03 CRUD Operaciones de actualización

Método update()

https://docs.mongodb.com/manual/reference/method/db.collection.update/#db.collection.update

Sintaxis

```
db.collection.update(
  <query>,
  <update>,
  {
    upsert: <boolean>,
    multi: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    arrayFilters: [ <filterdocument1>, ... ],
    hint:  <document|string>        // Available starting in MongoDB 4.2
  }
)
```

Parameter	Type	Description
query	document	The selection criteria for the update. The same query selectors as in the find() method are available. When you execute an update() with upsert: true and the query matches no existing document, MongoDB will refuse to insert a new document if the query specifies conditions on the _id field using dot notation.

update	document or pipeline	The modifications to apply. Can be one of the following: Update document Contains only update operator expressions. Replacement document Contains only <field1>: <value1> pairs. Aggregation pipeline Consists only of the following aggregation stages: $addFields and its alias $set $project and its alias $unset $replaceRoot and its alias $replaceWith.
upsert	boolean	Optional. If set to true, creates a new document when no document matches the query criteria. The default value is false, which does not insert a new document when no match is found.
multi	boolean	Optional. If set to true, updates multiple documents that meet the query criteria. If set to false, updates one document. The default value is false.
writeConcern	document	Optional. A document expressing the write concern. Omit to use the

		default write concern w: 1. Do not explicitly set the write concern for the operation if run in a transaction. To use write concern with transactions, see Transactions and Write Concern.
collation	document	Optional. Collation allows users to specify language-specific rules for string comparison, such as rules for lettercase and accent marks.
arrayFilters	document	Optional. An array of filter documents that determine which array elements to modify for an update operation on an array field. In the update document, use the $[<identifier>] to define an identifier to update only those array elements that match the corresponding filter document in the arrayFilters.
hint	document or string	Optional. A document or string that specifies the index to use to support the query predicate. The option can take an index specification document or the index name string.

		If you specify an index that does not exist, the operation errors.

Replace Entire Document

If the <update> document contains only field:value expressions, then:

The db.collection.update() method replaces the matching document with the <update> document. The db.collection.update() method does not replace the _id value.

db.collection.update() cannot update multiple documents.

The following operation passes an <update> document that contains only field and value pairs. The <update> document completely replaces the original document except for the _id field.

Práctica

```
> db.titulos.insert({title:"Cien Años de Soledad", autor: "Gabriel García Márquez", stock: 10})
WriteResult({ "nInserted" : 1 })
> db.titulos.insert({title:"La Ciudad y los Perros", autor: "Mario Vargas LLosa", stock: 10, prestados: 2})
WriteResult({ "nInserted" : 1 })
> db.titulos.insert({title:"El Otoño del Patriarca", autor: "Gabriel García Márquez", stock: 10, prestados: 0})

> db.titulos.update({autor: /Gabriel/},{title:"Cien Años de Soledad", autor: "Gabriel García Márquez", stock: 10, prestados: 0})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.titulos.update({autor: /Gabriel/},{_id: "001A", title:"Cien Años de Soledad", autor: "Gabriel García Marquez", stock: 10, prestados: 0})
WriteResult({
```

```
        "nMatched" : 0,
        "nUpserted" : 0,
        "nModified" : 0,
        "writeError" : {
                "code" : 66,
                "errmsg" : "After applying the update, the (immutable) field '_id' was
found to have been altered to _id: \"001A\""
        }
}) // Error si se intenta añadir un nuevo _id en el documento de actualización
```

Use Update Operator Expressions

If the <update> document contains update operator modifiers, such as those using the $set modifier, then:

The <update> document must contain only update operator expressions.

The db.collection.update() method updates only the corresponding fields in the document.

To update an embedded document or an array as a whole, specify the replacement value for the field.

To update particular fields in an embedded document or in an array, use dot notation to specify the field.

$set

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

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

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior

If the field does not exist, $set will add a new field with the specified value, provided that the new field does not violate a type constraint. If you specify a dotted path for a non-existent field, $set will create the embedded documents as needed to fulfill the dotted path to the field.

If you specify multiple field-value pairs, $set will update or create each field.

