

Introduction Updating with Aggregation Pipelines and Arrays to MongoDB

Objectives

By the end of this session, you should be able to:

- ☐ Introduction
- ☐ Updating Array Fields
- ☐ Data Aggregation
- ☐ Introduction
- ☐ aggregate Is the New find
- ☐ Manipulating Data
- ☐ Working with Large Datasets
- ☐ Getting the Most from Your Aggregations

Updating with an Aggregation Pipeline

- ❑ A pipeline is composed of multiple update expressions called stages.
- ❑ When an update operation containing multiple stages of update expressions is executed, each of the matched documents is processed and transformed through each stage sequentially.
- ❑ The output of the first stage is input for the next stage, until the last stage in the pipeline produces the final output. Apart from writing multi-stage update expressions, pipeline support also allows the use of field references in the update expressions.
- ❑ The following code snippet shows the syntax for using aggregation pipelines in **updateMany()**. It is the same for all other update functions:

```
db.collection.updateMany(  
  <query condition>,  
  [<update expression 1>, <update expression 2>, ...],  
  <options>  
)
```

Introduction

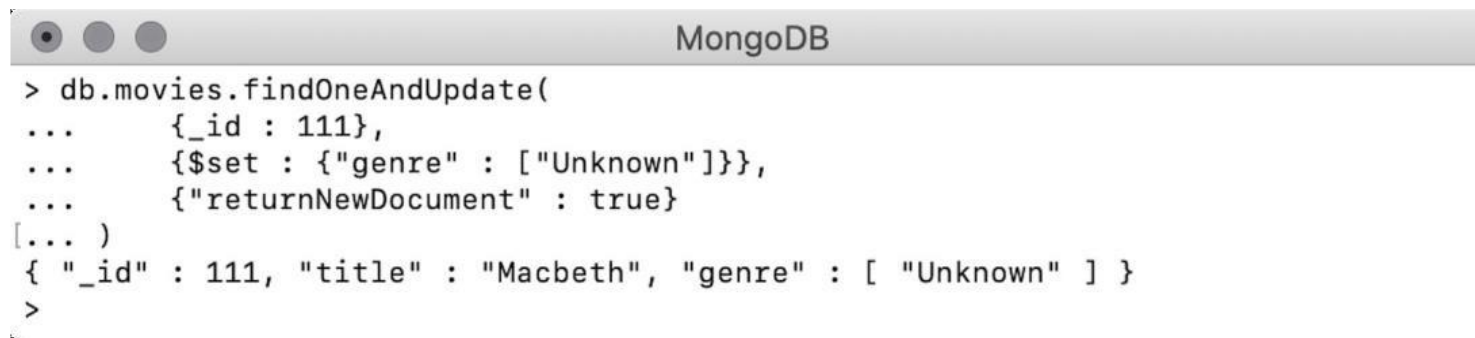
- ❑ In more complex update operations, you may need to use dynamically derived fields that are based on the values of other fields. Or, the update operation may involve multiple steps of update expressions
- ❑ Previous versions of MongoDB, referring to other fields' values or writing multi-step update operations was not possible, but, with the release of MongoDB 4.2, all of its update functions have started supporting aggregation pipelines.
- ❑ *The aggregation pipelines and various aggregation operators will be covered in **detail** in Chapter 7, Aggregation Pipelines. For now, we will limit the discussion to writing update expressions using pipeline support.*

Creating Array Fields

- ❑ \$Set -Adds new fields to the documents.

{ \$set: { <new field>: <expression, ...> } }

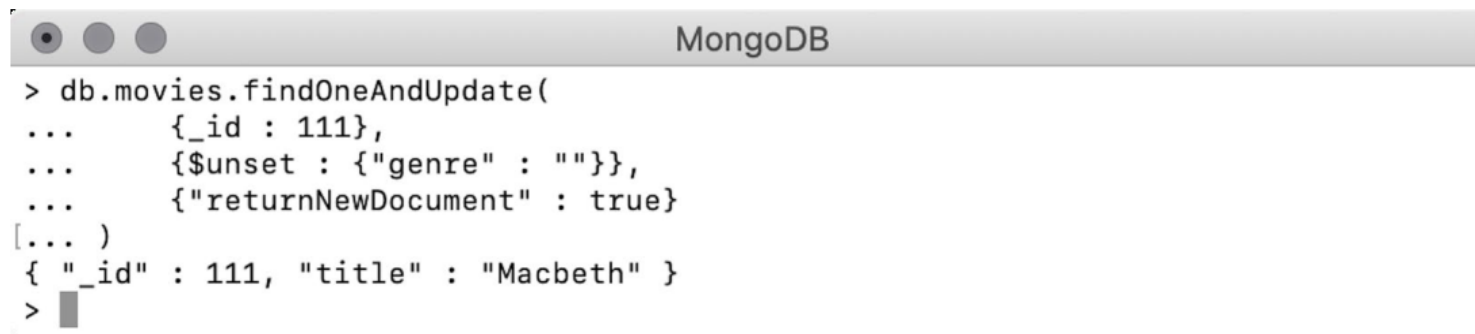
Ex:-



```
MongoDB
> db.movies.findOneAndUpdate(
...   { _id : 111 },
...   { $set : { "genre" : [ "Unknown" ] } },
...   { "returnNewDocument" : true }
... )
{ "_id" : 111, "title" : "Macbeth", "genre" : [ "Unknown" ] }
```

