# Indice

[**Indice**](#_wpc2avosj8pg) **1**

[**Operaciones CRUD**](#_obvbcwq6vbac) **3**

[Insert - Alternativa 1 (mongoimport)](#_3wngi4rqvw2n) 3

[Insert - Alternativa 2 (consola)](#_8tuszrkx2eb0) 4

[Insert - Alternativa 3 (función)](#_4btuzqqimf2w) 5

[Insert - Alternativa 4 (array)](#_juu2ajpw0vou) 6

[**Documentos Referenciados**](#_durwc1mt20wv) **7**

[**find, skip & limit**](#_9mnso9pjb4ld) **8**

[**find**](#_d6gwqp13w9it) **10**

[**findOne**](#_hh1m8ucokpai) **11**

[**distinct**](#_aj3xc7bvq7lk) **11**

[**Sort**](#_59s0xl8oax6m) **13**

[save](#_pyxyulproikf) **14**

[update](#_ah6sgnsfxacs) 16

[$unset](#_lw0gudr4xuxc) 17

[upsert](#_t0kxsqgi5ems) 17

[Multi](#_49tlzy4rpf3n) 18

[**Consultas Básicas**](#_4cidgoa6wvj3) **20**

[**Operadores en MongoDB**](#_98az5hycm57h) **21**

[Operadores de comparación](#_g7tq8jib5akf) 21

[$eq](#_x68d83qq09ae) 21

[$gt $gte $lt $lte](#_gdpk133p6ilh) 22

[$in](#_950zzywk3dfu) 23

[$ne](#_s92j3dut1zgl) 23

[$nin](#_irmdlmn4abke) 23

[Operadores lógicos](#_nd9wys3rc2as) 24

[$and](#_m0b41gcb5a30) 24

[$not](#_s9u2lq1l5dw) 24

[$nor](#_xip8nfy0omd4) 24

[$or](#_3ukbh6xin7mg) 25

[Operadores de consulta de elementos](#_eu9ua0lvhfn) 25

[$exists](#_g9fg1w1ydec9) 25

[$type](#_wb1tudsnf8rm) 26

[Operadores de consulta de evaluación](#_ovl80w6g98l5) 27

[$regex(hace una busqueda con string dentro de un volumen grande)](#_cffbzzyvxywf) 27

[$text](#_sk8lqb1ck9u) 28

[$where](#_mbxoyfyywqvb) 29

[Operadores para gestión de Arrays](#_gesm8gj2j69p) 30

[$all (opuesto de $in)](#_p8xvyl6tsgod) 30

[$elemMatch](#_816ow7sqp0mz) 31

[$size](#_e9fxkterrsj0) 32

[Ubicación en el array](#_hluaw9fotgym) 32

[Operadores de proyección](#_57clj6kwxrgb) 32

[$](#_6rzyz7sjbrda) 32

[$slice](#_pjtr09tn5bvr) 34

[Operadores de actualización](#_g184e1iqizl9) 34

[Claves](#_z835oomzois3) 35

[$currentDate](#_ep261nq608az) 35

[$inc](#_w4puedvp91fp) 36

[$min](#_ofgfns35yryj) 37

[$max](#_yb7opigatlny) 37

[$mul](#_cpw35ihghbfd) 38

[$rename](#_ozqsbdtacidd) 39

[$set](#_h47ab8t2j8hz) 40

[$unset](#_m2jxlv2r0zxu) 42

[$setOnInsert](#_smp5ewao0e9q) 43

[Arrays](#_23jz0rqbjs7s) 44

[$ (update)](#_2sdcwittzbsu) 44

[$addToSet](#_d5d57f5ili10) 45

[$each](#_8x5p51mz1jzl) 45

[$pop](#_cy4mt992qhpq) 46

[$pull](#_n7u9qs5so86d) 47

[$push](#_rmkc671f8xkr) 52

[$pullAll](#_a7786icdt344) 56

[$each](#_la6ternc91gx) 56

[$position](#_qbzf82pv8s7m) 56

[$slice](#_9dvffea9kpt0) 57

[**Fechas**](#_kpnj6ikfcvpx) **59**

[**Remove**](#_ergu2hlt5xj5) **59**

[**Drop**](#_dsfq5l519h8s) **60**

# **Operaciones CRUD**

**C**reate ⇨ CREATE (RDBMS)

**R**ead ⇨ SELECT

**U**pdate ⇨ UPDATE

**D**elete ⇨ DELETE

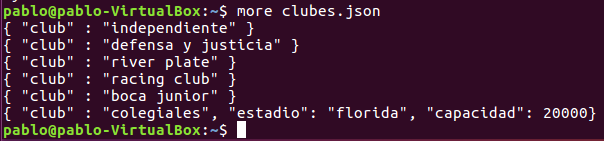
Cada vez que realicemos el insert de un documento en mongodb, vamos a darnos cuenta que cualquier sea el documento, siempre va a tener una clave llamada **\_id**. Esta clave se generará automáticamente, siempre que nosotros no lo especifiquemos nosotros durante la inserción.

