Curso Intro Big Data-MongoDB

MongoDB – Arrays - Operadores $ Adicionales - Save - FindAndModify

DBlandIT®

Operaciones sobre Arrays

y

Operadores $ adicionales

DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $push

El operador $push nos permite insertar elementos repetidos en un array. Si el array no existe lo crea.

Dado el siguiente documento:

```
> db.updtst.find({_id:100})
{ "_id" : 100, "x" : 100, "y" : 100, "z" : 1000 }
```

Ejemplo 1 – Agregar un elemento a un array inexistente, creando dicho array

```
db.updtst.update({_id:100},{$push : {items:"elemArray"}})
```

Este comando crea un array ítems con el valor "elemArray"

```
> db.updtst.update({_id:100},{$push : {items:"elemArray"}})
> db.updtst.find({_id:100})
{ "_id" : 100, "items" : [  "elemArray" ], "x" : 100, "y" : 100, "z" : 1000 }
>
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $push (Cont.)

Ejemplo 2 – Agregar varios elementos a un array existente con igual valor

```
db.updtst.update({_id:100},{$push : {items:"elemArray"}})
db.updtst.update({_id:100},{$push : {items:"elemArray"}})
db.updtst.update({_id:100},{$push : {items:"elemArray"}})
```

Agrega nuevos valores al array ítems repitiendo el elemento "elemarray".

```
> db.updtst.insert({_id:100,x:10,y:100,z:1000})
> db.updtst.update({_id:100},{$push : {items:"elemArray"}})
> db.updtst.update({_id:100},{$push : {items:"elemArray"}})
> db.updtst.update({_id:100},{$push : {items:"elemArray"}})
> db.updtst.find({_id:100})
{ "_id" : 100, "items" : [  "elemArray",  "elemArray",  "elemArray" ], "x" : 10,
 "y" : 100, "z" : 1000 }
```

Ejemplo 3 – Agregar un documento a un array

```
db.updtst.update({_id:100},{$push:{items:{producto:"PRU",cantidad:10}}})
```

Agrega un objeto dentro del array ítems

```
> db.updtst.update({_id:100},{$push : {items:{producto:"PRU",cantidad:10}}})
> db.updtst.find({_id:100})
{ "_id" : 100, "items" : [      "elemArray",    "elemArray",    "elemArray",
{       "producto" : "PRU",     "cantidad" : 10 } ], "x" : 10, "y" : 100, "z" :
1000 }
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $addToSet

El operador $push nos permite insertar elementos únicos en un array. Si el array no existe lo crea.

Dado el siguiente documento:

```
> db.updtst.find({_id:200})
{ "_id" : 200, "x" : 10, "y" : 100, "z" : 1000 }
```

Ejemplo 1 – Agregar un elemento a un array inexistente, creando dicho array

```
db.updtst.update({_id:100},{$addToSet : {items:"elemArray"}})
```

Este comando crea un array ítems con el valor “elemArray”

```
> db.updtst.update({_id:200},{$addToSet: {items:"elemArray"}})
> db.updtst.find({_id:200})
{ "_id" : 200, "items" : [  "elemArray" ], "x" : 10, "y" : 100, "z" : 1000 }
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $addToSet (Cont.)

Ejemplo 2 – Agregar varios elementos a un array existente con igual valor

```
db.updtst.update({_id:100},{$addToSet : {items:"elemArray"}})
db.updtst.update({_id:100},{$addToSet : {items:"elemArray"}})
db.updtst.update({_id:100},{$addToSet : {items:"elemArray"}})
```

Agrega nuevos valores al array ítems sólo con valores únicos el elemento “elemarray”.

```
{ "_id" : 200, "items" : [  "elemArray" ], "x" : 10, "y" : 100, "z" : 1000 }
> db.updtst.update({_id:200},{$addToSet: {items:"elemArray"}})
> db.updtst.update({_id:200},{$addToSet: {items:"elemArray"}})
> db.updtst.update({_id:200},{$addToSet: {items:"elemArray"}})
> db.updtst.find({_id:200})
{ "_id" : 200, "items" : [  "elemArray" ], "x" : 10, "y" : 100, "z" : 1000 }
>
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $pop

El operador $pop nos permite eliminar el primero ó último elemento de un array.

