0, 0 ] }

//Add an extra hour to Tom on Thursday
> db.a.updateOne({name: "Tom"}, { $inc: { "hrs.3": 1 }})
{"acknowledged": true, "matchedCount": 1,
"modifiedCount": 1}

> db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 0, 0, 1, 1, 0 ] }
```

- Array elements can be updated by using numeric index e.g.
  - db.a.updateOne({name:"Tom"},{ $inc : { "hrs.3" : 1 }})
- Arrays are zero indexed in MongoDB
- If we explicitly set a value which is higher than the length of the array, the array will be extended to the required length and all the new elements except the one we are setting will be set to null.

Arrays - Update Matched Element

Use **$ as a placeholder** for the **first position matched** by a query

In this case, finds the first element in the array with a value less than 1 and increments the value in 2

```
> db.a.drop()

//Tom works one hour on wednesday
> db.a.insertOne({name: "Tom", hrs: [ 0, 0, 1, 0, 0 ] })

//Find the first day Tom has no hours,add two to that day
> db.a.updateOne(
      {name:"Tom",hrs:{$lt:1}},
      { $inc: { "hrs.$": 2 }}
  )
{ "acknowledged" : true, "matchedCount" : 1,
"modifiedCount" : 1 }

> db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 2, 0, 1, 0, 0 ] }
```

- $ can be used as a placeholder for first position matched by query e.g.
  - db.a.updateOne({name:"Tom",hrs:{$lt:1}},{ $inc : { "hrs.$" : 2 }})

Arrays - Update All Elements

Update **every element of an array with $[]**

In this example, 2 extra hours are added to every entry for every document that matches the query

```
> db.a.drop()

> db.a.insertOne({name: "Tom", hrs: [ 0, 0, 1, 0, 0 ] } )

//Give Tom two more hours work every day
> db.a.updateOne({name:"Tom"},{$inc : { "hrs.$[]" : 2 }})

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

> db.a.find({},{_id:0})

{ "name" : "Tom", "hrs" : [ 2, 2, 3, 2, 2 ] }
```

- $[ ] changes all array elements
- Was added in MDB 3.6
- Not widely used

Arrays - Update All Matched Elements

Query to find documents is not used to decide what elements to change

Separate arrayFilter(s) apply update to matching array elements

This example adds 2 to everything less than 1 hr

```
> db.a.drop()

> db.a.insertOne({name: "Tom", hrs: [ 0, 0, 1, 0, 0 ] } )

//Find a week with a day with no hours (query)
//For each day Tom has no hours and add 2 to those days
//arrayFilter assumes there might be multiple records
// so we cannot use JUST arrayfilters.

> db.a.updateOne({name:"Tom",hrs:{$lt:1}},
                 {$inc : { "hrs.$[nohrs]" : 2 }},
                 {"arrayFilters": [
                       {"nohrs":{"$lt":1}}]}
                )