- ❑ \$unset -To remove fields from documents

Ex:-

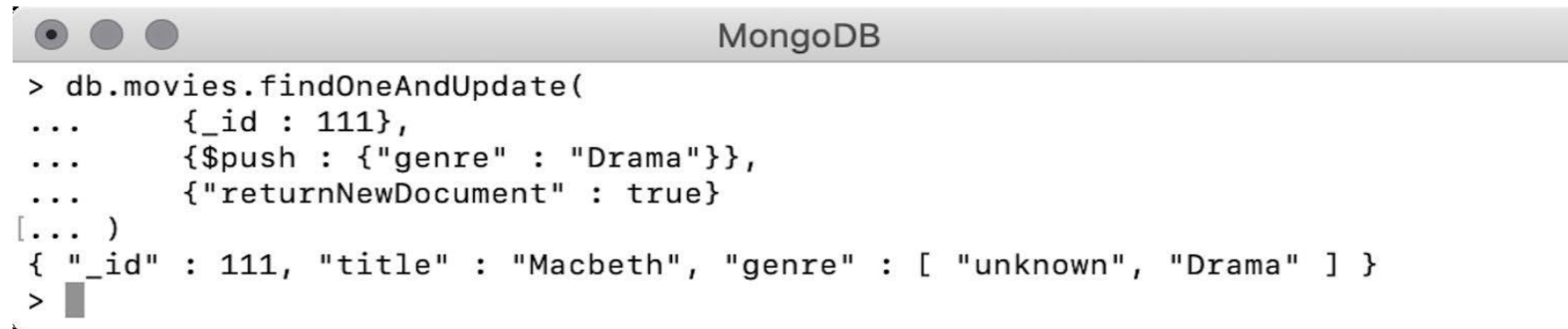


```
MongoDB
> db.movies.findOneAndUpdate(
...   { _id : 111 },
...   { $unset : { "genre" : "" } },
...   { "returnNewDocument" : true }
... )
{ "_id" : 111, "title" : "Macbeth" }
```

Array is treated like any other field.

Adding Array Fields

- ❑ **\$push** -can be used to add more elements to an array The operator pushes a given element to the end of an array, and if the given field is not present, it is created.



```
MongoDB
> db.movies.findOneAndUpdate(
...   { _id : 111 },
...   { $push : { "genre" : "Drama" } },
...   { "returnNewDocument" : true }
... )
{ "_id" : 111, "title" : "Macbeth", "genre" : [ "unknown", "Drama" ] }
```

- ❑ **Adding Multiple Elements**

`$push : { <field_name> : { $each : [<element 1>, <element2>, ..] } }`

- ❑ When such an update expression is executed, **\$each** iterates through each element, and the element is pushed to the array:

Sort Array

- ❑ Arrays in MongoDB, and in general, are an ordered but unsorted collection of elements, the array will always remain in the order in which they were inserted.
- ❑ **\$sort**-while executing an update command with \$push, we can also sort an array, use the \$sort operator with \$each.
 - \$sort : 1 –Ascending.
 - \$sort : -1- Descending.
 - ❑ if we have an array of objects that contains multiple fields, sorting can be performed based on the fields of nested objects.
 - \$sort : {"price" : -1}
- ❑ The **items** field is an array of objects, each containing three fields. We will sort the array by price in descending order.

An Array as a Set

- ❑ An array is an ordered collection of elements that can be iterated over or accessed using its specific index position.
- ❑ A set is a collection of unique elements whose order is not guaranteed.
- ❑ MongoDB supports only plain arrays and no other types of collections, However, you may want your array to contain unique elements only. MongoDB provides a way to do that by using the **\$addToSet** operator.
- ❑ The **\$addToSet** operator is like **\$push**, with the only difference being that an element will be pushed only if it is not present already. This operator does not change the underlying array, but it ensures that only unique elements are pushed into it.