Dado el siguiente documento:

```
> db.updtst.find({_id:300})
{ "_id" : 300, "x" : 1, "y" : 200, "items" : [  1,  2,  3,  6,  8,  10,  11 ] }
>
```

Ejemplo 1 – Eliminación de último elemento del array

```
db.updtst.update({ _id:300 }, { $pop: { items: 1 }})
```

```
> db.updtst.update({ _id:300 }, { $pop: { items: 1 }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  1,  2,  3,  6,  8,  10 ], "x" : 1, "y" : 200 }
>
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $pop (Cont.)

Dado el siguiente documento:

```
> db.updtst.update({ _id:300 }, { $pop: { items: 1 }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  1,  2,  3,  6,  8,  10 ], "x" : 1, "y" : 200 }
>
```

Ejemplo 2 – - Eliminación de primer elemento del array

```
db.updtst.update({ _id:300 }, { $pop: { items: -1 }})
```

```
>
> db.updtst.update({ _id:300 }, { $pop: { items: -1 }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  2,  3,  6,  8,  10 ], "x" : 1, "y" : 200 }
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $pull

El operador $pull nos permite eliminar elementos de un array cuyo valor coincida con el indicado.
Elimina todos los elementos del array que cumplan con la condición.

Dado el siguiente documento:

```
>
> db.updtst.update({ _id:300 }, { $pop: { items: -1 }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  2,  3,  6,  8,  10 ], "x" : 1, "y" : 200 }
```

Ejemplo 1 – Eliminación del elemento del array con valor 6

```
db.updtst.update({ _id:300 }, { $pull: { items: 6 }})
```

```
>
> db.updtst.update({ _id:300 }, { $pull: { items: 6 }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  2,  3,  8,  10 ], "x" : 1, "y" : 200 }
>
```


DBlandIT®

Modificando Documentos Parciales

Update Parciales

Operador $pullAll

El operador $pullAll sirve para eliminar elemento/s de un array cuyo valor coincida con alguno de los valores de la lista indicados:

Dado el siguiente documento:

```
>
> db.updtst.update({ _id:300 }, { $pull: { items: 6 }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  2,  3,  8,  10 ], "x" : 1, "y" : 200 }
> _
```

Ejemplo 1 – - Eliminación de los elemento del array items con los valores 2 u 8.

```
db.updtst.update({ _id:300 }, { $pullAll: { items: [2, 8] } })
```

```
>
> db.updtst.update({ _id:300 }, { $pullAll: { items: [2, 8] }})
> db.updtst.find({_id:300})
{ "_id" : 300, "items" : [  3,  10 ], "x" : 1, "y" : 200 }
>
```


DBlandIT®

Consultar elementos de un array de documentos

$elemMatch - Introducción

Si quisiera consultar las facturas en las que se haya comprado dos (2) productos “CORREA 10mm”, se podría creer que la consulta correcta sería la siguiente:

```
db.facturas.find({"item.producto":"CORREA 10mm", "item.cantidad":2})
```

Pero esto incluirá los arrays que contengan estos 2 valores, **aunque estén en subdocumentos diferentes**.

```
> db.facturas.find({"item.producto":"CORREA 10mm","item.cantidad":2},
... {"item.cantidad":1,"item.producto":1,_id:0}).limit(2).pretty()
{ "item" : [ { "producto" : "CORREA 10mm", "cantidad" : 2 } ] }
{
        "item" : [
                {
                        "cantidad" : 2,
                        "producto" : "TUERCA 2mm"
                },
                {
                        "cantidad" : 1,
                        "producto" : "CORREA 10mm"
                }
        ]
}
```


DBlandIT®

Consultar elementos de un array de documentos

$elemMatch

El operador $elemMatch nos permite verificar que se cumple cada condición en algún subdocumento.

```
db.facturas.find({item:{$elemMatch:{producto:"CORREA 10mm" ,cantidad:2}}})
```

```
> db.facturas.find({item:{$elemMatch:{producto:"CORREA 10mm",cantidad:2}}},
... {"item.cantidad":1,"item.producto":1,_id:0}).limit(2).pretty()
{ "item" : [ { "producto" : "CORREA 10mm", "cantidad" : 2 } ] }
{ "item" : [ { "cantidad" : 2, "producto" : "CORREA 10mm" } ] }
>
```