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


> db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 2, 2, 1, 2, 2 ] }
```

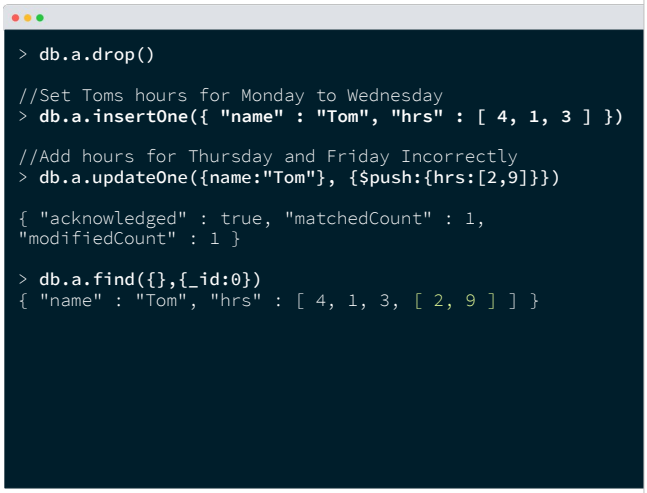
- Array filters allow us to select specific members of an array to modify by defining addition match criteria/queries.
- They are separate from the find() part of the query so we can find() with one query but then decide how to update the array based on different criteria
- If we do include some of the filter criteria in the find() part as we have done here - then we can avoid the computation of even trying to update records where the filter won't match.
- We can define multiple named array filters in our update - we called this one **nohrs**, nohrs is called an identifier
- In the filter the identifier is evaluated against each array element and those that match are updated.
- Here we say If the element is less than one then update it
- If the array elements are objects - we can dereference them in the filter with dots - e.g in an array of items on a purchase receipt { "arrayFilters" : [{ "item.type" : "socks, "item.price" : {$gt:5} }] }

Arrays and $each

Use $push to add an element at the end of an array

Passing an array of elements with push adds them as one array element inside the original array

To add the members individually use $each

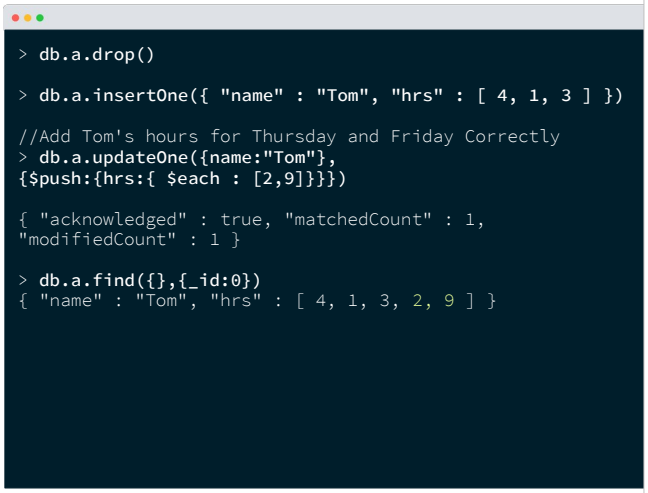


In the example, $push does not give the desired effect - the array itself is added to the existing array, not the individual elements

Arrays and $each

Use **$each** to push the elements separately

Also works with **$addToSet**

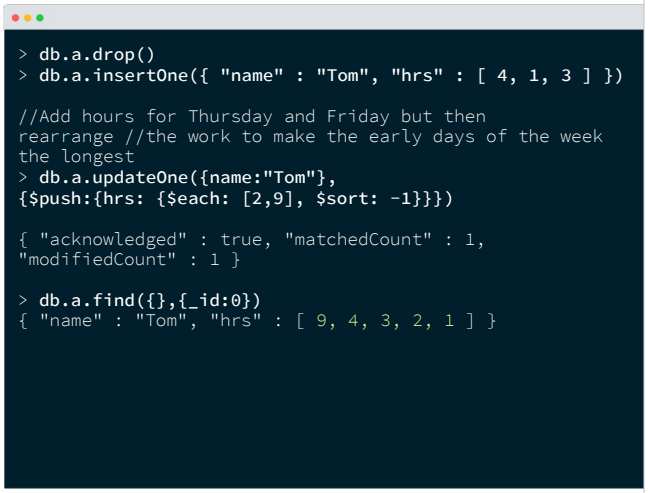


- Using $each adds the element separately to the array
- $addToSet would also achieve this

Arrays and $each

While pushing array elements, it is possible to sort them

Replicating **$push** and **$sort** manually on the client will likely not get the desired results



Elements can also be sorted as they are pushed but doing this client side will likely get a undesired effects.

Example,

If following steps are done simultaneously by different users, we might end up overwriting data:

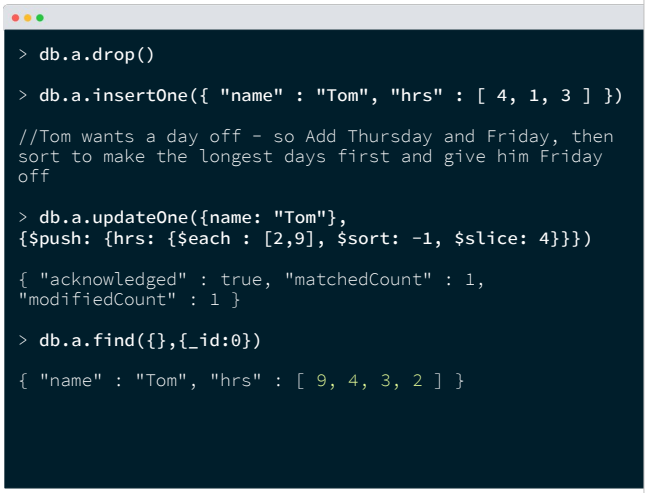
```
User1
 doc = db.a.findOne({name: "Tom"})
 doc.hrs.push(2)
 doc.hrs.push(9)
 doc.hrs.sort()
 db.a.updateOne({name: "Tom"}, {$set: doc})


User2
 doc = db.a.findOne({name: "Tom"})
 doc.hrs.push(3)
 doc.hrs.push(5)
 doc.hrs.sort()
 db.a.updateOne({name: "Tom"}, {$set: doc})
```

Arrays and $each

Sort and keep the Top (or bottom) N

This is an example of a design pattern

Used for High/Low lists - high scores, top 10 temperatures etc.



