

MongoDB教育訓練-05

Arrays - Update Matched Element

We can use **\$** as a placeholder for the first position matched by a query

In this case find array with a value less than 1

Ignore the fact we are also querying by primary key

```
> 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 and 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 ] }
```

更新所有陣列元素

Arrays - Update All Elements

We can update every element of an array by specifying it with **\$[]**

In this example we are adding 2 to every entry in hours for every document that matches our 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 ] }
```

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 says "add 2 to everything < 1 in hrs"

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

//Find a week where tom has a day with no hours (query)
// for each day Tom has no hours and add 2 to those days
// (arrayFilter) - assume there might be multiple records for Tom
// 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 ] }
```

Arrays and \$each

When we use \$push to add to an array that value is added as the last element.

In this case we now have an array inside an array.

However, we wanted to add just the members.

```
> db.a.drop()

//Set Toms hours for Monday to Wednesday
> db.a.insertOne({ "name" : "Tom", "hrs" : [ 4, 1, 3 ] })

//Add hours for Thursday and Friday Incorrectly
> db.a.updateOne({name:"Tom"}, {$push:{hrs:[2,9]}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 4, 1, 3, [ 2, 9 ] ] }
```

Arrays and \$each

Using \$each pushes the elements separately

Also works with \$addToSet

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

//Add Tom's hours for Thursday and Friday Correctly
> db.a.updateOne({name:"Tom"}, {$push:{hrs:{$each : [2,9]}}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

> db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 4, 1, 3, 2, 9 ] }
```

Arrays and \$each

As we push them in we can also sort the array

If we try to do this client side we will likely get race conditions

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

//Add hours for Thursday and Friday but then rearrange the work to
//make the early days of the week the longest
> db.a.updateOne({name:"Tom"},
{$push:{hrs: {$each: [2,9], $sort: -1}}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

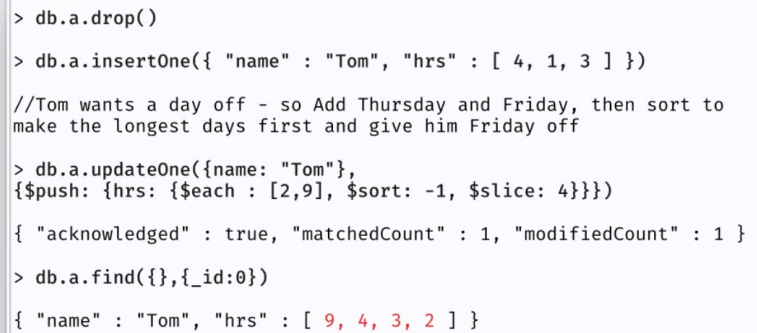
>db.a.find({},{_id:0})
{ "name" : "Tom", "hrs" : [ 9, 4, 3, 2, 1 ] }
```

Arrays and \$each

We can also 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 ] }
```

子查詢更新

Expressive Updates

Expressive updates use aggregation pipeline expressions to define what fields to \$set and their new values.

Updates can be based on other values in the document or conditions calculated from them.

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

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

Exercise - Array updates

Again, use the **sample_training** database in the Mongo shell.

In the **grades** collection - if anyone got >90% for any homework, reduce their score by 10.

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

Drop each student's worst score in the **grades** collection. You need to use two updates to do this.

```
Atlas atlas-vnfeuv-shard-0 [primary] sample_training> db.grades.findOne()
{
  _id: ObjectId("56d5f7eb604eb380b0d8d8ce"),
  student_id: 0,
  scores: [
    { type: 'exam', score: 78.40446309504266 },
    { type: 'quiz', score: 73.36224783231339 },
    { type: 'homework', score: 46.980982486720535 },
    { type: 'homework', score: 76.67556138656222 }
  ],
  class_id: 339
}
```

```

1 db.grades.explain("executionStats").updateMany({
2     scores: {
3         $elemMatch: {
4             type: "homework",
5             score: {
6                 $gt: 90
7             }
8         }
9     }
10 }, {
11     $inc: {
12         "scores.$[filter].score": -10
13     }
14 }, {
15     arrayFilters: [{
16         "filter.type": "homework",
17         "filter.score": {
18             $gt: 90
19         }
20     }]
21 })
22

```

```

db.grades.updateMany({
    scores: {
        $elemMatch: {
            type: "homework",
            score: {
                $gt: 90
            }
        }
    }
}, {
    $inc: {
        "scores.$[filter].score": -10
    }
}, {
    arrayFilters: [{
        "filter.type": "homework",
        "filter.score": {
            $gt: 90
        }
    }]
})

```



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

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

Update & Insert

資料存在則更新，否則新增

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.
```

返回更新前後的值(updateOne無法返回)

需要返回值的時候使用，不然就只用updateOne

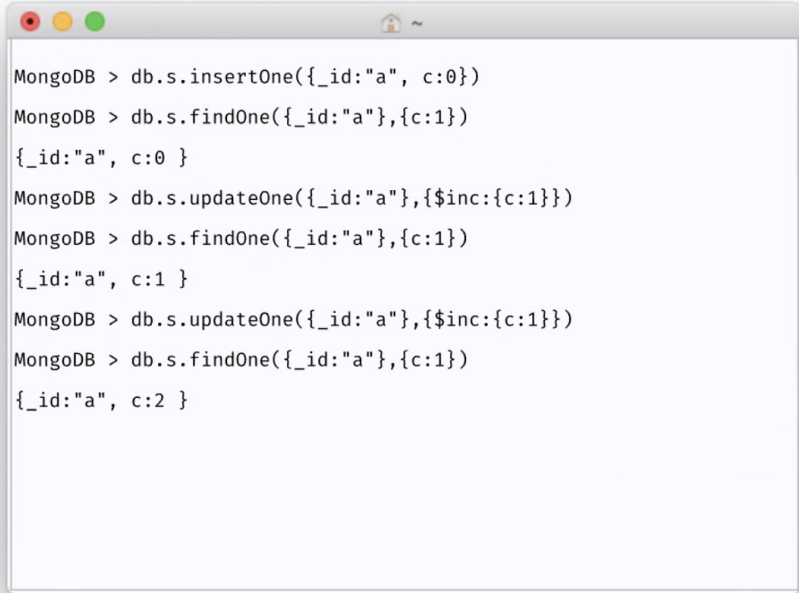
findOneAndUpdate()

To understand `findOneAndUpdate()`, we must first understand `updateOne()`.

In `updateOne` the find and the change are atomic

However `updateOne` doesn't return the updated document.

Imagine getting the next one-up number from a sequence



```
MongoDB > db.s.insertOne({_id:"a", c:0})
MongoDB > db.s.findOne({_id:"a"},{c:1})
{ _id: "a", c: 0 }
MongoDB > db.s.updateOne({_id:"a"},{$inc:{c:1}})
MongoDB > db.s.findOne({_id:"a"},{c:1})
{ _id: "a", c: 1 }
MongoDB > db.s.updateOne({_id:"a"},{$inc:{c:1}})
MongoDB > db.s.findOne({_id:"a"},{c:1})
{ _id: "a", c: 2 }
```

🕒 修訂版本 #3

★ 由 treeman 建立於 16 🕒 2021 23:38:54

✍ 由 treeman 更新於 5 🕒 2023 10:14:59