Este clave \_id es la clave principal (PK) que tiene cada uno de los documentos. La clave \_id, como cualquier otra clave de MongoDB, puede ser de cualquier tipo. Podemos insertar números, texto o como hace MongoDB un ObjectId. Una vez insertado, el campo id no se puede modificar. En dicho caso, tenemos que borrar el documento, e insertarlo otra vez, pero con un \_id distinto. Un documento en MongoDB no puede pesar más de 16 Megas.

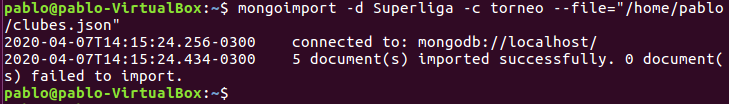
Para ingresar un nuevo documento en una base de datos MongoDB es posible de al menos, 4 alternativas diferentes: ￼

## **Insert - Alternativa 1 (mongoimport)**

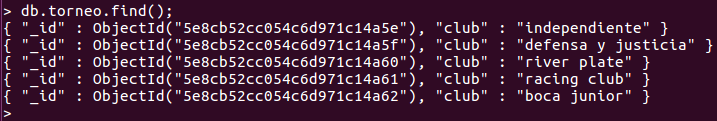
Vamos a utilizar una herramienta provista por el software de MongoDB: **mongoimport**. Para ello vamos a crear un archivo .JSON cuya característica es que cada documento ocupa una fila:



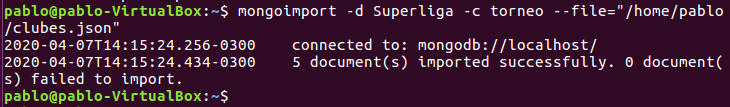
Luego, ejecutamos la utilidad mongoimport como se muestra a continuación:



Y podemos obtener el siguiente resultado:



También es posible ejecutar la utilidad mongoimport sin parámetros, como se muestra a continuación:

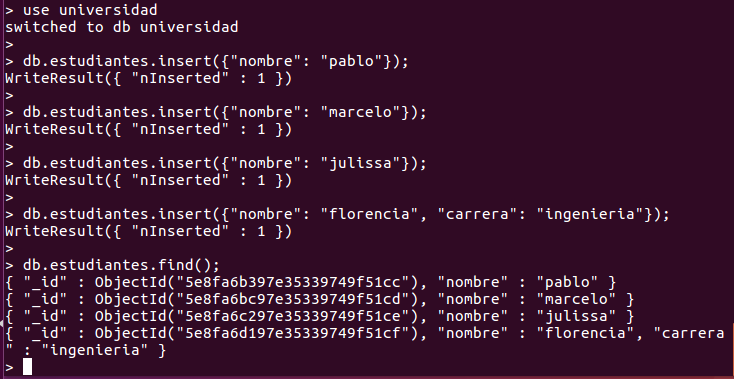




| **Importar documentos en MongoDB**  [*Insertando datos con $mongoimport*](https://youtu.be/xtK4x-H_-eY) |
| --- |

## **Insert - Alternativa 2 (consola)**

La segunda alternativa consiste en insertar un documento de a uno, de la siguiente manera:

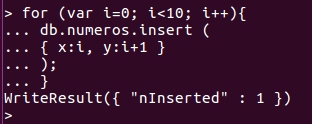




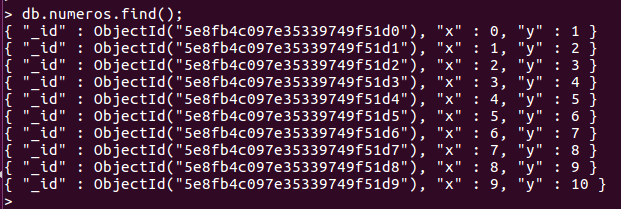
| **Importar documentos en MongoDB**  [*Insertando datos con INSERT*](https://youtu.be/vfUF_E1QN5M) |
| --- |

## **Insert - Alternativa 3 (función)**

La tercer alternativa de crear una función. Muy útil para pruebas desde el shell:



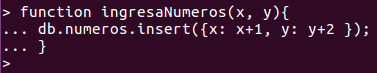




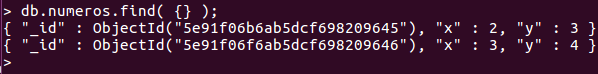


| **Importar documentos en MongoDB**  *Insertando datos con INSERT* |  |
| --- | --- |









Otra forma de crear un documento en MongoDB es asignar el documento a una variable y luego ingresar dicha variable:





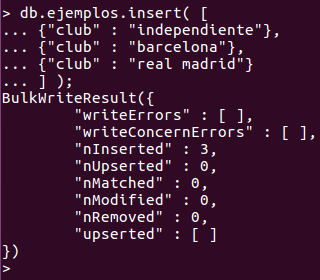


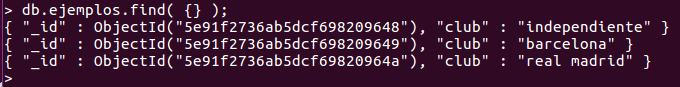
## **Insert - Alternativa 4 (array)**

La otra alternativa es insertar un array de documentos de forma simultánea:

