Introduction to MongoDB updateOne() method

The updateOne() method allows you to update a single document that satisfies a condition.

The following shows the syntax of the updateOne() method:

db.collection.updateOne(filter, update, options)

Code language: CSS (css)

In this syntax:

- The filter is a document that specifies the criteria for the update. If the filter matches multiple documents, then the updateOne() method updates only the first document. If you pass an empty document {} into the method, it will update the first document returned in the collection.
- The update is a document that specifies the change to apply.
- The options argument provides some options for updates that won't be covered in this tutorial.

The updateOne() method returns a document that contains some fields. The notable ones are:

- The matchedCount returns the number of matched documents.
- The modifiedCount returns the number of updated documents. In the case of the updateOne() method, it can be either 0 or 1.

$set operator

The $set operator allows you to replace the value of a field with a specified value.
The $set operator has the following syntax:

{ $set: { <field1>: <value1>, <field2>: <value2>, ...}}

Code language: HTML, XML (xml)

If the field doesn't exist, the $set operator will add the new field with the specified value to the document as long as the new field doesn't violate a type constraint.

If you specify the field with the dot notation e.g., embededDoc.field and the field does not exist, the $set will create the embedded document (embedded).

MongoDB updateOne() method examples

We'll use the following products collection:

db.products.insertMany([

```
   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec"
: { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},

   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec"
: { "ram" : 16, "screen" : 9.5, "cpu" : 3.66
},"color":["white","black","purple"],"storage":[128,256,512]},

   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"),
"spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},

   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-
14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"),
"spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256]}

 ])
```

Code language: JavaScript (javascript)

1) Using the MongoDB updateOne() method to update a single document

The following example uses the updateOne() method to update the price of the document with _id: 1:

```
db.products.updateOne({
   _id: 1
}, {
   $set: {
      price: 899
   }
})
```

Code language: PHP (php)

In this query:

- The { _id : 1 } is the filter argument that matches the documents to update. In this example, it matches the document whose _id is 1.
- The { $set: { price: 899 } } specifies the change to apply. It uses the $set operator to set the value of the price field to 899.

The query returns the following result:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

In this result document, the matchedCount indicates the number of matching documents (1) and the modifiedCount shows the number of the updated documents (1).

To verify the update, you can use the findOne() method to retrieve the document _id: 1 as follows:

```
db.products.findOne({ _id: 1 }, { name: 1, price: 1 })
```

Code language: CSS (css)

It returned the following document:

```
{ _id: 1, name: 'xPhone', price: 899 }
```

Code language: CSS (css)

As you can see clearly from the output, the price has been updated successfully.

2) Using the MongoDB updateOne() method to update the first matching document

The following query selects the documents from the products collection in which the value of the price field is 899:

```
db.products.find({ price: 899 }, { name: 1, price: 1 })
```

Code language: CSS (css)

It returned the following documents:

```
[
  { _id: 1, name: 'xPhone', price: 899 },
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 }
```

]

Code language: JavaScript (javascript)

The following example uses the updateOne() method to update the first matching document where the price field is 899:

```
db.products.updateOne({ price: 899 }, { $set: { price: null } })
```

Code language: PHP (php)

It updated one document as shown in the following result:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

If you query the document with _id: 1, you'll see that its price field is updated:

```
db.products.find({ _id: 1}, { name: 1, price: 1 })
```

Code language: CSS (css)

Output:

```
[ { _id: 1, name: 'xPhone', price: null } ]
```

Code language: JavaScript (javascript)

3) Using the updateOne() method to update embedded documents

The following query uses the find() method to select the document with _id: 4:

```
db.products.find({ _id: 4 }, { name: 1, spec: 1 })
```

Code language: CSS (css)

It returned the following document:

```
[
  {
```

```
    _id: 4,
    name: 'SmartPad',
    spec: { ram: 8, screen: 9.7, cpu: 1.66 }
  }
]
```

Code language: JavaScript (javascript)

The following example uses the updateOne() method to update the values of the ram, screen, and cpu fields in the spec embedded document of the document _id: 4:

```
db.products.updateOne({
    _id: 4
}, {
    $set: {
        "spec.ram": 16,
        "spec.screen": 10.7,
        "spec.cpu": 2.66
    }
})
```