DBlandIT®

Consultar elementos de un array de documentos

$ (proyección)

El operador $ nos permite hacer la proyección del primer elemento del array que cumpla con la condición especificada en el where.

```
db.facturas.find({item:{$elemMatch:{producto:"SET HERRAMIENTAS",cantidad:1}},
                  nroFactura:1004}, {"item.$":1,nroFactura:1,_id:0})
```

```
> db.facturas.find({item:{$elemMatch:{producto:"SET HERRAMIENTAS",cantidad:1}},n
roFactura:1004}, {"item.$":1,nroFactura:1,_id:0})
{ "item" : [ { "cantidad" : 1, "precio" : 700, "producto" : "SET HERRAMIENTAS" }
 ], "nroFactura" : 1004 }
>
```


DBlandIT®

Modificar elementos de un array de documentos

$ (update)

El operador $ nos permite modificar el primer elemento del array que cumpla con la condición especificada en el where.
Es decir, el mismo que es mostrado al hacer la proyección.

```
db.facturas.update({item:{$elemMatch:{producto:"SET HERRAMIENTAS",cantidad:1}},
                    nroFactura:1004},
                   {$set:{"item.$.precio":500}})
```

```
> db.facturas.update({item:{$elemMatch:{producto:"SET HERRAMIENTAS",cantidad:1}}
,nroFactura:1004}, {$set:{"item.$.precio":500}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

```
> db.facturas.find({item:{$elemMatch:{producto:"SET HERRAMIENTAS",cantidad:1}},n
roFactura:1004}, {"item.$":1,nroFactura:1,_id:0})
{ "item" : [ { "cantidad" : 1, "precio" : 500, "producto" : "SET HERRAMIENTAS" }
 ], "nroFactura" : 1004 }
>
```


DBlandIT®

Modificando Arrays

$each

Este operador permite hacer operaciones multiples sobre arrays.

Por ejemplo, lograr un comportamiento similar al $pushAll (cláusula deprecada) para ser ejecutado con el operador el **$addToSet**:

```
db.array.update({},{$addToSet: {puntajes:{$each:[2,5,7]}}})
```

El **$each** también puede usarse con el **$push**, y a su vez combinarlo con otros operadores: **$slice** y **$sort**

```
> db.array.insert({puntajes:[1]})
WriteResult({ "nInserted" : 1 })
> db.array.update({},{$addToSet: {puntajes:{$each:[2,5,7]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.array.find()
{ "_id" : ObjectId("561ee7ce788c0ec79885eda2"), "puntajes" : [ 1, 2, 5, 7 ] }
>
```


DBlandIT®

Modificando Arrays

$each (continuación)

$slice: indica el tamaño que deberá tener el array

```
db.array.update({},{$push:{puntajes:{$each:[1,3,9,20],$slice:5}}})
```

```
> db.array.find({},{_id:0})
{ "puntajes" : [ 2, 5, 7 ] }
>
> db.array.update({},{$push:{puntajes:{$each:[1,3,9,20],$slice:5}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.array.find({},{_id:0})
{ "puntajes" : [ 2, 5, 7, 1, 3 ] }
>
```


DBlandIT®

Modificando Arrays

$each (continuación)

$sort: permite guardar un array manteniéndolo ordenado.

Por ejemplo, mantener un array de parciales ordenados por nota.

```
db.array.update({},{$push:{parciales:{$each:[{id:1,nota:8},{id:2,nota:7},{id:3,nota:6}]
,$sort:{nota:1}}}})
```

```
> db.alumnos.update({},{$push:{parciales:{
... $each: [{id:1,nota:8},{id:2,nota:7},{id:3,nota:6}]
... ,$sort:{nota:1}}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.alumnos.findOne({},{_id:0})
{
        "parciales" : [
                {
                        "id" : 3,
                        "nota" : 6
                },
                {
                        "id" : 2,
                        "nota" : 7
                },
                {
                        "id" : 1,
                        "nota" : 8
                }
        ]
}
```

DBlandIT®

Modificando Arrays

$each (continuación)

$sort: permite guardar un array manteniéndolo ordenado con solo 5 elementos.

Por ejemplo, mantener un array de parciales ordenados por nota.