An Array as a Set

- ❑ Add new category based on condition

Ex:- Classics are the only movies that both critics and viewers have rated above 95. So, your company wants to assign all those movies in the database to a new genre, called "Classic."

- Note that the genres in the array should always be unique and so you would use **\$addToSet** instead of **\$push** to add the **Classic** element to the **genres** array.

```
MongoDB
> db.movies.updateMany(
...   {
...     "tomatoes.viewer.meter" : {$gt : 95},
...     "tomatoes.critic.meter" : {$gt : 95}
...   },
...   {
...     $addToSet : {"genre" : "Classic"}
...   }
... )
{ "acknowledged" : true, "matchedCount" : 30, "modifiedCount" : 30 }
>
```

Removing Array Elements

- ❑ MongoDB also provides the means of removing elements from arrays.
- ❑ The **\$pop** operator, when used in an update command, allows you to remove the first or last element in an array.
- ❑ It removes one element at a time and can only be used with the values

1 (for the last element) or -1 (for the first element):

- ❑ **Removing All Elements**

When you only need to remove certain elements from an array, you can use the **\$pullAll** operator.

To do so, you provide one or more elements to the operator, which then removes all occurrences of those elements from the array.

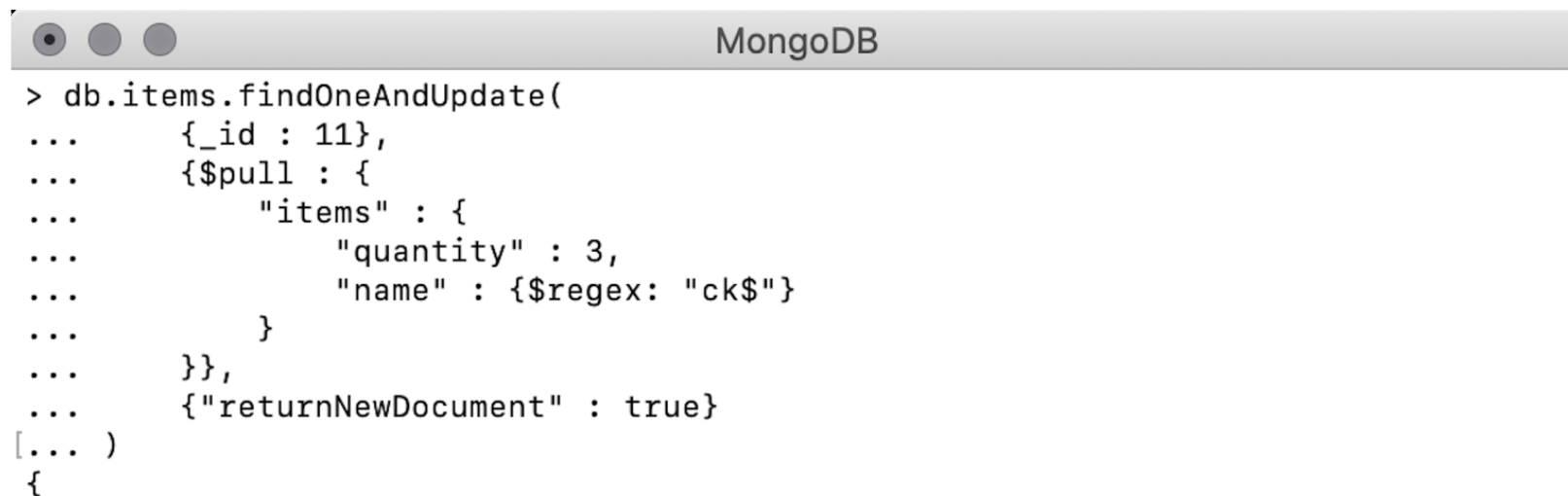
```
MongoDB
> db.movies.findOneAndUpdate(
...   { _id : 111 },
...   { $pullAll : { "genre" : [ "Action", "Crime" ] } },
...   { "returnNewDocument" : true }
... )
{ "_id" : 111, "title" : "Macbeth", "genre" : [ "History", "Drama" ] }
[>
```

Removing Array Elements

❑ Removing Matched Elements

We will use another operator, called **\$pull**, to write a query condition, using various logical and conditional operators, and the array elements that match the query will then be removed just like any **find** query .