Set Top-Level Fields

Práctica

```
> db.titulos.update({_id: ObjectId("5e281476e5cb58e060e229ff")},{$set: {prestados: 2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.titulos.update({_id: ObjectId("5e281476e5cb58e060e229ff")},{$set: {prestados: 4, categorias:["novela","castellano"]}}) // Actualiza cada campo y además lo crea si no exite
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Set Fields in Embedded Documents

Práctica

```
> db.titulos.insert({titulo:"El Quijote", autor: { nombre: "Miguel", apellidos: "Cervantes Saavedra", pais: "España"}})

> db.titulos.update({"_id" : ObjectId("5e281c85fe0a9f8b3c6230a9")}, {$set: {"autor.apellidos": "De Cervantes Saavedra"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Set Elements in Arrays

```
> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")},{$set: {"categorias.1":"español"}})
```

$setOnInsert

If an update operation with upsert: true results in an insert of a document, then $setOnInsert assigns the specified values to the fields in the document. If the update operation does not result in an insert, $setOnInsert does nothing.

```
db.collection.update(
   <query>,
   { $setOnInsert: { <field1>: <value1>, ... } },
   { upsert: true }
)
```

Práctica

```
> db.titulo.update({titulo: "La Historia Interminable"},{$setOnInsert: {titulo: "La
Historia Interminable", autor: "Michael Ende"}},{upsert: true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("5e2820858d5aab8edaeeff2c")
}) // Operación idempotente
```

$unset

The $unset operator deletes a particular field. Consider the following syntax:

{ $unset: { <field1>: "", ... } }
The specified value in the $unset expression (i.e. "") does not impact the operation.

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
If the field does not exist, then $unset does nothing (i.e. no operation).

Práctica

```
> db.titulos.update({titulo: "El Quijote"},{$unset: {autor: ""}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

$currentDate

The $currentDate operator sets the value of a field to the current date, either as a Date or a timestamp. The default type is Date.

The $currentDate operator has the form:

{ $currentDate: { <field1>: <typeSpecification1>, ... } }
<typeSpecification> can be either:

- a boolean true to set the field value to the current date as a Date, or
- a document { $type: "timestamp" } or { $type: "date" } which explicitly specifies the type. The operator is case-sensitive and accepts only the lowercase "timestamp" or the lowercase "date".

Práctica

> db.titulos.update({titulo: "El Quijote"},{$set: {autor: "Miguel De Cervantes Saavedra"}, $currentDate: {modifyAt: true}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

$inc

The $inc operator increments a field by a specified value and has the following form:

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

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
The $inc operator accepts positive and negative values.

If the field does not exist, $inc creates the field and sets the field to the specified value.

Use of the $inc operator on a field with a null value will generate an error.

Práctica

> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")}, {$inc: {prestados: -1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

$min

The $min updates the value of the field to a specified value if the specified value is less than the current value of the field. The $min operator can compare values of different types, using the BSON comparison order.

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

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
If the field does not exists, the $min operator sets the field to the specified value.

For comparisons between values of different types, such as a number and a null, $min uses the BSON comparison order.

Práctica

```
> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")}, {$min: {stock: 12}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")}, {$min: {stock: 8}}) // Solo actualiza si el nuevo valor es menor que el anterior
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

$max

The $max operator updates the value of the field to a specified value if the specified value is greater than the current value of the field. The $max operator can compare values of different types, using the BSON comparison order.

The $max operator expression has the form:

{ $max: { <field1>: <value1>, ... } }
To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
If the field does not exists, the $max operator sets the field to the specified value.

