

Bases de No Datos Relacionales

- Operaciones CRUD II

UPDATE / DELETE

DELETE

¿Cuáles son los comandos que consideran nos permiten Eliminar documentos en una base de MongoDB?

- remove
- deleteOne
- deleteMany
- bulkWrite



DELETE

Eliminar documentos utilizando **remove** (comando de la base de datos)

De forma predeterminada, `remove()` quita todos los documentos que coinciden con la expresión de consulta.

Especificando la opción `justOne` (valor booleano en `true`) se limita la operación a la eliminación de un solo documento.

```
db.collection.remove({query},<justOne>)
```

Por ejemplo:

`db.coleccion.remove({"x":2},true) →` eliminará solo el primer documento que coincida con el criterio de filtro

`db.coleccion.remove({"x":2},false) ó`
`db.coleccion.remove({"x":2})` eliminarán todos los documentos que coincidan con el criterio de filtro



DELETE

ATENCIÓN:

`db.coleccion.remove({})` **NO es equivalente a** `db.coleccion.drop()`

Si bien ambas eliminan todos los documentos de la colección, `drop` eliminará además los índices de la misma, por lo tanto es más eficiente utilizar `drop()` si lo que quiero es realmente eliminar la colección.



DELETE

Eliminar documentos utilizando **deleteOne**

Elimina el primer documento que coincida con el filtro.

Consejo :

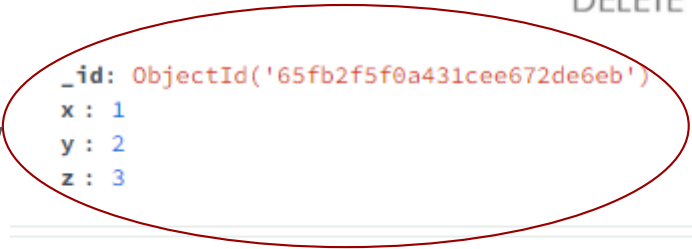
Se debería utilizar un campo que forme parte de un índice único, como `_id`, para eliminaciones precisas.

```
db.coleccion.deleteOne( { "_id" : ObjectId("563237a41a4d68582c2509da") } )
```

Atención:

Podría pasarle una expresión de filtro también y que existan varios documentos que la cumplan, pero SOLO se eliminará el primero.

```
db.coleccion.deleteOne({"x":1,"y":2})
```



```
_id: ObjectId('65fb2f5f0a431cee672de6eb')  
x : 1  
y : 2  
z : 3
```

```
_id: ObjectId('65fb30530a431cee672de6ec')  
x : 1  
y : 2
```



DELETE

Eliminar documentos utilizando **deleteMany**

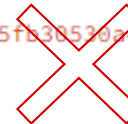
Elimina todos los documentos que coincidan con la expresión de filtro.

```
db.coleccion.deleteMany( {"x":1,"y":2} )
```

```
_id: ObjectId('65fb2f5f0a431cee672de6eb')  
x : 1  
y : 2  
z : 3
```



```
_id: ObjectId('65fb38530a431cee672de6ec')  
x : 1  
y : 2
```



¿Qué ocurrirá si le paso un documento vacío? Es decir `db.coleccion.deleteMany({})`

UPDATE

¿Cuáles son los comandos que consideramos nos permiten Modificar documentos en una base de MongoDB?

- update
- updateOne
- updateMany
- bulkWrite



UPDATE

Modificar documentos utilizando **update** (comando de la base de datos)

Modifica uno o varios documentos existentes en una colección. Puede modificar atributos específicos de uno o varios documentos existentes o reemplazar un documento por completo.

db.coleccion.update ({query}, {update}, { upsert, multi, writeConcern})

opciones extras!!!

- **upsert** (true o false): En true significa que realizará un **update** si existe un documento que concuerde con el criterio, o un **insert** si no existe algún documento que concuerde con el criterio. Por default es false (es decir no realiza un insert cuando no existe un documento que concuerde con el criterio)
- **multi** (true o false): Configurado en true, el update realiza la actualización de múltiples documentos que concuerdan con el criterio {query}. Si es, modifica solo un documento. Por default es false.
- **writeConcern**: Relacionado con el consentimiento exigido sobre la escritura en cantidad de nodos, de tiempo o escritura en journal (similar a un archivo de log de operaciones)



UPDATE

Modificar documentos utilizando **updateOne**

Modifica un único documento en una colección, el primero que coincida con el criterio de filtro. Puede modificar atributos específicos o reemplazar un documento por completo.

db.coleccion.updateOne ({query}, {update}, {upsert, writeConcern} ...)

- **upsert** (true o false): En true significa que realizará un **update** si existe un documento que concuerde con el criterio, o un **insert** si no existe algún documento que concuerde con el criterio. Por default es false (es decir no realiza un insert cuando no existe un documento que concuerde con el criterio)
- **writeConcern**: Relacionado con el consentimiento exigido sobre la escritura en cantidad de nodos, de tiempo o escritura en journal (similar a un archivo de log de operaciones)



UPDATE

Modificar documentos utilizando **updateMany**

Actualiza todos los documentos que coinciden con el filtro especificado para una colección. Puede modificar atributos específicos o reemplazar un documento por completo.

db.coleccion.updateOne ({query}, {update}, {upsert, writeConcern} ...)