```
> db.a.drop()

> db.a.insertOne({ "name" : "Tom", "hrs" : [ 4, 1, 3 ] })

//Tom wants a day off - so Add Thursday and Friday, then
sort to make the longest days first and give him Friday
off

> db.a.updateOne({name: "Tom"},
{$push: {hrs: {$each : [2,9], $sort: -1, $slice: 4}}})

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

> db.a.find({},{_id:0})

{ "name" : "Tom", "hrs" : [ 9, 4, 3, 2 ] }
```

$slice allows us to keep the top (or bottom) N number of elements

Expressive Updates

Expressive updates use aggregation expressions to define the fields to **$set**

Updates can be based on values in the document or calculated conditions

Example: Add an area field to all our rectangular records. If we add a field like this we can build an index on it.

```
db.shapes.updateMany({
     shapename: {
          $in:["rectangle","square"]
     }
},
[{
     $set: {
          area: {
               $multiply: ["$w","$h"]
          }
     }
}])
```

- Updates can also be performed using an aggregation pipelines expression - we cover them later in the course.
- This can be used to update a document based on other values in the document e.g. if a=1 then add 2 to b else add c to b
- The update operation is an array containing a pipeline that outputs the new changes
- The update mutation is an array of pipeline stages, hence the square brackets.

Exercise: Array updates

Use the **sample_training** database and update the **grades** collection:

1. If anyone got >90% for any homework, reduce their score by 10
2. Add a new field containing their mean score in each class
3. Drop each student's worst score. Use two updates to do this

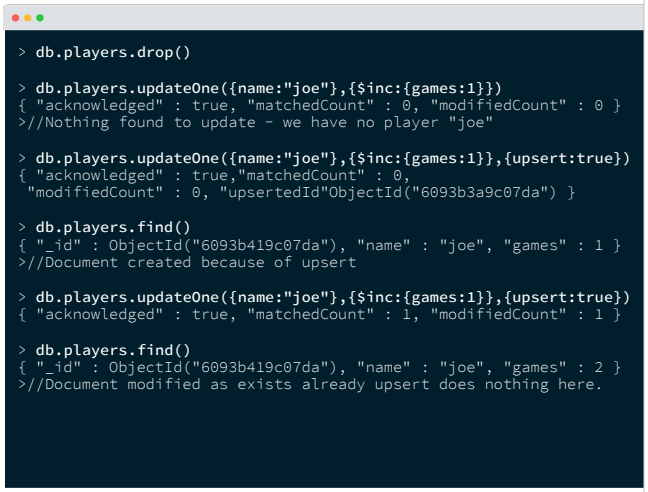
Hint: Use the **$avg** operator for calculating the mean score.
How do you get the Students worst score to a known place in the array?

Upsert

Most MongoDB operations that update also allow the flag `"upsert: true"`

Upsert **inserts** a new document if none are found to update

Values in both the **Query and Update** are used to create the new record



```
> db.players.drop()

> db.players.updateOne({name:"joe"},{$inc:{games:1}})
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0 }
>//Nothing found to update - we have no player "joe"

> db.players.updateOne({name:"joe"},{$inc:{games:1}},{upsert:true})
{ "acknowledged" : true,"matchedCount" : 0,
 "modifiedCount" : 0, "upsertedId"ObjectId("6093b3a9c07da") }

> db.players.find()
{ "_id" : ObjectId("6093b419c07da"), "name" : "joe", "games" : 1 }
>//Document created because of upsert