Práctica

```
> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")}, {$max: {stock: 10}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 }) // Sólo actualiza si el nuevo valor es mayor que el actual
> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")}, {$max: {stock: 5}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

$mul

Multiply the value of a field by a number. To specify a $mul expression, use the following prototype:

{ $mul: { <field1>: <number1>, ... } }
The field to update must contain a numeric value.

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
Missing Field

If the field does not exist in a document, $mul creates the field and sets the value to zero of the same numeric type as the multiplier.

Práctica

> db.titulos.update({ "_id" : ObjectId("5e281476e5cb58e060e229ff")}, {$mul: {stock: 2}})

$rename

The $rename operator updates the name of a field and has the following form:

{$rename: { <field1>: <newName1>, <field2>: <newName2>, ... } }

The new field name must differ from the existing field name. To specify a <field> in an embedded document, use dot notation.

Behavior

The $rename operator logically performs an $unset of both the old name and the new name, and then performs a $set operation with the new name. As such, the operation may not preserve the order of the fields in the document; i.e. the renamed field may move within the document.

If the document already has a field with the <newName>, the $rename operator removes that field and renames the specified <field> to <newName>.

If the field to rename does not exist in a document, $rename does nothing (i.e. no operation).

For fields in embedded documents, the $rename operator can rename these fields as well as move the fields in and out of embedded documents. $rename does not work if these fields are in array elements.

Práctica

> db.titulos.insert({title:"Mas Ruido que Nueces", autor: "William Shakespeare" })

> db.titulos.update({},{$rename: {"title":"titulo"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })

> db.titulos.update({},{$rename: {"title":"titulo"}},{multi: true})
WriteResult({ "nMatched" : 5, "nUpserted" : 0, "nModified" : 3 })

> db.titulos.update({titulo:"Mas Ruido que Nueces"},{$set: {categoria: {primaria: "novela", lengua: "inglés"}}})

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.titulos.update({},{$rename: {"categoria.lengua": "categoria.idioma"}},{multi:
true})
WriteResult({ "nMatched" : 5, "nUpserted" : 0, "nModified" : 1 })


> db.titulos.update({_id: ObjectId("5e281476e5cb58e060e229ff")},{$set: {editorial:
[{nombre:"Planeta", isbn: "e723575237"},{nombre:"Deusto", isbn: "dggj68768271"}]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.titulos.update({},{$rename: {"editorial.isbn":"editorial.ISBN"}},{multi: true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 0,
        "nModified" : 0,
        "writeError" : {
                "code" : 28,
                "errmsg" : "cannot use the part (editorial of editorial.isbn) to traverse
the element ({editorial: [ { nombre: \"Planeta\", isbn: \"e723575237\" }, { nombre:
\"Deusto\", isbn: \"dggj68768271\" } ]})"
        }
})
> db.titulos.update({},{$rename: {"editorial.0.isbn":"editorial.0.ISBN"}},{multi: true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 0,
        "nModified" : 0,
        "writeError" : {
                "code" : 2,
                "errmsg" : "The source field cannot be an array element,
'editorial.0.isbn' in doc with _id: ObjectId('5e281476e5cb58e060e229ff') has an
array field called 'editorial'"
        }
})
```

$ (update)

The positional $ operator identifies an element in an array to update without explicitly specifying the position of the element in the array.

The positional $ operator has the form:

{ "<array>.$" : value }

When used with update operations, e.g. db.collection.update() and db.collection.findAndModify(),

- the positional $ operator acts as a placeholder for the first element that matches the query document, and
- the array field must appear as part of the query document.

Behavior

upsert
Do not use the positional operator $ with upsert operations because inserts will use the $ as a field name in the inserted document.

Nested Arrays
The positional $ operator cannot be used for queries which traverse more than one array, such as queries that traverse arrays nested within other arrays, because the replacement for the $ placeholder is a single value

Unsets
When used with the $unset operator, the positional $ operator does not remove the matching element from the array but rather sets it to null.

Negations
If the query matches the array using a negation operator, such as $ne, $not, or $nin, then you cannot use the positional operator to update values from this array.

However, if the negated portion of the query is inside of an $elemMatch expression, then you can use the positional operator to update this field.

Práctica

```
> db.titulos.update({categorias: "novela"}, {$set: {"categorias.$": "Novela"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.titulos.update({"editorial.nombre":"Deusto"},{$set: {"editorial.$.isbn":
"5647r4537skdhdjh"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

$[]

The all positional operator $[] indicates that the update operator should modify all elements in the specified array field.

The $[] operator has the following form:

{ <update operator>: { "<array>.$[]" : value } }

Use in update operations, e.g. db.collection.update() and db.collection.findAndModify(), to modify all array elements for the document or documents that match the query condition. For example:

```
db.collection.updateMany(
  { <query conditions> },
  { <update operator>: { "<array>.$[]" : value } }
)
```

For an example, see Update All Elements in an Array.

Behavior
upsert
If an upsert operation results in an insert, the query must include an exact equality match on the array field in order to use the $[] positional operator in the update statement.

Práctica

```
> db.titulos.insert({autor: "John Grisham", titulo: "The Firm", recomendaciones: [4, 3, 5, 5, 2, 2, 5]})
WriteResult({ "nInserted" : 1 })

> db.titulos.update({titulo: "The Firm"}, {$set: {"recomendaciones.$[]": "5"  }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: "The Firm"})
{ "_id" : ObjectId("5e283632fe0a9f8b3c6230ab"), "autor" : "John Grisham", "titulo" : "The Firm", "recomendaciones" : [ "5", "5", "5", "5", "5", "5", "5" ] }
```

$[<identifier>]

The filtered positional operator $[<identifier>] identifies the array elements that match the arrayFilters conditions for an update operation, e.g. db.collection.update() and db.collection.findAndModify().

Used in conjunction with the arrayFilters option, the $[<identifier>] operator has the following form:

{ <update operator>: { "<array>.$[<identifier>]" : value } },
{ arrayFilters: [{ <identifier>: <condition> }] }

Práctica

> db.titulos.insert({autor:"John Grisham", titulo:"The Fitgerald Case", precios: [18, 12, 22, 21, 22, 12]})

> db.titulos.update({titulo:"The Fitgerald Case"},{$set:{"precios.$[elem]": 15}},{arrayFilters: [{"elem": {$lt: 15}}]})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

$addToSet

The $addToSet operator adds a value to an array unless the value is already present, in which case $addToSet does nothing to that array.

The $addToSet operator has the form:

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

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
$addToSet only ensures that there are no duplicate items added to the set and does not affect existing duplicate elements. $addToSet does not guarantee a particular ordering of elements in the modified set.

Missing Field
If you use $addToSet on a field that is absent in the document to update, $addToSet creates the array field with the specified value as its element.

Field is Not an Array
If you use $addToSet on a field that is not an array, the operation will fail.

Práctica