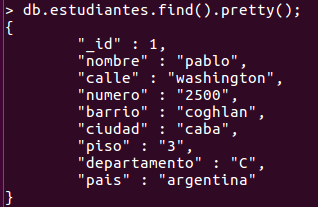


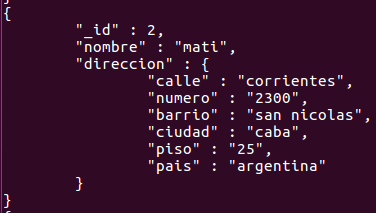


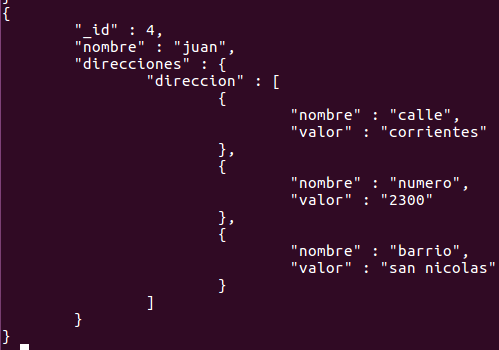


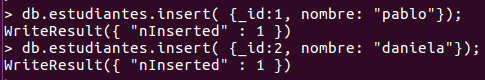


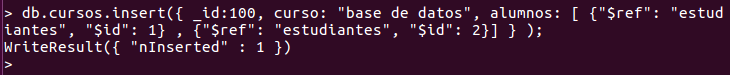
# **Documentos Referenciados**

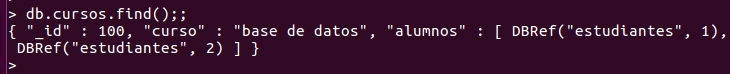


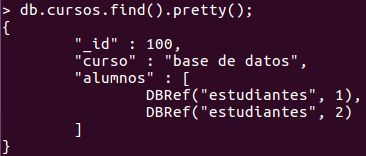


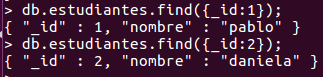












# **find, skip & limit**

Permite traer todos los documentos de una colección. El orden es irrelevante, ya que primero se aplicará el filtro (si existe), después skip, y finalmente limit.

db.ejemplo.find()

db.ejemplo.find({})

Inicialmente solo muestra los 10 primeros documentos. Para ver más teclear *it*. find devuelve un cursor, que puede verse como un iterador. It significa "iterate", es decir avanza el cursor a los siguientes resultados.

A menudo los documentos son demasiado complejos y difíciles de ver por lo tanto, se puede mejorar la presentación del documento en la shell mediante el uso de:

db.ejemplo.find().pretty();

En el shell es muy normal encadenar funciones con "." . En cualquier momento podemos añadir help() al final para ver las opciones disponibles.

Por ejemplo:

db.ejemplo.find({}).help()

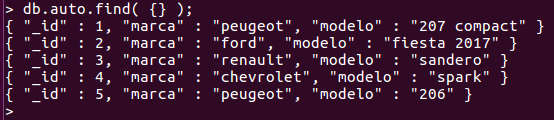
Otra forma más elegante de mostrar la información es mediante arrays, que además pueden sernos útiles para consultas complejas:

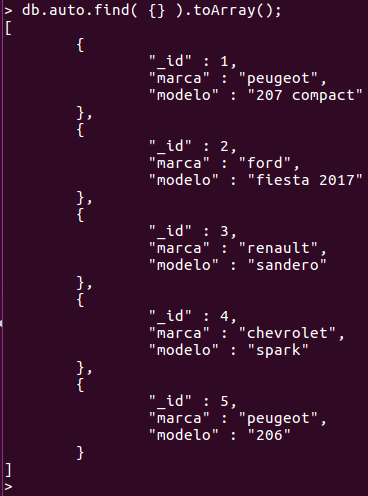
db.ejemplo.find({}).toArray()

que transfiere la colección productos a un array javascript y lo muestra de forma ordenada.

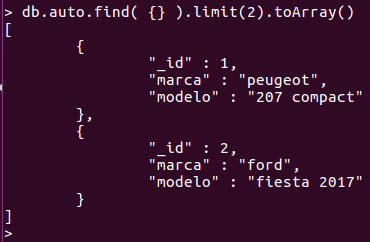




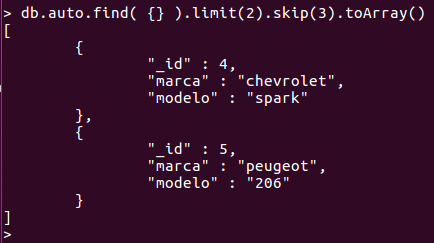




se quiere hacer un array con solo los 2 primeros elementos se puede limitar:



También se pueden "saltar" saltar unos cuantos resultados. Digamos que queremos los documentos que ocupan las posiciones 2 y 3, ambas incluídas:



# **find**

El find (similar a la sentencia SELECT en el paradigma relacional) tiene la siguiente estructura:

find(consulta, proyección)

consulta ⇨ indica qué documentos queremos seleccionar.

proyección ⇨ indica cual es la información que nos interesa mostrar de los documentos seleccionados con el find.