Code language: PHP (php)

It returned the following document:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

If you query the document with _id 4 again, you'll see the change:

```
db.products.find({ _id: 4 }, { name: 1, spec: 1 })
```

Code language: CSS (css)

Output:

```
[
  {
    _id: 4,
    name: 'SmartPad',
    spec: { ram: 16, screen: 10.7, cpu: 2.66 }
  }
]
```

Code language: JavaScript (javascript)

4) Using the MongoDB updateOne() method to update array elements

The following example uses the updateOne() method to update the first and second elements of the storage array in the document with _id 4:

```
db.products.updateOne(
 { _id: 4},
 {
    $set: {
        "storage.0": 16,
        "storage.1": 32
    }
 }
)
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
```

```
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

If you query the document with _id 4 from the products collection, you'll see that the first and second elements of the storage array have been updated:

```
db.products.find({ _id: 4 }, { name: 1, storage: 1 });
```

Code language: CSS (css)

Output:

```
[ { _id: 4, name: 'SmartPad', storage: [ 16, 32, 1024 ] } ]
```

Code language: CSS (css)

Summary

- Use the updateOne() method to update the first document within a collection that satisfies a condition.
- Use the $set operator to replace the value of a field with a specified value.

Introduction to MongoDB updateMany() method

The updateMany() method allows you to update all documents that satisfy a condition.

The following shows the syntax of the updateMany() method:

```
db.collection.updateMany(filter, update, options)
```

Code language: CSS (css)

In this syntax:

- The filter is a document that specifies the condition to select the document for update. If you pass an empty document ({}) into the method, it'll update all the documents the collection.
- The update is a document that specifies the updates to apply.
- The options argument provides some options for updates that won't be covered in this tutorial.

The updateMany() method returns a document that contains multiple fields. The following are the notable ones:

- The matchedCount stores the number of matched documents.
- The modifiedCount stores the number of modified documents.

To form the update argument, you typically use the $set operator.

$set operator

The $set operator replaces the value of a field with a specified value. It has the following syntax:

{ $set: { <field1>: <value1>, <field2>: <value2>, ...}}

Code language: HTML, XML (xml)

If the field doesn't exist, the $set operator will add the new field with the specified value to the document. If you specify the field with the dot notation e.g., embededDoc.field and the field does not exist, the $set operator will create the embedded document (embedded).

MongoDB updateMany() method examples

We'll use the following products collection:

```
db.products.insertMany([

   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},

   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66
},"color":["white","black","purple"],"storage":[128,256,512]},

   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},

   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256]}

 ])
```

Code language: JavaScript (javascript)

1) Using the MongoDB updateMany() method to update multiple documents

The following example uses the updateMany() method to update the documents where the value of the price field is 899:

```
db.products.updateMany(
   {  price: 899},
   { $set: {  price: 895 }}
)
```

Code language: PHP (php)

In this query:

The { price : 899 } is the filter argument that specified the documents to update.

The { $set: { price: 895} } specifies the update to apply, which uses the $set operator to set the value of the price field to 895.

The query returns the following result:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
```

Code language: CSS (css)

In this result document, the matchedCount stores the number of matching documents (2) and the modifiedCount stores the number of the updated documents (2).

To check the update, you can use the find() method to select the documents where the value of the price field is 895 as follows:

```
db.products.find({
   price: 895
}, {
   name: 1,
   price: 1
```

```
})
```

Code language: CSS (css)

The query returned the following documents:

```
[
  { _id: 2, name: 'xTablet', price: 895 },
  { _id: 3, name: 'SmartTablet', price: 895 }
]
```

Code language: JavaScript (javascript)

The price field values have been updated successfully.