Ex:- In this update command, the **\$pull** operator is provided with a query condition in the array field **items**. The conditions filter the array elements, where the **quantity** is **3** and the **name** ends with 'ck'.

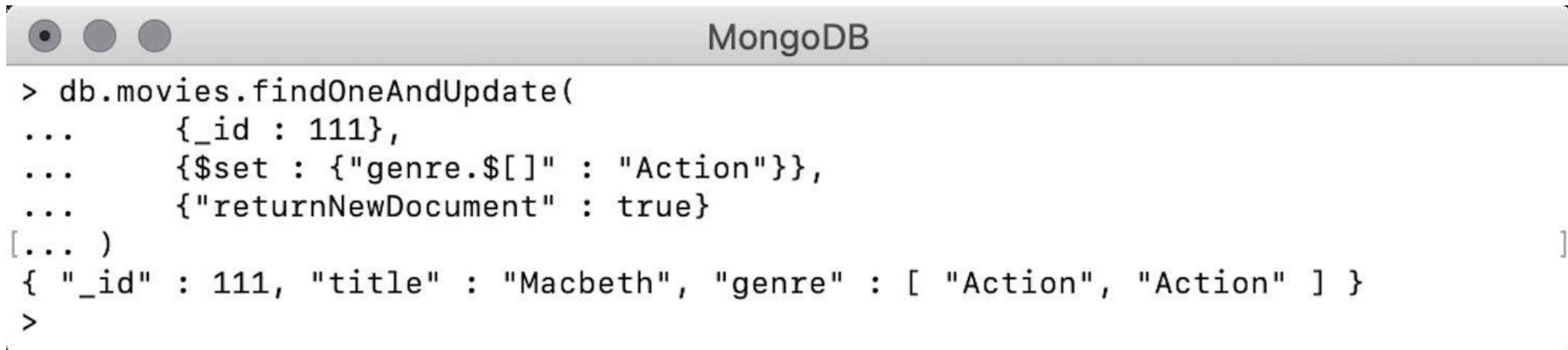


```
MongoDB
> db.items.findOneAndUpdate(
...   {_id : 11},
...   {$pull : {
...     "items" : {
...       "quantity" : 3,
...       "name" : {$regex: "ck$"}
...     }
...   }},
...   {"returnNewDocument" : true}
[... ]
{
```

Updating Array Elements

- ❑ In an array, each element is bound to a specific index position. These index positions start at zero, and we can use a pair of square brackets ([,]) with the respective index position to refer to an element from the array.
- ❑ Using such a pair of square brackets with \$ allows you to update elements of an array.
- ❑ Ex:- > db.movies.find({"_id" : 111})

```
{ "_id" : 111, "title" : "Macbeth", "genre" : [ "History", "Drama" ] }
```



```
MongoDB
> db.movies.findOneAndUpdate(
...   { _id : 111 },
...   { $set : { "genre.$[]" : "Action" } },
...   { "returnNewDocument" : true }
[... ]
{ "_id" : 111, "title" : "Macbeth", "genre" : [ "Action", "Action" ] }
>
```

The `$[]` operator refers to all the elements contained by the given array and the update expression will be applied to all of them

Updating Array Elements

- ❑ we can also update specific elements from an array, To do so, we first need to find such elements and identify them.
- ❑ To derive an element identifier, we can use the update option of **arrayFilters** to provide a query condition and assign it a variable (known as an identifier) to the matching elements.
- ❑ We then use the identifier along with **\$[]** to update the values of those specific elements.

```
db.items.findOneAndUpdate(  
  {  
    _id : 11},  
    {  
      $set : { "items.$[myElements]" : { "quantity" : 7, "price" : 4.5, "name" : "marker" }  
    }, {  
      "returnNewDocument" : true,  
      "arrayFilters" : [{"myElements.quantity" : null }]  
    }  
  } )
```

- ❑ The query condition of **{quantity: null}** is matched by the last element of the array and has been updated with the new document.

Summary

- ❑ In this session, you learned about:
 - We started this chapter by learning how to update documents using aggregation pipeline support. .
 - Pipeline support, which was introduced in MongoDB version 4.2, helps us to perform some complex updates. Using pipeline support, we can write multi-stage update expressions, where the output of a stage is provided as input to the next stage. .
 - It also allows us to use field references and aggregation operators. We also learned how to manipulate elements in array fields, how to add, remove, and update elements in an array, how to sort an array, and how to add only unique elements to an array. .
- ❑ In the next chapter, we will learn about MongoDB aggregation framework and pipeline in detail.