```
db.array.update({},{$push:{parciales:{$each:[{id:1,nota:8},{id:2,nota:7},{id:3,nota:6}] ,$slice:5
,$sort:{nota:-1}}}})
```

```
> db.alumnos.update({},{$push:{parciales:{
... $each: [{id:1,nota:8},{id:2,nota:7},{id:3,nota:6}]
... ,$sort:{nota:1}}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.alumnos.findOne({},{_id:0})
{
        "parciales" : [
                {
                        "id" : 3,
                        "nota" : 6
                },
                {
                        "id" : 2,
                        "nota" : 7
                },
                {
                        "id" : 1,
                        "nota" : 8
                }
        ]
}
```

DBlandIT®

Upsert

$setOnInsert

Este operador se utiliza en el update usando **upsert: true.**

Permite asignar un valor cuando se inserta un documento y no tendrá efecto si lo está modificando.

```
db.paginas.update({url:"www.dblandit.com"},
                  {$inc:{visitas:1},$setOnInsert:{creado:ISODate()}},
                  {upsert:true})
```

```
> db.paginas.update({url:"www.dblandit.com"},{$inc:{visitas:1},$setOnInsert:{cre
ado:ISODate()}},{upsert:true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("56246b757b1d7fcfc9eae525")
})
```


DBlandIT®

Upsert

$setOnInsert (continuación)

Consultamos el documento creado.

```
> db.paginas.findOne({url:"www.dblandit.com"})
{
        "_id" : ObjectId("56246b757b1d7fcfc9eae525"),
        "url" : "www.dblandit.com",
        "visitas" : 1,
        "creado" : ISODate("2015-10-19T04:03:01.078Z")
}
```

Al repetir la operación y volver a consultar, vemos que la key “creado” no se modifica.

```
> db.paginas.update({url:"www.dblandit.com"},{$inc:{visitas:1},$setOnInsert:{cre
ado:ISODate()}},{upsert:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.paginas.update({url:"www.dblandit.com"},{$inc:{visitas:1},$setOnInsert:{cre
ado:ISODate()}},{upsert:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.paginas.findOne({url:"www.dblandit.com"})
{
        "_id" : ObjectId("56246b757b1d7fcfc9eae525"),
        "url" : "www.dblandit.com",
        "visitas" : 3,
        "creado" : ISODate("2015-10-19T04:03:01.078Z")
}
```


DBlandIT®

Consultando tipos de datos

$type

Utilizando los números de la tabla, se puede consultar documentos por el tipo de dato de sus atributos.

En este ejemplo obtendremos los documentos con tipo de dato “Date”

```
db.tipos.find({x:{$type:9}},{_id:0})
```

```
db.tipos.find({x:{$type:"date"}},{_id:0})
```

```
> db.tipos.find({},{_id:0})
{ "x" : [ ] }
{ "x" : 1 }
{ "x" : ISODate("2015-10-19T04:19:15.911Z") }
{ "x" : ISODate("2015-10-19T04:19:19.380Z") }
{ "x" : true }
>
> db.tipos.find({x:{$type:9}},{_id:0})
{ "x" : ISODate("2015-10-19T04:19:15.911Z") }
{ "x" : ISODate("2015-10-19T04:19:19.380Z") }
```

Tipo	Número	Alias (3.2)
Double	1	“double”
String	2	“string”
Object	3	“object”
Array	4	“array”
Binary data	5	“binData”
Object Id	7	“objectId”
Boolean	8	“bool”
Date	9	“date”
Null	10	“null”

Esto nos puede servir cuando la estructura de nuestro documento es muy cambiante.

DBlandIT®

save

El método save() actualiza un documento existente o inserta un documento dependiendo del documento indicado.

```
db.coleccion.save(<documento_a_escribir>)
```

- Si el documento no contiene el atributo _id, el save insertará un nuevo documento con un _id autogenerado.
- Si el documento posee el atributo _id, el save realizará un update con “upsert”, es decir, si existe un documento con igual _id lo reemplaza, y si no existe un documento con igual _id lo inserta.

DBlandIT®

save

```
> db.test.save
function ( obj , opts ){
    if ( obj == null )
        throw "can't save a null";

    if ( typeof( obj ) == "number" || typeof( obj) == "string" )
        throw "can't save a number or string"

    if ( typeof( obj._id ) == "undefined" ){
        obj._id = new ObjectId();
        return this.insert( obj , opts );
    }
    else {
        return this.update( { _id : obj._id } , obj , Object.merge({ upsert:true
 }, opts));
    }
}
```


DBlandIT®

save

Sintaxis

```
db.collection_name.save( documento )
```

Ejemplo 1 – Se pasa como atributo un documento sin _id, el save insertará un documento con un _id autogenerado.