2) Using the updateMany() method to update embedded documents

The following query uses the find() method to select the documents where the value in the price field is greater than 700:

```
db.products.find({
    price: { $gt: 700}
}, {
    name: 1,
    price: 1,
    spec: 1
})
```

Code language: CSS (css)

The query returned the following documents:

```
[
  {
    _id: 1,
    name: 'xPhone',
    price: 799,
    spec: { ram: 4, screen: 6.5, cpu: 2.66 }
  },
```

```
  {
    _id: 2,
    name: 'xTablet',
    price: 895,
    spec: { ram: 16, screen: 9.5, cpu: 3.66 }
  },
  {
    _id: 3,
    name: 'SmartTablet',
    price: 895,
    spec: { ram: 12, screen: 9.7, cpu: 3.66 }
  }
]
```

Code language: JavaScript (javascript)

The following example uses the updateMany() method to update the values of the ram, screen, and cpu fields in the spec embedded documents of these documents:

```
db.products.updateMany({
    price: { $gt: 700}
}, {
    $set: {
        "spec.ram": 32,
        "spec.screen": 9.8,
        "spec.cpu": 5.66
    }
})
```

Code language: PHP (php)

The query returned the following document indicating that the three documents have been updated successfully:

```
{
```

```
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
```

Code language: CSS (css)

3) Using the MongoDB updateMany() method to update array elements

The following example uses the updateMany() method to update the first and second elements of the storage array of the documents where the _id is 1, 2 and 3:

```
db.products.updateMany({
   _id: {
      $in: [1, 2, 3]
   }
}, {
   $set: {
      "storage.0": 16,
      "storage.1": 32
   }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
```

}

Code language: CSS (css)

If you query the documents whose _id is 1, 2, and 3 from the products collection, you'll see that the first and second elements of the storage array have been updated:

```
db.products.find(
  { _id: { $in: [1,2,3]}},
  { name: 1, storage: 1}
)
```

Code language: CSS (css)

Output:

```
[
 { _id: 1, name: 'xPhone', storage: [ 16, 32, 256 ] },
 { _id: 2, name: 'xTablet', storage: [ 16, 32, 512 ] },
 { _id: 3, name: 'SmartTablet', storage: [ 16, 32, 128 ] }
]
```

Code language: JavaScript (javascript)

Summary

- Use the updateMany() method to update all the documents within a collection that satisfy a condition.
- Use the $set operator to replace the value of a field with a specified value.

Introduction to the MongoDB $inc operator

Sometimes, you need to increment the value of one or more fields by a specified value. In this case, you can use the update() method with the $inc operator.

The $inc operator has the following syntax:

```
{ $inc: {<field1>: <amount1>, <field2>: <amount2>, ...} }
```

Code language: HTML, XML (xml)

In this syntax, the amount can be positive or negative. When it's a positive value, the $inc increases the field by amount. If the amount is a negative value, then the $inc decreases the field by the absolute value of the amount.

If the field doesn't exist, the $inc creates the field and sets the field to the specified amount.

MongoDB $inc operator examples

We'll use the following products collection:

```
db.products.insertMany([
   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},
   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 },"color":["white","black","purple"],"storage":[128,256,512]},
   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},
   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256,1024]},
   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256]}
])
```

Code language: JavaScript (javascript)

1) Using the MongoDB $inc operator to increase the value of a field

The following example uses the $inc operator to increase the price of the document _id 1 from the products collection by 50:

```
db.products.updateOne({
   _id: 1
}, {
   $inc: {
      price: 50
   }
```

})

Code language: PHP (php)

It returned the following document indicating that one document matched and has been updated:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

If you query the document _id 1, you'll see that the price has been increased:

```
db.products.find(
   {_id: 1},
   {name: 1, price: 1}
)
```

Code language: CSS (css)

Output:

```
[ { _id: 1, name: 'xPhone', price: 849 } ]
```

Code language: CSS (css)

2) Using the MongoDB $inc operator to decrease the value of a field

The following example uses the $inc operator to decrease the price by 150:

```
db.products.updateOne({
   _id: 1
}, {
   $inc: {
      price: -150
```

```
    }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

The price of the document _id 1 has been decreased as shown in the output of the following query:

```
db.products.find(
    { _id: 1 },
    { name: 1, price: 1 }
)
```

Code language: CSS (css)

Output:

```
[ { _id: 1, name: 'xPhone', price: 699 } ]
```

Code language: CSS (css)

3) Using MongoDB $inc operator to update values of multiple fields

The following example uses the $inc operator to increase the value of the price field as well as the ram field of the spec embedded document:

```
db.products.updateOne({
    _id: 1
}, {
    $inc: {
```

```
        price: 50,
        "spec.ram": 4
    }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

Note that to specify a field in an embedded document, you use the dot notation e.g., "spec.ram".