El ejemplo más simple:

db.coleccion.find();

db.coleccion.find({});

# **findOne**

Si solo queremos el primer elemento, podemos usar findOne:

db.coleccion.findOne();

Si no hay ningún documento que cumpla la query entonces devolverá el documento vacío, **null**, mientras que find(), que devuelve un cursor simplemente no devolverá nada. Supongamos el siguiente ejemplo:

use auto;

db.auto.drop();

db.auto.findOne();

null

# **distinct**

Devuelve el número de valores distintos de una clave (entre comillas) o de una query. count cuenta el número de documentos devuelto por una query.

Ejemplos:





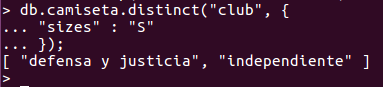


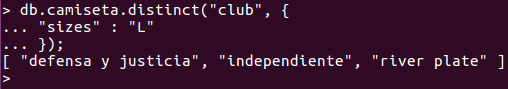






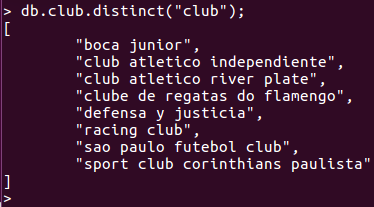
También podemos usarlo con una Query:

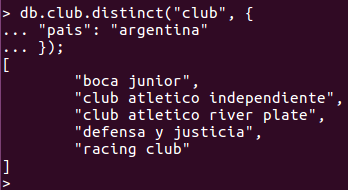


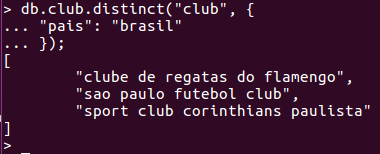


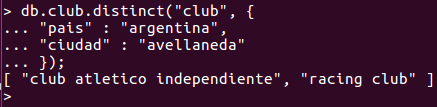
Otro ejemplo con otra base de datos:

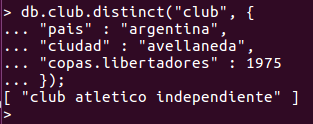












# **Sort**

Para ordenar se puede utilizar la siguiente estructura:

**db.coleccion.find(.....).sort(...);**

El criterio de ordenación se especifica de la siguiente manera:

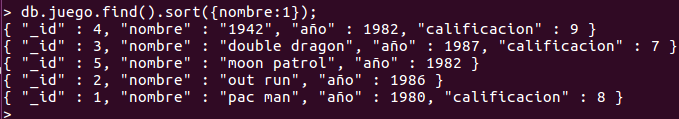
{ clave: 1 } ascendente

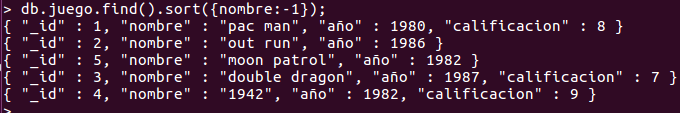
{ clave:-1 } descendente

Analizar las siguientes formas de uso del sort.



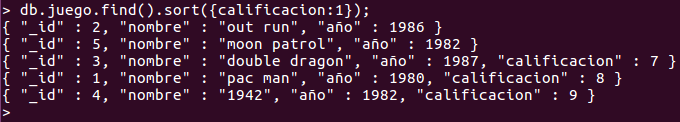


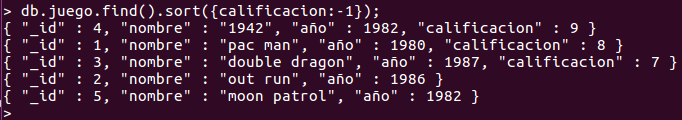






| Los documentos que no contengan alguna de las claves usadas en la ordenación se muestran al principio si es ascendente o se muestran al final caso sea descendente. |  |
| --- | --- |





| Orden de ejecución: skip(2), sort(1) y limit(3). |
| --- |

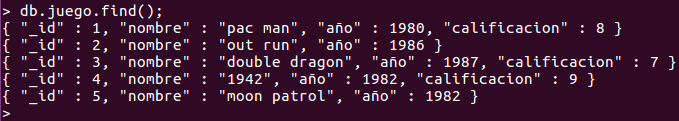
# **save**

El método [save()](https://docs.mongodb.com/manual/reference/method/db.collection.save/) se utiliza para actualizar un documento existente o insertar un nuevo documento si este no existe. Como podemos ver, este método hace uso tanto del comando [insert](https://docs.mongodb.com/manual/reference/command/insert/#dbcmd.insert) como del [update](https://docs.mongodb.com/manual/reference/command/update/#dbcmd.update).