> db.players.updateOne({name:"joe"},{$inc:{games:1}},{upsert:true})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.players.find()
{ "_id" : ObjectId("6093b419c07da"), "name" : "joe", "games" : 2 }
>//Document modified as exists already upsert does nothing here.
```

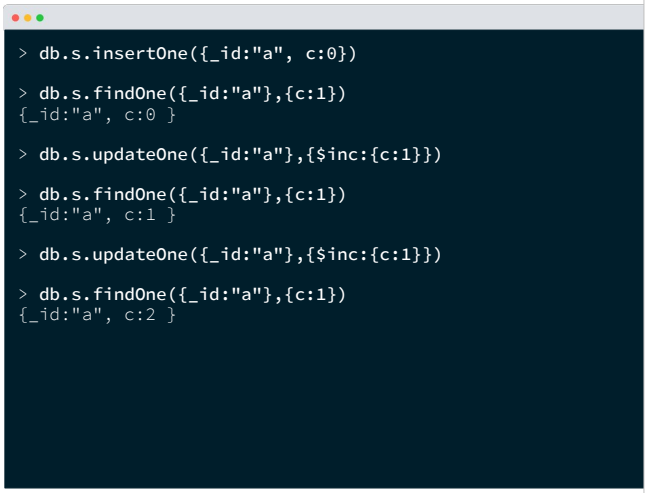
- Upserts are a useful feature that will be covered more later
- If a document isn't found to update then one that matches your criteria is created
- Passed as an option to updateOne() and updateMany() and other update operations
- Not atomic between find() and insert() - it's possible for two simultaneously run to both do an insert.
  - If this is an issue then supplying _id using $setOnInsert update modifier helps
  - unique constraints help too.
- Useful in a few design patterns and simplifies code versus a call to update then to insert.

findOneAndUpdate()

To understand **findOneAndUpdate()**, we must first understand **updateOne()**

updateOne() finds and changes document atomicly doesn't return the updated document

Imagine getting the next one-up number from a sequence

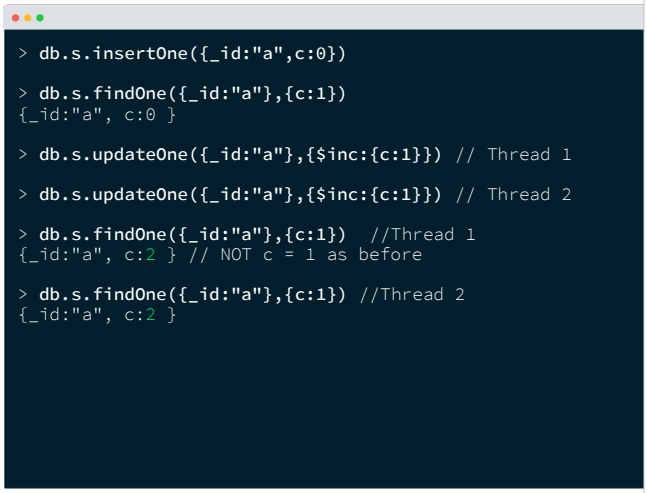


If we wanted to update a value and fetch it, to create a sequence say for a "Customer number" or "Invoice NUmber" we could do it like this - and in a single thread this is fine. If there is no projection applied in the findOne() commands, the result would still be the same as there is only one field apart from the _id field.

findOneAndUpdate()

Imagine 2 parallel processes running **updateOne()** and **findOne()** on the same document

What would happened if they interleave the calls?

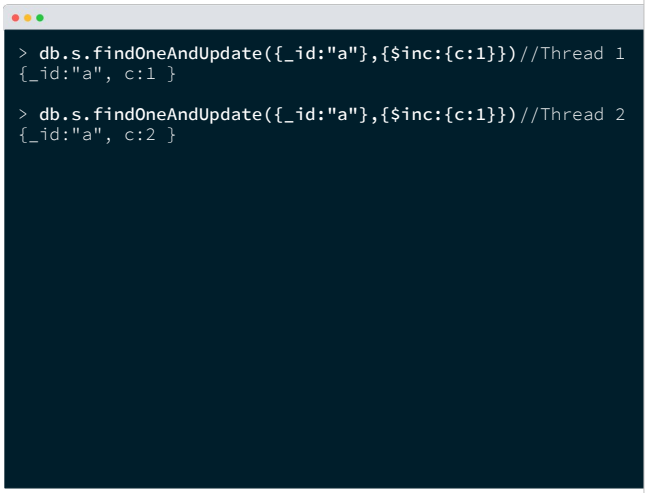


If we extend this to multiple threads then we can have a race condition where we update twice then fetch twice.

It would be fine if we don't provide the projection in findOne() commands as there is only one field other than the _id. It would give us the same result.

findOneAndUpdate()

findOneAndUpdate() makes find, modify, and fetch atomic operations and hence fixes the problem



findOneAndUpdate() solves issues from previous slide

With findOneAndUpdate the single, atomic command finds, modifies and returns the document.

By default it will return the document BEFORE the update. Use the {returnNewDocument: true} parameter to return the document AFTER the update.

You can apply things like sorting and projection to control what it finds and modifies.

It only affects one document and also will work with upsert.

Recap

There are multiple ways to update specific array elements

Expressive Updates can compute new values for fields

Upsert updates or creates a record

FindOneAndUpdate atomically returns the version of the updated document

Quiz Time!

#1. Which of the following queries will return the document: { scores: [1,2,3] }?

A