This query selects the document _id 1 to verify the update:

```
db.products.find(
    {_id: 1},
    {name: 1, price: 1, "spec.ram": 1}
)
```

Code language: CSS (css)

The values of the price and ram fields have been increased as shown in the following output:

```
[ { _id: 1, name: 'xPhone', price: 749, spec: { ram: 8 } } ]
```

Code language: CSS (css)

Summary

- Use the $inc operator to increase a value of a field by a specified amount.
- Use a negative value in the $inc operator to decrease the value of the field.

Introduction to the MongoDB $min operator

The $min operator is a field update operator that allows you to update the value of a field to a specified value if the specified value is less than (<) the current value of the field.

The $min operator has the following syntax:

{ $min: {<field1>: <value1>, ...} }

Code language: HTML, XML (xml)

If the current value of a field is greater than or equal to the value that you want to update, the $min operator won't update the value.

If the field doesn't exist, the $min operator creates the field and sets its value to the specified value.

MongoDB $min operator example

We'll use the following products collection:

```
db.products.insertMany([
   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},
   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 },"color":["white","black","purple"],"storage":[128,256,512]},
   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},
   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256,1024]},
   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256]}
])
```

Code language: JavaScript (javascript)

The following example uses the $min operator to update the price of the document _id 5:

```
db.products.updateOne({
   _id: 5
```

```
}, {
    $min: {
        price: 699
    }
})
```

Code language: PHP (php)

The query found a matching document. However, it didn't update any:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Code language: CSS (css)

The reason is that the new value 699 is greater than the current value 599.

The following example uses the $min operator to update the price of the document _id 5:

```
db.products.updateOne({
    _id: 5
}, {
    $min: {
        price: 499
    }
})
```

Code language: PHP (php)

In this case, the price field of the document _id 5 is updated to 499:

```
{
  acknowledged: true,
```

```
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

This query verifies the update:

```
db.products.find({ _id: 5 }, { name: 1, price: 1 })
```

Code language: CSS (css)

Output:

```
[ { _id: 5, name: 'SmartPhone', price: 499 } ]
```

Code language: CSS (css)

Summary

- Use the $min operator to update the value of a field to a specified value when the specified value is less than the current field value.

Introduction to the MongoDB $max operator

The $max is a field update operator that allows you to update the value of a field to a specified value if the specified value is **greater than** (>) the current value of the field.

Here is the syntax of the $max operator:

```
{ $max: {<field1>: <value1>, ...} }
```

Code language: HTML, XML (xml)

If the current value of a field is less than or equal to the value that you want to update, the $max operator won't update the value.

If the field doesn't exist, the $max operator creates the field and sets its value to the specified value.

MongoDB $max operator example

We'll use the following products collection:

```
db.products.insertMany([
```

```
   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec"
: { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},

   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec"
: { "ram" : 16, "screen" : 9.5, "cpu" : 3.66
},"color":["white","black","purple"],"storage":[128,256,512]},

   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"),
"spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},

   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-
14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"),
"spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256]}

 ])
```

Code language: JavaScript (javascript)

The following example uses the $max operator to update the price of the document _id 1:

```
db.products.updateOne({
   _id: 1
}, {
   $max: {
      price: 699
   }
})
```

Code language: PHP (php)

The query found one matching document but it didn’t update any document as shown in the modifiedCount value:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
```

```
  upsertedCount: 0
}
```

Code language: CSS (css)

The reason is that the new value that we want to update is 699 which is less than the current value of the price field 799.

The following example uses the $min operator to update the price of the document _id 1 to 899:

```
db.products.updateOne({
   _id: 1
}, {
   $max: {
      price: 899
   }
})
```

Code language: PHP (php)

In this case, the price field of the document _id 1 is updated to 899:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

This query verifies the update:

```
db.products.find({ _id: 1 }, { name: 1, price: 1 })
```

Code language: CSS (css)

Output:

```
[ { _id: 1, name: 'xPhone', price: 899 } ]
```

Code language: CSS (css)

Summary

- Use the $max operator to update the value of a field to a specified value when the specified value is greater than the current value.