Ejemplos:

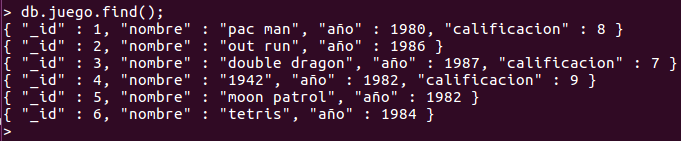






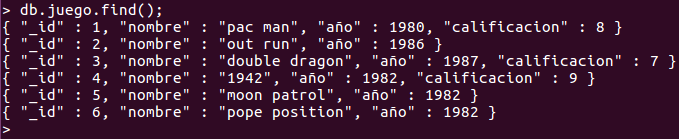
Insertamos un documento que no existe con save:





Actualiza un documento existente:







| **SAVE**  [*Repasamos cómo funciona*](https://youtu.be/-_HHXRNh6ic) |  |
| --- | --- |

## update

Las modificaciones o actualizaciones de los documentos siguen la siguiente estructura:

db.coleccion.update(*query*, *update*, *options*);

* *query* ⇨ selecciona o busca los documentos que se quieren actualizar.
* *update* ⇨ especifica cuales son los cambios que se van a aplicar.
* options ⇨ es opcional. Admite las opciones:
  + $upsert
  + $multi



| El update como lo conocemos en el RDBMS funciona diferente dado que el update en MongoDB reemplaza un documento por otro. A excepción del \_id que nunca puede ser reemplazado. |
| --- |

En este tipo de update, al segundo parámetro, deberíamos considerarlo simplemente como el valor (o los pares clave:valor) que queremos poner en el documento (excepto el \_id que será el mismo que en el documento inicial.

Veamos un ejemplo:

use automoviles;

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

Supongamos que queremos hacer un cambio: el modelo 207 lo queremos actualizar a : “207 compact feline”:

db.auto.update({\_id:1}, {modelo:"207 compact feline"});

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

db.auto.find({});

{ "\_id" : 1, "modelo" : "207 compact feline" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

Como podemos observar, el update reemplazó el documento completo. Volvemos a ejecutar el update de manera de no perder las claves que originalmente teníamos en el documento:

db.auto.update({\_id:1}, { "marca" : "peugeot", "modelo" : "207 compact" });

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

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

### $unset

db.auto.update({\_id:1}, {$unset : { año: 2012}});

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

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

### upsert



| Cuando la condición del update no coincide con ninguna clave de nuestra base de datos, podemos hacer que el update funcione como un insert, para ello debemos usar el operador upsert. |
| --- |

db.auto.update({marca:"fiat"}, {modelo:"palio"});

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

db.auto.update({marca:"fiat"}, {modelo:"palio"}, {upsert: true});

WriteResult({

"nMatched" : 0,

"nUpserted" : 1,

"nModified" : 0,

"\_id" : ObjectId("5a4e05ad602dab02df9fb93d")

})

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

{ "\_id" : ObjectId("5a4e05ad602dab02df9fb93d"), "modelo" : "palio" }

db.auto.remove({modelo:"palio"});

WriteResult({ "nRemoved" : 1 })

db.auto.update({marca:"fiat"}, {\_id: 6, marca: "fiat", modelo:"palio"}, {upsert: true});

WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0, "\_id" : 6 })

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

{ "\_id" : 6, "marca" : "fiat", "modelo" : "palio" }

### Multi

db.auto.update({}, {$set : { procedencia: "argentina"}});

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

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "procedencia" : "argentina" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

db.auto.update({}, {$unset : { procedencia: "argentina"}});

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

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

db.auto.update({}, {$set : { procedencia: "argentina"}}, {multi : true});

WriteResult({ "nMatched" : 5, "nUpserted" : 0, "nModified" : 5 })

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "procedencia" : "argentina" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017", "procedencia" : "argentina" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero", "procedencia" : "argentina" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark", "procedencia" : "argentina" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206", "procedencia" : "argentina" }

db.auto.update({"marca" : "peugeot"}, {$set : { cantidad: 0}}, {multi : true});

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "procedencia" : "argentina", "cantidad" : 0 }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017", "procedencia" : "argentina" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero", "procedencia" : "argentina" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark", "procedencia" : "argentina" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206", "procedencia" : "argentina", "cantidad" : 0 }

**Proyecciones**

También tienen la forma de un documento JSON. Indican qué información se mostrará de los documentos obtenidos en una consulta.

La proyección no cambia el número (cantidad) de documentos que mostrará (esto es objetivo de la consulta y del uso de funciones como limit y skip). Lo que determina la proyección es qué clave/s y su/s respectivo valor se van a mostrar.

Las reglas básicas de la proyección son las siguientes:

* { }

muestra todos los pares clave:valor

* {clave1:true,...,clave2:true}

muestra solo las claves que tengan asignado true (y además la clave \_id)

* {clave1:false,...,clave2:false}

muestra todas las claves salvo la que tienen asignado false (y además la clave \_id)



| No se pueden mezclar valores true y false, con una excepción; en el caso de documento con valores true se puede añadir "\_id":false para indicar que no queremos que se muestre el identificador. |
| --- |

Algunos ejemplos utilizando la [base de datos de productos](#_wmnlotec913):

use productos;

db.producto.find({},{nombre:true, cantidad:false});

Error: error: }

"waitedMS" : NumberLong(0),

"ok" : 0,

"errmsg" : "Projection cannot have a mix of inclusion and exclusion.",

"code" : 2

}

db.producto.find({},{nombre:true, cantidad:true });