- **upsert** (true o false): En true significa que realizará un **update** si existen documentos que concuerde con el criterio, o un **insert** si no existe algún documento que concuerde con el criterio. Por default es false (es decir no realiza un insert cuando no existen documentos que concuerde con el criterio)
- **writeConcern**: Relacionado con el consentimiento exigido sobre la escritura en cantidad de nodos, de tiempo o escritura en journal (similar a un archivo de log de operaciones)



UPDATE

Modificar documentos utilizando **updateMany**

Modifica un único documento en una colección, el primero que coincida con el criterio de filtro. Puede modificar atributos específicos o reemplazar un documento por completo.

db.coleccion.updateMany ({query}, {update}, {upsert, writeConcern})

- **upsert** (true o false): En true significa que realizará un **update** si existe un documento que concuerde con el criterio, o un **insert** si no existe algún documento que concuerde con el criterio. Por default es false (es decir no realiza un insert cuando no existe un documento que concuerde con el criterio)
- **writeConcern**: Relacionado con el consentimiento exigido sobre la escritura en cantidad de nodos, de tiempo o escritura en journal (similar a un archivo de log de operaciones)



UPDATE

Revisemos los parámetros

- **{query}** : Para construir el filtro o query (documento) están disponibles los mismos selectores/operadores de consulta que en el método **find()**.

Ejemplos:

```
db.pizzas.udpate({$and:[{ price:{$ne: 7}},{size:"small"}]},...)
db.pizzas.udpateOne({$and:[{ price:{$ne: 7}},{size:"small"}]},...)
db.pizzas.udpateMany({$and:[{ price:{$ne: 7}},{size:"small"}]},...)
```

{query}

- **{update}** : Es nuevamente un documento que contiene expresiones de **operadores** de actualizaciones a realizar. La forma del documento deberá ser la siguiente:

```
{
  <operador1>: { <atributo1>: <valor1>, ... },
  <operador2>: { <atributo2>: <valor2>, ... },
  ...
}
```

Ahora...¿cuáles pueden ser esos operadores?



UPDATE

- **{update}** : Operadores

Operador	Acción
\$currentDate	Establece el valor de un campo en la fecha actual, ya sea como fecha o marca de tiempo.
\$inc	Incrementa el valor del campo en la cantidad especificada.
\$min	Solo actualiza el campo si el valor especificado es menor que el valor del campo existente.
\$max	Solo actualiza el campo si el valor especificado es mayor que el valor del campo existente.
\$mul	Multiplica el valor del campo por la cantidad especificada.
\$rename	Cambia el nombre de un campo.
\$set	Establece el valor de un campo en un documento.
\$setOnInsert	Establece el valor de un campo si una actualización da como resultado una inserción de un documento.
\$unset	Elimina el campo especificado de un documento.



UPDATE

Entonces veamos algunos ejemplos de operadores:

```
db.pizzas.udpate({query},{ $set: {size : "medium"}})
```

```
db.pizzas.udpateOne(({query},{ $inc: {price : 3}}))
```

Puedo combinarlos...

```
db.pizzas.udpateMany(({query},{ $set: {size : "medium"}, $inc: {price : 3}}))
```

Todo junto... aplico un filtro y actualizo datos

```
db.pizzas.update({_id:0},{ $set: {size: "small"}})
```

```
db.pizzas.updateOne({_id:1},{ $set: {size: "medium"}, $inc: { price: 2 }})
```

```
db.pizzas.updateMany({size:"small"},{ $inc: { price: 2 }})
```

```
db.pizzas.update({_id:7},{ $unset: { spice: "" }})
```

¿Qué ocurre si el **_id:8** no existe?

```
db.pizzas.update({_id:8},{ $set: { size: "small" }},{upsert:true})
```



UPDATE

Un método más para modificar : **replaceOne**

Modifica un único documento en una colección. Reemplazando el documento completo que coincida con el filtro pasado como parámetro.

db.coleccion.replaceOne ({query}, {document}, {upsert, writeConcern...})

- **upsert** (true o false): En true significa que realizará un **update** si existe un documento que concuerde con el criterio, o un **insert** si no existe algún documento que concuerde con el criterio. Por default es false (es decir no realiza un insert cuando no existe un documento que concuerde con el criterio)
- **writeConcern**: Relacionado con el consentimiento exigido sobre la escritura en cantidad de nodos, de tiempo o escritura en journal (similar a un archivo de log de operaciones)



UPDATE

Un ejemplo ...

```
db.pizzas.replaceOne({_id:5},{type:"potatos",size:"small"})
```

No puedo utilizar aquí los operadores, simplemente debo enviar el documento que quiero que aparezca en mi colección.

Ejemplo con upsert: (agrega el documento si no hay coincidencia con el filtro)

```
db.pizzas.replaceOne({_id:9},{type:"fish",size:"small"},{upsert:true})
```

Si le paso como filtro un documento vacío {}, se modificará **el primer documento de la colección**



READ y un poco más...