```
find ({scores: 3})
```

B

```
find ({
  scores: [3,2,1]
})
```

C

```
find ({
  scores: {$all: [1,7]}
})
```

D

```
find ({
 scores: {$in: [1,3,7]}
})
```

E

```
find ({
 $expr: {
   $eq:[{$sum:"$scores"}, 6]
 }})
```

Answer in the next slide.

#1. Which of the following queries will return the document: { scores: [1,2,3] }?

A

```
find ({scores: 3})
```

B

```
find ({
  scores: [3,2,1]
})
```

C

```
find ({
  scores: {$all: [1,7]}
})
```

D

```
find ({
 scores: {$in: [1,3,7]}
})
```

E

```
find ({
 $expr: {
  $eq:[{$sum:"$scores"}, 6]
 }})
```

Answer: A, D & E
The scores array field contains a 3
The scores array field is not exactly like the array we are looking for in the B option as the number are in different order
The scores array field contains 1 but does not contain a 7 ($all forces that all need to be in the array)
The scores array field contains 1 and 3 so it matches the $in condition
The expression ($expr) matches the document: the $sum of all the values in the array is $eq (equal) to the number 6

#2. Why is findOneAndUpdate() on a server better than find() and update() on an application?

A	Stops contention between writers	B	Prevents one client from overwriting another's changes	C	Reduces network traffic
D	Excess locking can be prevented	E	Multiple retries from writers can be avoided		

Answer in the next slide.

#2. Why is findOneAndUpdate() on a server better than find() and update() on an application?

A	Stops contention between writers	B	Prevents one client overwriting another's changes	C	Reduces network traffic
D	Excess locking can be prevented	E	Multiple retries from writers can be avoided		

Answer: B & C

#3. Consider the following operations executed on an empty collection:

```
updateOne({_id: 700}, {$push: {sensor: {$each: [4,0,2], $sort: -1}}},{upsert: true})
updateOne({_id:700 }, { $push: { sensor: { $each: [0, 7, 3, 4, 5], $slice: 5}}})
```

Select the number of array elements stored in the sensor array field:

A. There are 0 elements, the collection is empty

B. There are 3 elements in total in the array

C. There are 5 elements due to the slice operator

D. There are 8 elements in total in the array

Answer in the next slide.

#3. Consider the following operations executed on an empty collection:

```
updateOne({_id: 700}, {$push: {sensor: {$each: [4,0,2], $sort: -1}}},{upsert: true})
updateOne({_id:700 }, { $push: { sensor: { $each: [0, 7, 3, 4, 5], $slice: 5}}})
```

Select the number of array elements stored in the sensor array field:

A. There are 0 elements, the collection is empty

B. There are 3 elements in total in the array

C. There are 5 elements due to the slice operator

D. There are 8 elements in total in the array

Answer: D
The $slice operator limits the number of array elements to 5 in this case.

Exercise Answers

Answers - Exercise: Array updates I

In the grades collection, if anyone got >90% for any homework, reduce their score by 10. Note we need to use $elemMatch

```
db.grades.updateMany({

scores:{$elemMatch:{type:"homework", score:{$gt:90}}}},

{$inc: {"scores.$[filter].score":-10}},

{ arrayFilters: [{"filter.type":"homework","filter.score": {$gt:90}}]})
```

Answers - Exercise - Array updates II

In the grades collection add a new field containing their the mean score in each class.

```
db.grades.updateMany( {},
[{ $set: { average :  { $avg: "$scores.score" }}}]
)
```

Answers - Exercise - Array updates III

Drop each student's worst score

Use multiple updates to do this

```
db.grades.updateMany({},
   {
     $push: {
       scores: {
          $each: [],
          $sort: { score: 1 }
       }
     }
   }
)
db.grades.updateMany({}, {$pop: { scores: -1 }})
```