{ "\_id" : 100, "nombre" : "alfajor", "cantidad" : 150 }

{ "\_id" : 101, "nombre" : "muffin de chocolate", "cantidad" : 500 }

{ "\_id" : 102, "nombre" : "coca zero", "cantidad" : 70 }

{ "\_id" : 103, "nombre" : "coca light", "cantidad" : 200 }

{ "\_id" : 104, "nombre" : "barra de cereal", "cantidad" : 2000 }

{ "\_id" : 105, "nombre" : "champagne", "cantidad" : 20 }

Analizar el resultado de ejecutar:

db.producto.find({},{nombre:true});

db.producto.find({},{nombre:true, \_id:false});

# Consultas Básicas

Las consultan toman la forma de un documento JSON. Se pueden comparar a la sección *where* de una consulta *SQL* . Sirven para seleccionar un subconjunto de documentos de una colección (aquellos que cumplen la condición expresada por la consulta).

* El documento vacío { } equivale a aceptar todos los elementos de la colección. Por ejemplo:

db.coleccion.findOne({ });

db.coleccion.findOne();

* {a:b}

Esto significa: "traer todos los documentos en los que la clave a toma el valor b.

En SQL sería "where a=b".

Vamos a buscar de nuestra base de datos clubes, colección club solamente 2 equipo de fútbol que sean de la argentina:

db.club.find({"pais":"brasil"},{"club":true,"\_id":false}).limit(2);

{ "club" : "sao paulo futebol club" }

{ "club" : "sport club corinthians paulista" }

Se pueden agregar condiciones simplemente añadiendo parejas clave:valor.

db.club.find({"ciudad": "avellaneda"},{"club":true,"\_id":false}).limit(2);

{ "club" : "club atletico independiente" }

{ "club" : "racing club" }

# Operadores en MongoDB

## Operadores de comparación

$eq $gt $gte $in $lt $lte $ne $nin

### $eq

Veamos algunos ejemplos:

use videojuegos;

db.juego.find({año: 1980});

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980 }

db.juego.find({año: { $eq:1980}});

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980 }

db.juego.find({nombre: "out run"});

{ "\_id" : 2, "nombre" : "out run", "año" : 1986 }

db.juego.find({nombre: { $eq:"out run"}});

{ "\_id" : 2, "nombre" : "out run", "año" : 1986 }

use preferencias;

db.gustos.find( { hobby: { $eq: "futbol"}}, {nombre:true, \_id:false});

{ "nombre" : "pablo" }

db.gustos.find( { hobby: { $eq: "viajar"}}, {nombre:true, \_id:false});

{ "nombre" : "pablo" }

{ "nombre" : "marcelo" }

db.gustos.find( { hobby:"viajar"}, {nombre:true, \_id:false});

{ "nombre" : "pablo" }

{ "nombre" : "marcelo" }

db.gustos.find( { hobby:"buceo"}, {nombre:true, \_id:false});

{ "nombre" : "marcelo" }

db.gustos.find( { comida:"m&m"}, {nombre:true, \_id:false});

{ "nombre" : "pablo" }

db.gustos.find( { comida: { $eq: "m&m"}}, {nombre:true, \_id:false});

{ "nombre" : "pablo" }

### $gt $gte $lt $lte

use numeros;

db.coordenadas.find({x:{$gt:5}, y:{$lt:6}},{"\_id":false});

{ "x" : 6, "y" : 3 }

{ "x" : 7, "y" : 3.5 }

{ "x" : 8, "y" : 4 }

{ "x" : 9, "y" : 4.5 }

Escribir una consulta en la colección coordenadas, que nos permita encontrar valores tales que 3<x<5:

db.coordenadas.find({x:{$gt:3, $lt:5}},{"\_id":false});

{ "x" : 4, "y" : 2 }



| Con el objetivo de no repetir claves y hacer una consulta más clara, se pueden utilizar en un documento, las restricciones que se sean necesario.  En este próximo ejemplo podemos ver dos restricciones para la misma variable. |
| --- |

En la colección de clubes traer todos los equipos que empiezan con r:

use clubes;

db.club.find({"club":{"$gte":"r", "$lt":"s"}}, {"club":true, "\_id":false});

{ "club" : "racing" }

db.club.find({"club":{"$gte":"a", "$lt":"z"}}, {"club":true, "\_id":false});

{ "club" : "club atletico independiente" }

{ "club" : "racing club" }

{ "club" : "boca junior" }

{ "club" : "club atletico river plate" }

{ "club" : "defensa y justicia" }

{ "club" : "sao paulo futebol club" }

{ "club" : "sport club corinthians paulista" }

{ "club" : "clube de regatas do flamengo" }

### $in

Con el operador $in podemos preguntar por algunos de los elementos que le indiquemos. Es decir, podemos buscar documentos que contengan en el array alguno de los elementos especificados. Veamos un ejemplo:

use preferencias;

db.gustos.find({"hobby":{ $in: ["jockey","rugby","futbol"]}}, {nombre:true,"\_id":false});

{ "nombre" : "pablo" }

db.gustos.find({"comida":{ $in: ["chocolate", "pizza", "coca zero"]}}, {nombre:true,"\_id":false});