Introduction to MongoDB $mul operator

The $mul is a field update operator that allows you to **multiply** the value of a field by a specified number.

The $mul operator has the following syntax:

{ $mul: { <field1>: <number1>, <field2>: <number2>, ... } }

Code language: HTML, XML (xml)

The <field> that you want to update must contain a numeric value. To specify a field in an embedded document or in an array, you use the dot notation e.g., <embedded_doc>.<field> or <array>.<index>

If the field doesn't exist in the document, the $mul operator creates the field and sets its value to zero.

MongoDB $mul operator examples

We'll use the following products collection:

```
db.products.insertMany([
   { "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},
   { "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 },"color":["white","black","purple"],"storage":[128,256,512]},
   { "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},
   { "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256,1024]},
   { "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256]}
```

])

Code language: JavaScript (javascript)

1) Using the MongoDB $mul to multiply the value of a field

The following example uses the $mul operator to multiply the price of the document _id 5 by 10%:

```
db.products.updateOne({ _id: 5 }, { $mul: { price: 1.1 } })
```

Code language: PHP (php)

The output document showed that the query matched one document and updated it:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query verifies the update:

```
db.products.find({
    _id: 5
}, {
    name: 1,
    price: 1
})
```

Code language: CSS (css)

Output:

```
[ { _id: 5, name: 'SmartPhone', price: 658.9000000000001 } ]
```

Code language: CSS (css)

2) Using the MongoDB $mul to multiply the values of array elements

The following query uses the $mul operator to double the values of the first, second, and third array elements in the storage array of the document _id 1:

```
db.products.updateOne({
   _id: 1
}, {
   $mul: {
      "storage.0": 2,
      "storage.1": 2,
      "storage.2": 2
   }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query uses the findOne() method to select the document with _id 1 to verify the update:

```
 db.products.findOne({ _id: 1 }, { name: 1, storage: 1 })
```

Code language: CSS (css)

Output:

```
{ _id: 1, name: 'xPhone', storage: [ 128, 256, 512 ] }
```

Code language: CSS (css)

3) Using the $mul operator to multiply the values of a field in embedded documents

This example uses the $mul operator to double the values of the ram field in the spec embedded documents of all documents from the products collection:

```
db.products.updateMany({}, {
   $mul: {
      "spec.ram": 2
   }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query returns all documents from the products collection:

```
db.products.find({}, { name: 1, "spec.ram": 1 })
```

Code language: CSS (css)

Output:

```
[
  { _id: 1, name: 'xPhone', spec: { ram: 8 } },
  { _id: 2, name: 'xTablet', spec: { ram: 32 } },
  { _id: 3, name: 'SmartTablet', spec: { ram: 24 } },
  { _id: 4, name: 'SmartPad', spec: { ram: 16 } },
  { _id: 5, name: 'SmartPhone', spec: { ram: 8 } }
]
```

Code language: JavaScript (javascript)

Summary

- Use the MongoDB $mul operator to **multiply** the value of a field by a specified number.

Introduction to the MongoDB $unset operator

Sometimes, you may want to remove one or more fields from a document. In order to do it, you can use the $unset operator.

The $unset is a field update operator that completely removes a particular field from a document.

The $unset operator has the following syntax:

{ $unset: {<field>: "", ... }}

Code language: PHP (php)

In this syntax, you specify the field that you want to remove and its value. The field value isn't important and doesn't impact the operation. You can specify any value, the $unset will remove the field completely. If the <field> doesn't exist in the document,
then $unset operator will do nothing. It also won't issue any warnings or errors.

To specify a field in an embedded document, you use the dot notation like this:

{ $unset: { "<embedded_doc>.<field>: "", ... }}

Code language: HTML, XML (xml)

Note that the $unset operator doesn't remove array elements. Instead, it sets the array elements to null.

{ $unset: {"<array>.<index>": "", ...}

Code language: PHP (php)

This behavior keeps the array size and element positions consistent.