```
> db.titulos.update({titulo: /Cien/}, {$addToSet: {categorias: "Colombia"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.update({titulo: /Cien/}, {$addToSet: {categorias: "Colombia"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

$each

The $each modifier is available for use with the $addToSet operator and the $push operator.

Use with the $addToSet operator to add multiple values to an array <field> if the values do not exist in the <field>.

{ $addToSet: { <field>: { $each: [<value1>, <value2> ...] } } }

Práctica

```
> db.titulos.update({titulo: "The Firm"}, {$set: {categorias: ["novela"]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.update({titulo: "The Firm"}, {$addToSet: {categorias: {$each:
["USA","thriller"]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

$pop

The $pop operator removes the first or last element of an array. Pass $pop a value of -1 to remove the first element of an array and 1 to remove the last element in an array.

The $pop operator has the form:

{ $pop: { <field>: <-1 | 1>, ... } }

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
The $pop operation fails if the <field> is not an array.

If the $pop operator removes the last item in the <field>, the <field> will then hold an empty array.

Práctica

```
> db.titulos.update({titulo: "The Firm"}, {$pop: {categorias: -1}}) // Elimina el primero
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: "The Firm"})
{ "_id" : ObjectId("5e283632fe0a9f8b3c6230ab"), "autor" : "John Grisham", "titulo" :
"The Firm", "recomendaciones" : [ "5", "5", "5", "5", "5", "5", "5" ], "categorias" : [ "USA",
"thriller" ] }

> db.titulos.update({titulo: "The Firm"}, {$pop: {categorias: 1}}) // Elimina el último
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: "The Firm"})
{ "_id" : ObjectId("5e283632fe0a9f8b3c6230ab"), "autor" : "John Grisham", "titulo" :
"The Firm", "recomendaciones" : [ "5", "5", "5", "5", "5", "5", "5" ], "categorias" : [ "USA" ]
}
```

$pull

The $pull operator removes from an existing array all instances of a value or values that match a specified condition.

The $pull operator has the form:

{ $pull: { <field1>: <value|condition>, <field2>: <value|condition>, ... } }

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
If you specify a <condition> and the array elements are embedded documents, $pull operator applies the <condition> as if each array element were a document in a collection. See Remove Items from an Array of Documents for an example.

If the specified <value> to remove is an array, $pull removes only the elements in the array that match the specified <value> exactly, including order.

If the specified <value> to remove is a document, $pull removes only the elements in the array that have the exact same fields and values. The ordering of the fields can differ.

Práctica

```
> db.titulos.update({titulo: "The Firm"}, {$set: {categorias:
["USA","thriller","novela","abogados","USA"]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: "The Firm"})
{ "_id" : ObjectId("5e283632fe0a9f8b3c6230ab"), "autor" : "John Grisham", "titulo" :
"The Firm", "recomendaciones" : [ "5", "5", "5", "5", "5", "5", "5" ], "categorias" : [ "USA",
"thriller", "novela", "abogados", "USA" ] }

> db.titulos.update({titulo: "The Firm"}, {$pull: {categorias: "USA"}}) // Solo elimina el
primer elemento con la coincidencia
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.titulos.update({titulo: "The Firm"}, {$pull: {categorias: {$in:
["abogados","novela"]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

$pullAll

The $pullAll operator removes all instances of the specified values from an existing array. Unlike the $pull operator that removes elements by specifying a query, $pullAll removes elements that match the listed values.

The $pullAll operator has the form:

{ $pullAll: { <field1>: [<value1>, <value2> ...], ... } }

To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
If a <value> to remove is a document or an array, $pullAll removes only the elements in the array that match the specified <value> exactly, including order.