{ "nombre" : "pablo" }

{ "nombre" : "marcelo" }

### $ne

$ne selecciona los documentos donde el valor de la clave no es igual al valor especificado. Dicho con otras palabras devuelve los documentos donde el valor de la clave no es igual al valor especificado. Incluye los documentos que no están contenidos en la clave.

use videojuegos;

db.juego.find( { año: { $ne: 1986 } } )

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980, "calificacion" : 8 }

{ "\_id" : 3, "nombre" : "double dragon", "año" : 1987, "calificacion" : 7 }

{ "\_id" : 4, "nombre" : "1942", "año" : 1982, "calificacion" : 9 }

{ "\_id" : 5, "nombre" : "moon patrol", "año" : 1982 }

db.juego.find( { nombre: { $ne: "pac man" } } )

{ "\_id" : 2, "nombre" : "out run", "año" : 1986 }

{ "\_id" : 3, "nombre" : "double dragon", "año" : 1987, "calificacion" : 7 }

{ "\_id" : 4, "nombre" : "1942", "año" : 1982, "calificacion" : 9 }

{ "\_id" : 5, "nombre" : "moon patrol", "año" : 1982 }

### $nin

Asegura que la clave que es del tipo array no incluye ninguno de los valores que se le pasa.

use preferencias;

db.gustos.find({hobby:{$nin:["golf","tenis"]}},{nombre:true,"\_id":false});

{ "nombre" : "pablo" }

{ "nombre" : "marcelo" }

{ "nombre" : "francisco" }

db.gustos.find({hobby:{$nin:["ahorro","tenis"]}},{nombre:true,"\_id":false});

{ "nombre" : "pablo" }

{ "nombre" : "marcelo" }

db.gustos.find({hobby:{$nin:["ahorro","futbol"]}},{nombre:true,"\_id":false});

{ "nombre" : "marcelo" }

## Operadores lógicos

$and $not $nor $or

### $and

use videojuegos;

db.juego.find( { $and: [ {año: 1980 } , {calificacion: 8} ] } )

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980, "calificacion" : 8 }

### $not

Devuelve el valor opuesto al valor de entrada. Si el valor es true, devuelve false. Si el valor es false, devuelve true. Dicho de otra manera, busca los documentos que no cumplan una determinada condición.



| El operador $not sólo puede usarse con otros operadores como $gt o $lt. No puede usarse con valores directos o documentos. Para eso contamos con el operador $ne que hemos [explicado anteriormente](#_s92j3dut1zgl). |
| --- |

use videojuegos;

db.juego.find( { año: {$not: { $gt: 1985 } } } )

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980, "calificacion" : 8 }

{ "\_id" : 4, "nombre" : "1942", "año" : 1982, "calificacion" : 9 }

{ "\_id" : 5, "nombre" : "moon patrol", "año" : 1982 }

db.juego.find( { $and: [ { año: { $not: { $gt: 1985 } } } , {calificacion: { $gt: 8}} ] } )

{ "\_id" : 4, "nombre" : "1942", "año" : 1982, "calificacion" : 9 }

### $nor

El operador $nor acepta dos o más valores. Veamos su aplicación:



| Tanto los operadores $not y $nor, pueden devolver documentos aún si las claves no existen. Para evitar esto, si es algo que no queremos que suceda, podemos añadir el operador [$exists](#_g9fg1w1ydec9). |
| --- |

use clubes;

db.club.find( { $nor: [ { pais: "argentina"} , { capacidad: { $lte: 50000}} ] },

{\_id:false, club:true, capacidad: true});

{ "club" : "sao paulo futebol club", "capacidad" : 67400 }

{ "club" : "clube de regatas do flamengo", "capacidad" : 95000 }

### $or

Buscar coordenadas que tengan o bien 2<x<4 o 6<x<10.

use numeros;

db.coordenadas.find({

$or:[ { x: {$gt:2, $lt:4} }, { x: {$gt:6, $lt:10} }

]},

{"\_id":false});

{ "x" : 3, "y" : 1.5 }

{ "x" : 7, "y" : 3.5 }

{ "x" : 8, "y" : 4 }

{ "x" : 9, "y" : 4.5 }

Veamos un ejemplo más:

use articulos;

db.articulo.find( {

$and: [

{ $or: [ {precio: 450}, {precio: 100} ]},

{ $or: [ {autor: "martin"}, {calificacion: {$gte: 7}}] }

]

});

{ "\_id" : ObjectId("5a43767904fd14a6cc0ab56c"), "nombre" : "gestion de proyectos", "autor" : "martin", "idiomas" : { "espanol" : true, "ingles" : false, "portugues" : false }, "precio" : 450, "impuestos" : 140, "calificacion" : 9 }

## Operadores de consulta de elementos

$exists $type

### $exists

El operador $exists es muy útil para comprobar si una determinada clave existe (o no) en un documento. Supongamos la base de datos videojuegos y queremos saber todos los juegos que no tienen la clave calificación.

use videojuegos;

db.juego.find( { calificacion: {$exists: false}});