- findAndModify
- findOneAndDelete
- findOneAndReplace
- findOneAndUpdate



READ y un poco más...

Leer y modificar documentos utilizando el método de Mongosh **findAndModify**, recibe como parámetro un documento con la configuración de la búsqueda mediante etiquetas y valores.

```
db.collection.findAndModify({
```

query: <document>,

Opcional. Criterios de selección para la modificación. Emplea los mismos selectores de consulta que se utilizan en el método find(). Aunque la consulta puede coincidir con varios documentos, db.collection.findAndModify() solo seleccionará un documento para modificarlo.

sort: <document>,

Opcional. Modificará el primer documento según el orden

remove: <boolean>,

Quita el documento que cumple con el query especificado. El valor predeterminado es false.

update: <document or aggregation pipeline>, Realiza un update en el documento que cumple con el query especificado.

new: <boolean>,
del original. El

Opcional. Cuando es true, devuelve el documento actualizado en lugar

valor predeterminado es false.

fields: <document>,

Opcional. Indica el subset de campos a retornar

upsert: <boolean>,

Opcional. Se utiliza junto con el campo de update. Cuando es true, findAndModify() crea un nuevo documento si no hay documentos que coincidan con la consulta, sino actualiza un único documento que coincida con la consulta.

```
.....  
});
```



READ and more

Leer y modificar documentos utilizando el método de Mongosh **findOneAndDelete**

Elimina un único documento en función de los criterios de filtro y ordenación, devolviendo el documento eliminado.

SINTAXIS

```
db.collection.findOneAndDelete(  
  <filter>,                                     →document  
  {  
    writeConcern: <document>,  
    projection: <document>,  
    sort: <document>,  
    maxTimeMS: <number>,  
    collation: <document>  
  }  
)
```



READ and more

Leer y modificar documentos utilizando el método de Mongosh **findOneAndReplace**

Reemplaza un solo documento en función del filtro y orden.

SINTAXIS

```
db.collection.findOneAndReplace(  
  <filter>,          →  
  document  
  <replacement>,    →  
  document  
  {  
    writeConcern: <document>,  
    projection: <document>,  
    sort: <document>,  
    maxTimeMS: <number>,  
    upsert: <boolean>,  
    returnDocument: <string>,  
    returnNewDocument: <boolean>,  
    collation: <document>  
  }  
)
```



READ and more

Leer y modificar documentos utilizando el método de Mongosh **findOneAndUpdate**

Actualiza un único documento en función de los criterios de filtro y ordenación.

Devuelve el documento original de forma predeterminada. Devuelve el documento actualizado si returnNewDocument se establece en true.

SINTAXIS

```
db.collection.findOneAndUpdate(  
  <filter>,                                → document  
  <update document..>, → con operadores de update p.e. $set  
  {  
    writeConcern: <document>,  
    projection: <document>,  
    sort: <document>,  
    maxTimeMS: <number>,  
    upsert: <boolean>,  
    returnDocument: <string>,  
    returnNewDocument: <boolean>,  
    collation: <document>,  
    arrayFilters: [ <filterdocument1>, ... ]  
  })
```



READ and more

Ejemplos:

● **findAndModify**

```
db.pizzas.findAndModify({query:{},update:{$set: {size: "medium"}, $inc: { price: 2 }}})
db.pizzas.findAndModify({query:{_id:5},remove:true})
```

● **findOneAndDelete**

```
db.pizzas.findOneAndDelete({}, { sort : { "_id" : -1 } })
```

● **findOneAndReplace**

```
db.pizzas.findOneAndReplace({_id:1}, {"type": "chedar", "size": "medium", "price": 8})
```

● **findOneAndUpdate**

```
db.pizzas.findOneAndUpdate({_id:2}, {$set: {size: "small"}})
db.pizzas.findOneAndUpdate({_id:1}, {$set: {type: "chedar", size: "medium", price: 8}})
```

BulkWrite

BulkWrite() admite una combinación de operaciones de inserción, actualización, eliminación y reemplazo. El método toma una matriz de objetos de operación de escritura y un objeto de opciones (opcional). Cada objeto de operación de escritura representa una única operación de base de datos y puede ser de uno de los siguientes tipos: insertOne, updateOne, updateMany, deleteOne, deleteMany o replaceOne.

Por defecto las operaciones se ejecutan en orden. Si se establece la opción ordered en falso, mongod puede reordenar las operaciones para aumentar el rendimiento.

db.pizzas.bulkWrite ([

```
{ insertOne: { document: { _id: 33, type: "beef", size: "medium", price: 6 } } },
{ insertOne: { document: { _id: 44, type: "sausage", size: "large", price: 10 } } },
{ updateOne: {
  filter: { type: "cheddar" },
  update: { $set: { price: 8 } }
},
{ deleteOne: { filter: { type: "pepperoni" } } },
{ replaceOne: {
  filter: { type: "vegan" },
  replacement: { type: "tofu", size: "small", price: 4 }
}
}, { ordered: true } ] > opciones
```