MongoDB $unset operator examples

We'll use the following products collection:

db.products.insertMany([

{ "_id" : 1, "name" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},

{ "_id" : 2, "name" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 },"color":["white","black","purple"],"storage":[128,256,512]},

{ "_id" : 3, "name" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},

{ "_id" : 4, "name" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

{ "_id" : 5, "name" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256]}

])

Code language: JavaScript (javascript)

1) Using the MongoDB $unset operator to remove a field from a document

The following example uses the $unset operator to remove the price field from the document _id 1 in the products collection:

```
db.products.updateOne({
   _id: 1
}, {
   $unset: {
      price: ""
   }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
```

```
  upsertedCount: 0
}
```

Code language: CSS (css)

The modifiedCount indicated that one document has been modified. This query returns all documents from the products collection to verify the update:

```
db.products.find({}, { name: 1, price: 1 })
```

Code language: CSS (css)

Output:

```
[
  { _id: 1, name: 'xPhone' },
  { _id: 2, name: 'xTablet', price: 899 },
  { _id: 3, name: 'SmartTablet', price: 899 },
  { _id: 4, name: 'SmartPad', price: 699 },
  { _id: 5, name: 'SmartPhone', price: 599 }
]
```

Code language: JavaScript (javascript)

As you can see clearly from the output, the $unset operator has completely removed the price field from the document with _id 1.

2) Using the MongoDB $unset operator to remove a field in an embedded document

The following statement uses the $unset operator to remove the ram field from the spec embedded documents of all documents in the products collection:

```
db.products.updateMany({}, {
   $unset: {
      "spec.ram": ""
   }
})
```

Code language: PHP (php)

Output:

```
{
```

```
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query returns all documents from the products collection:

```
db.products.find({}, {
    spec: 1
})
```

Code language: CSS (css)

Output:

```
[
  { _id: 1, spec: { screen: 6.5, cpu: 2.66 } },
  { _id: 2, spec: { screen: 9.5, cpu: 3.66 } },
  { _id: 3, spec: { screen: 9.7, cpu: 3.66 } },
  { _id: 4, spec: { screen: 9.7, cpu: 1.66 } },
  { _id: 5, spec: { screen: 5.7, cpu: 1.66 } }
]
```

As you can see, the ram field has been removed from spec embedded document in all documents.

3) Using the MongoDB $unset operator to set array elements to null

The following example uses the $unset operator to set the first elements of the storage arrays to null:

```
 db.products.updateMany({}, { $unset: { "storage.0": "" } })
```

Code language: PHP (php)

Output:

```
{
```

```
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query selects the storage array from all documents in the products collection:

```
db.products.find({}, { "storage":1})
```

Code language: JavaScript (javascript)

Output:

```
[
  { _id: 1, storage: [ null, 128, 256 ] },
  { _id: 2, storage: [ null, 256, 512 ] },
  { _id: 3, storage: [ null, 64, 128 ] },
  { _id: 4, storage: [ null, 256, 1024 ] },
  { _id: 5, storage: [ null, 256 ] }
]
```

Code language: JavaScript (javascript)

In this example, the $unset operator sets the first elements of the storage arrays to null instead of removing them completely.

4) Using the MongoDB $unset operator to remove multiple fields from a document

The following statement uses the $unset operator to remove the releaseDate and spec fields from all the documents in the products collection:

```
db.products.updateMany({}, {
   $unset: {
      releaseDate: "",
      spec: ""
   }
```

```
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query verifies the update:

```
db.products.find({}, {
    name: 1,
    storage: 1,
    releaseDate: 1,
    spec: 1
})
```

Code language: CSS (css)

Output:

```
[
  { _id: 1, name: 'xPhone', storage: [ null, 128, 256 ] },
  { _id: 2, name: 'xTablet', storage: [ null, 256, 512 ] },
  { _id: 3, name: 'SmartTablet', storage: [ null, 64, 128 ] },
  { _id: 4, name: 'SmartPad', storage: [ null, 256, 1024 ] },
  { _id: 5, name: 'SmartPhone', storage: [ null, 256 ] }
]
```

Code language: JavaScript (javascript)

As shown clearly from the output, the releaseDate and spec fields now are gone.

Summary

- Use the $unset operator to completely remove a field from a document.

Introduction to the MongoDB $rename operator

Sometimes, you want to rename a field in a document e.g., when it is misspelled or not descriptive enough. In this case, you can use the $rename operator.

The $rename is a field update operator that allows you to rename a field in a document to the new one.

The $rename operator has the following syntax:

{ $rename: { <field_name>: <new_field_name>, ...}}

Code language: HTML, XML (xml)

In this syntax, the <new_field_name> must be different from the <field_name>.

If the document has a field with the same name as the <new_field_name>,
the $rename operator removes that field and renames the
specified <field_name> to <new_field_name>.

In case the <field_name> doesn't exist in the document, the $rename operator does nothing. It also won't issue any warnings or errors.

The $rename operator can rename fields in embedded documents. In addition, it can move these fields in and out of the embedded documents.

MongoDB $rename field operator examples

We'll use the following products collection:

```
db.products.insertMany([
   { "_id" : 1, "nmea" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},
   { "_id" : 2, "nmea" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66
},"color":["white","black","purple"],"storage":[128,256,512]},
   { "_id" : 3, "nmea" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},
```

```
  { "_id" : 4, "nmea" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-
14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

  { "_id" : 5, "nmea" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"),
"spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66
},"color":["white","orange","gold","gray"],"storage":[128,256]}

 ])
```

Code language: JavaScript (javascript)

1) Using MongoDB $rename to rename a field in a document

The following example uses the $rename operator to rename the misspelled field nmea to name:

```
db.products.updateMany({}, {
   $rename: {
      nmea: "name"
   }
})
```

Code language: PHP (php)

In this example, the $rename operator changed the field name from nmea to name as indicated in the following returned document:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

Code language: CSS (css)

To verify the update, you can use the find() method to select all documents from the products collection:

```
db.products.find({}, { name: 1 })
```

Code language: CSS (css)

Output:

```
[
  { _id: 1, name: 'xPhone' },
  { _id: 2, name: 'xTablet' },
  { _id: 3, name: 'SmartTablet' },
  { _id: 4, name: 'SmartPad' },
  { _id: 5, name: 'SmartPhone' }
]
```

Code language: JavaScript (javascript)

2) Using MongoDB $rename operator to rename fields in embedded documents

The following example uses the $rename operator to change the size field of the spec embedded document to screenSize:

```
db.products.updateMany({}, {
   $rename: {
      "spec.screen": "spec.screenSize"
   }
})
```

Code language: PHP (php)

It returned the following result:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Code language: CSS (css)

This query uses the find() method to select all documents from the products collection:

```
db.products.find({}, {
   spec: 1
})
```

Code language: CSS (css)

Here is the output:

```
[
 { _id: 1, spec: { ram: 4, cpu: 2.66, screenSize: 6.5 } },
 { _id: 2, spec: { ram: 16, cpu: 3.66, screenSize: 9.5 } },
 { _id: 3, spec: { ram: 12, cpu: 3.66, screenSize: 9.7 } },
 { _id: 4, spec: { ram: 8, cpu: 1.66, screenSize: 9.7 } },
 { _id: 5, spec: { ram: 4, cpu: 1.66, screenSize: 5.7 } }
]
```

As you can see from the output, the screen fields in the spec embedded documents have been renamed to screenSize.

3) Using the MongoDB $rename to move field out of the embedded document

The following example uses the $rename operator to move the cpu field out of the spec embedded document in the document _id 1:

```
db.products.updateOne({
 _id: 1
},
{
   $rename: {
      "spec.cpu": "cpu"
   }
})
```

Code language: PHP (php)

Output:

```
{
```

```
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Code language: CSS (css)

The following query selects the document with _id 1 to verify the rename:

```
db.products.find({ _id: 1})
```

Code language: CSS (css)

Output:

```
[
  {
    _id: 1,
    price: 799,
    releaseDate: ISODate("2011-05-14T00:00:00.000Z"),
    spec: { ram: 4, screenSize: 6.5 },
    color: [ 'white', 'black' ],
    storage: [ 64, 128, 256 ],
    name: 'xPhone',
    cpu: 2.66
  }
]
```

Code language: JavaScript (javascript)

As you can see clearly from the output, the cpu field becomes the top-level field.