```
db.facturas.save( { nroFactura: 23434 } )
```

```
> db.facturas.save({nroFactura:23434})
WriteResult({ "nInserted" : 1 })
> db.facturas.find({nroFactura:23434})
{ "_id" : ObjectId("546cbbe01c50613c3ca5a1ed"), "nroFactura" : 23434 }
```


DBlandIT®

save

Ejemplo 2 – Se pasa como atributo un documento con _id inexistente, el save insertará un documento con ese _id.

```
db.facturas.save( {_id:10001, nroFactura: 23435 } )
```

```
> db.facturas.save( {_id:10001, nroFactura: 23435 } )
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 10001 })

>
> db.facturas.find( {_id:10001, nroFactura: 23435 } )
{ "_id" : 10001, "nroFactura" : 23435 }
```


DBlandIT®

save

Ejemplo 3 – Se pasa como atributo un documento con _id existente, el save reemplazará el documento existente con ese _id por el documento pasado por parámetro.

```
db.facturas.save({_id:10001, nroFactura: 23435, condPago:"CONTADO"})
```

```
> db.facturas.save( {_id:10001, nroFactura: 23435, condPago:"CONTADO" } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.facturas.find( {_id:10001, nroFactura: 23435, condPago:"CONTADO" } )
{ "_id" : 10001, "nroFactura" : 23435, "condPago" : "CONTADO" }
>
```


DBlandIT®

findAndModify

Modifica y retorna un único documento. Por omisión, el documento retornado no incluye las modificaciones realizadas por el update.

Para retornar el documento con las modificaciones en el update hay que utilizar la opción new.

Este método es una ayuda en el Shell para ejecutar el comando findAndModify.

```
> db.facturas.findAndModify
function (args){
    var cmd = { findandmodify: this.getName() };
    for (var key in args){
        cmd[key] = args[key];
    }

    var ret = this._db.runCommand( cmd );
    if ( ! ret.ok ){
        if (ret.errmsg == "No matching object found"){
            return null;
        }
        throw "findAndModifyFailed failed: " + tojson( ret );
    }
    return ret.value;
}
```


DBlandIT®

findAndModify

Sintaxis

```
db.collection.findAndModify({
        query: <document>,
        sort: <document>,
        remove: <boolean>,
        update: <document>,
        new: <boolean>,
        fields: <document>,
        upsert: <boolean>});
```


DBlandIT®

findAndModify

Sintaxis

```
db.collection.findAndModify({
    query: <document>,
    sort: <document>,
    remove: <boolean>,
    update: <document>,
     new: <boolean>,
    fields: <document>
    upsert: <boolean>});
```

--Opcional, criterio de selección para la modificación

--Opcional, determina el documento a modificar, si el query devolvió muchos docs.

--Configurar en true para borrar el documento seleccionado, El default es false.

--Utiliza los mismos operadores del update() y especificación de valores a modificar en el documento seleccionado.

--Opcional. Cuando es true, retorna el document modificado en lugar del original.

--Opcional. Un subconjunto de atributos a retornar, {<field1>: 1, <field2>: 1, ... }.

--Optional. Usado en conjución con el atributo update. El default es false.

Los atributos remove y update se deben poner uno u otro.

DBlandIT®

findAndModify