Práctica

```
> db.titulos.update({titulo: /Fitgerald/}, {$pullAll: {categorias: ["thriller","best
seller"]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

$push

The $push operator appends a specified value to an array.

The $push operator has the form:

{ $push: { <field1>: <value1>, ... } }
To specify a <field> in an embedded document or in an array, use dot notation.

Behavior
If the field is absent in the document to update, $push adds the array field with the value as its element.

If the field is not an array, the operation will fail.

If the value is an array, $push appends the whole array as a single element. To add each element of the value separately, use the $each modifier with $push. For an example, see Append Multiple Values to an Array. For a list of modifiers available for $push, see Modifiers.

Modifiers
You can use the $push operator with the following modifiers:

Modifier	Description
$each	Appends multiple values to the array field.
$slice	Limits the number of array elements. Requires the use of the $each modifier.
$sort	Orders elements of the array. Requires the use of the $each modifier.
$position	Specifies the location in the array at which to insert the new elements. Requires the use of the $each modifier. Without the $position modifier, the $push appends the elements to the end of the array.

When used with modifiers, the $push operator has the form:

{ $push: { <field1>: { <modifier1>: <value1>, ... }, ... } }

The processing of the push operation with modifiers occur in the following order, regardless of the order in which the modifiers appear:

- Update array to add elements in the correct position.
- Apply sort, if specified.
- Slice the array, if specified.
- Store the array.

Append a Value to an Array

Práctica

```
> db.titulos.update({titulo: /Fitgerald/}, {$push: {categorias: "USA"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: /Fitgerald/})
{ "_id" : ObjectId("5e283d56fe0a9f8b3c6230af"), "autor" : "John Grisham", "titulo" :
"The Fitgerald Case", "precios" : [ 18, 15, 22, 21, 22, 15 ], "categorias" : [ "USA" ] }
```

Append Multiple Values to an Array

Práctica

```
> db.titulos.update({titulo: /Fitgerald/}, {$push: {categorias: {$each:
["abogados","suspense"]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: /Fitgerald/})
{ "_id" : ObjectId("5e283d56fe0a9f8b3c6230af"), "autor" : "John Grisham", "titulo" :
"The Fitgerald Case", "precios" : [ 18, 15, 22, 21, 22, 15 ], "categorias" : [ "USA",
"abogados", "suspense" ] }
```

Use $push Operator with Multiple Modifiers

Práctica

```
> db.titulos.update({titulo: /Fitgerald/}, {$push: {categorias: {$each: ["thriller","best
seller"], $sort: -1, $slice: -3}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.titulos.find({titulo: /Fitgerald/})
{ "_id" : ObjectId("5e283d56fe0a9f8b3c6230af"), "autor" : "John Grisham", "titulo" :
"The Fitgerald Case", "precios" : [ 18, 15, 22, 21, 22, 15 ], "categorias" : [ "best seller",
"abogados", "USA" ] }
```

Use $push Operator with position Modifier

Práctica

```
> db.titulos.find({titulo: /Fitgerald/})
{ "_id" : ObjectId("5e283d56fe0a9f8b3c6230af"), "autor" : "John Grisham", "titulo" : "The Fitgerald Case", "precios" : [ 18, 15, 22, 21, 22, 15 ], "categorias" : [ "abogados", "tapa dura", "ebook", "USA" ] }
```

Método updateOne()

https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne

Updates a single document within the collection based on the filter.

Sintaxis

```
db.collection.updateOne(
   <filter>,
   <update>,
   {
     upsert: <boolean>,
     writeConcern: <document>,
     collation: <document>,
     arrayFilters: [ <filterdocument1>, ... ],
     hint:  <document|string>        // Available starting in MongoDB 4.2.1
   }
)
```

Método updateMany()

https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/#db.collection.updateOne

Updates all documents that match the specified filter for a collection.

Sintaxis

```
db.collection.updateMany(
  <filter>,
  <update>,
  {
    upsert: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    arrayFilters: [ <filterdocument1>, ... ],
    hint:  <document|string>        // Available starting in MongoDB 4.2.1
  }
)
```

Método replaceOne()

https://docs.mongodb.com/manual/reference/method/db.collection.replaceOne/

Replaces a single document within the collection based on the filter.

Sintaxis

```
db.collection.replaceOne(
  <filter>,
  <replacement>,
  {
    upsert: <boolean>,
    writeConcern: <document>,
    collation: <document>,
    hint: <document|string>                // Available starting in 4.2.1
  }
)
```

Additional Methods

The following methods can also update documents from a collection:

db.collection.findOneAndReplace().
db.collection.findOneAndUpdate().
db.collection.findAndModify().
db.collection.save().
db.collection.bulkWrite().

Collation