4) Using the MongoDB $rename to rename a field to an existing field

The following example uses the $rename operator to rename the field color to storage in the document with _id 2.

However, the storage field already exists. Therefore, the $rename operator removes the storage field and renames the field color to storage:

```
db.products.updateOne({
   _id: 2
}, {
   $rename: {
      "color": "storage"
   }
})
```

Code language: PHP (php)

Output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Code language: CSS (css)

Let’s check the document _id 2:

```
db.products.find({ _id: 2 })
```

Code language: CSS (css)

Output:

```
[
  {
    _id: 2,
    price: 899,
    releaseDate: ISODate("2011-09-01T00:00:00.000Z"),
```

```
  spec: { ram: 16, cpu: 3.66, screenSize: 9.5 },
  storage: [ 'white', 'black', 'purple' ],
  name: 'xTablet'
 }
]
```

Code language: JavaScript (javascript)

Summary

- Use the MongoDB $rename operator to rename a field to a new one.

Introduction to the MongoDB upsert

Upsert is a combination of **up**date and in**sert**. Upsert performs two functions:

- Update data if there is a matching document.
- Insert a new document in case there is no document matches the query criteria.

To perform an upsert, you use the following updateMany() method with the upsert option set to true:

```
document.collection.updateMany(query, update, { upsert: true} )
```

Code language: CSS (css)

The upsert field in the third argument is set to false by default. This means that if you omit it, the method will only update the documents that match the query.

Notice that the updateOne() method also can upsert with the { upsert: true }.

MongoDB upsert examples

We'll use the following products collection.

```
db.products.insertMany([
  { "_id" : 1, "nmea" : "xPhone", "price" : 799, "releaseDate": ISODate("2011-05-14"), "spec" : { "ram" : 4, "screen" : 6.5, "cpu" : 2.66 },"color":["white","black"],"storage":[64,128,256]},
  { "_id" : 2, "nmea" : "xTablet", "price" : 899, "releaseDate": ISODate("2011-09-01") , "spec" : { "ram" : 16, "screen" : 9.5, "cpu" : 3.66 },"color":["white","black","purple"],"storage":[128,256,512]},
  { "_id" : 3, "nmea" : "SmartTablet", "price" : 899, "releaseDate": ISODate("2015-01-14"), "spec" : { "ram" : 12, "screen" : 9.7, "cpu" : 3.66 },"color":["blue"],"storage":[16,64,128]},
```

{ "_id" : 4, "nmea" : "SmartPad", "price" : 699, "releaseDate": ISODate("2020-05-14"),"spec" : { "ram" : 8, "screen" : 9.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256,1024]},

{ "_id" : 5, "nmea" : "SmartPhone", "price" : 599,"releaseDate": ISODate("2022-09-14"), "spec" : { "ram" : 4, "screen" : 5.7, "cpu" : 1.66 },"color":["white","orange","gold","gray"],"storage":[128,256]}

])

Code language: JavaScript (javascript)

The following query uses the update() method to update the price for the document _id 6:

```
db.products.updateMany(
   {_id: 6 },
   { $set: {price: 999} }
)
```

Code language: PHP (php)

The query found no match and didn't update any document as shown in the following output:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Code language: CSS (css)

If you pass the { upsert: true } to the updateMany() method, it'll insert a new document. For example:

```
db.products.updateMany(
   { _id: 6 },
   { $set: {price: 999} },
   { upsert: true}
```

)

Code language: PHP (php)

The query returns the following document:

{

  acknowledged: true,

  insertedId: 6,

  matchedCount: 0,

  modifiedCount: 0,

  upsertedCount: 1

}

Code language: CSS (css)

The output indicates that there was no matching document (matchedCount is zero) and the updateMany() method didn't update any document.

However, the updateMany() method inserted one document and returned the id of the new document stored in the upsertedId field.

If you query the document with _id 6 from the products collection, you'll see the new document with the price field:

db.products.find({_id:6})

Code language: CSS (css)

Output:

[{ _id: 6, price: 999 }]

Code language: CSS (css)

Summary

- Use the { upsert: true } argument in the updateMany() or updateOne() method to perform an upsert.