```
> var doc = db.facturas.findAndModify({
... query: { "cliente.apellido": "Lavagno" },
... sort: { nroFactura: -1 },
... update: { $inc: { nroFactura: 1 } },
... upsert: true
... })
>
> doc
{
        "_id" : ObjectId("55e4a70fbfc68c676a0488f5"),
        "cliente" : {
                "apellido" : "Lavagno",
                "cuit" : 2729887543,
                "nombre" : "Soledad",
                "region" : "NOA"
        },
        "condPago" : "30 Ds FF",
        "fechaEmision" : ISODate("2014-02-24T00:00:00Z"),
        "fechaVencimiento" : ISODate("2014-03-26T00:00:00Z"),
        "item" : [
                {
                        "cantidad" : 1,
                        "precio" : 700,
                        "producto" : "SET HERRAMIENTAS"
                }
        ],
        "nroFactura" : 31866
}
> db.facturas.findOne({_id:doc._id},{nroFactura:1,_id:0})
{ "nroFactura" : 31867 }
```

Ejemplo 1 – Consulta las facturas del cliente “Lavagno”, las ordena descendentes y a la primera le suma 1 al número de factura.

Muestra el documento anterior a la modificación. Si no existe crea un documento.

```
db.facturas.findAndModify({
query: { "cliente.apellido": "Lavagno" },
sort: { nroFactura: -1 },
update: { $inc: { nroFactura: 1 } },
upsert: true
})
```


DBlandIT®

findAndModify

```
> db.facturas.findAndModify({
... query: { "cliente.apellido": "Lavagno" },
... sort: { nroFactura: -1 },
... update: { $inc: { nroFactura: 1 } },
... new: true,
... upsert: true
... })
{
        "_id" : ObjectId("55e4a70fbfc68c676a0488f5"),
        "cliente" : {
                "apellido" : "Lavagno",
                "cuit" : 2729887543,
                "nombre" : "Soledad",
                "region" : "NOA"
        },
        "condPago" : "30 Ds FF",
        "fechaEmision" : ISODate("2014-02-24T00:00:00Z"),
        "fechaVencimiento" : ISODate("2014-03-26T00:00:00Z"),
        "item" : [
                {
                        "cantidad" : 1,
                        "precio" : 700,
                        "producto" : "SET HERRAMIENTAS"
                }
        ],
        "nroFactura" : 31868
}
```

Ejemplo 2 – Consulta las facturas del cliente "Lavagno", las ordena descendentes y a la primera le suma 1 al número de factura.

Muestra el documento modificado. Si no existe crea un documento.

```
db.facturas.findAndModify({
query: { "cliente.apellido": "Lavagno" },
sort: { nroFactura: -1 },
update: { $inc: { nroFactura: 1 } },
new: true,
upsert: true
})
```


DBlandIT®

findAndModify

Ejemplo 3 – Consulta las facturas del cliente “Lavagno” y condición de pago “CONTADO”, las ordena descendentes y a la primera le suma 1 al número de factura. Como no existen documentos no realiza operación y la salida del comando es null.

```
db.facturas.findAndModify({
        query: { "cliente.apellido": "Lavagno", condPago:”CONTADO” },
        sort: { nroFactura: -1 },
        update: { $inc: { nroFactura: 1 } },
        new: true
})
```

```
> db.facturas.findAndModify({
... query: { "cliente.apellido": "Lavagno", condPago: "CONTADO" },
... sort: { nroFactura: -1 },
... update: { $inc: { nroFactura: 1 } },
... new: true
... })
null
```


DBlandIT®

findAndModify vs. Update

Comparación del método findAndModify con el método update

- **Por default ambos métodos modifican un único documento.** Aunque el **método update** cuenta con la opción **multi**, para actualizar múltiples documentos.
- **Si** el criterio de selección del **findAndModify() retorna múltiples documentos**, se puede a través del **sort** tener **algún control sobre cuál** documento se va a **actualizar**. Con el comportamiento default del método **update()**, **no se puede** especificar **que documento único** se va a **actualizar** cuando concuerdan múltiples documentos con el criterio de selección.
- Por default, **findAndModifiy() retorna la versión del documento previa** a la modificación. Para obtener la versión del documento **modificada**, se debe utilizar la **opción new**. El **update() retorna** el objeto **WriteResult** que contiene el status de la operación. Para poder obtener el documento modificado se deberá utilizar el find(). Sin embargo si múltiples documentos concordaron con el criterio de búsqueda se deberá realizar un proceso particular de recuperación para obtener la información del documento modificado.
- **No se puede** especificar un **Write Concern para el findAndModify()** para cambiar el default. A partir de MongoDB 2.6, **se puede especificar un write concern para el update()**.

DBlandIT®