{ "\_id" : 2, "nombre" : "out run", "año" : 1986 }

{ "\_id" : 5, "nombre" : "moon patrol", "año" : 1982 }

db.juego.find( { calificacion: {$exists: true}});

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980, "calificacion" : 8 }

{ "\_id" : 3, "nombre" : "double dragon", "año" : 1987, "calificacion" : 7 }

{ "\_id" : 4, "nombre" : "1942", "año" : 1982, "calificacion" : 9 }

db.juego.find( { calificacion: {$exists: true, $nin: [8, 9] }});

{ "\_id" : 3, "nombre" : "double dragon", "año" : 1987, "calificacion" : 7 }

### $type

El operador $type se utiliza para buscar elementos con un cierto tipo. [Los tipos son codificados](https://docs.mongodb.com/v3.2/reference/operator/query/type/) numéricamente siguiendo la definición de BSON.

use videojuegos;

db.juego.find( { nombre: {$type: 1}});

db.juego.find( { nombre: {$type: 2}});

{ "\_id" : 1, "nombre" : "pac man", "año" : 1980, "calificacion" : 8 }

{ "\_id" : 2, "nombre" : "out run", "año" : 1986 }

{ "\_id" : 3, "nombre" : "double dragon", "año" : 1987, "calificacion" : 7 }

{ "\_id" : 4, "nombre" : "1942", "año" : 1982, "calificacion" : 9 }

{ "\_id" : 5, "nombre" : "moon patrol", "año" : 1982 }

Supongamos el siguiente caso especial:

use ejemplos;

db.ejemplo.remove({});

db.ejemplo.insert({\_id:1, clave1: "uno"});

WriteResult({ "nInserted" : 1 })

db.ejemplo.insert({\_id:2, clave1:1});

WriteResult({ "nInserted" : 1 })

db.ejemplo.insert({\_id:3, clave1: new Date()});

WriteResult({ "nInserted" : 1 })

db.ejemplo.insert({\_id:4, clave1: 3.1515});

WriteResult({ "nInserted" : 1 })

db.ejemplo.insert({\_id:5, clave1: ["sim", "nao", "oi"]});

WriteResult({ "nInserted" : 1 })

db.ejemplo.insert({\_id:6, clave1: ["hola", "hi", "oi"]});

WriteResult({ "nInserted" : 1 })

db.ejemplo.insert({\_id:7, clave1: ["one", "two", "three", "four"]});

WriteResult({ "nInserted" : 1 })

db.ejemplo.find( { clave1: {$type: 1}});

{ "\_id" : 2, "clave1" : 1 }

{ "\_id" : 4, "clave1" : 3.1515 }

db.ejemplo.find( { clave1: {$type: "double"}});

{ "\_id" : 2, "clave1" : 1 }

{ "\_id" : 4, "clave1" : 3.1515 }

db.ejemplo.find( { clave1: {$type: 9}});

{ "\_id" : 3, "clave1" : ISODate("2018-01-08T16:19:12.378Z") }

db.ejemplo.find( { clave1: {$type: "date"}});

{ "\_id" : 3, "clave1" : ISODate("2018-01-08T16:19:12.378Z") }

db.ejemplo.find( { "clave1." : {$exists: true}});

{ "\_id" : 5, "clave1" : [ "sim", "nao", "oi" ] }

{ "\_id" : 6, "clave1" : [ "hola", "hi", "oi" ] }

{ "\_id" : 7, "clave1" : [ "one", "two", "three", "four" ] }

{ "\_id" : 8, "clave1" : [ ] }

## Operadores de consulta de evaluación

$regex $text

### $regex(hace una busqueda con string dentro de un volumen grande)

Permite definir expresiones regulares sobre strings. Veamos algunos ejemplos empleando la [base de datos de videojuegos](https://docs.google.com/document/d/1dx-4XSqk1uVM7qRJsLuOYGgw5RdAPlh4mp5FGKRv23M/edit#heading=h.xuokb4mx3gk9).

use videojuegos;

db.juego.find({nombre:{$regex:"1"}},{"nombre":true,"\_id":false});

{ "nombre" : "1942" }

db.juego.find({nombre:{$regex:"oo"}},{"nombre":true,"\_id":false});

{ "nombre" : "moon patrol" }

La forma de uso general es:

{ : { $regex: /pattern/, $options: '' } }  
{ : { $regex: 'pattern', $options: '' } }  
{ : { $regex: /pattern/ } }

La parte $options es opcional e incluye una cadena que puede contener los caracteres i,m,x,s :

**i** ⇨ No distingue mayúsculas o minúsculas

**m** ⇨ Para patrones con ^ o $, asume cadenas multilínea y devuelve un resultado por cada línea

**x** ⇨ Ignora espacios en blanco en el patrón a no ser que vengan "escapados" por \

**s** ⇨ En el patrón el caracter especial '.' encaja con cualquier carácter, incluyendo el fin de línea

Veamos más ejemplos:

db.juego.find({nombre:{$regex:"OO"}},{"nombre":true,"\_id":false});

db.juego.find({nombre:{$regex:"OO", "$options":"i"}},{"nombre":true,"\_id":false});

{ "nombre" : "moon patrol" }

db.juego.find({nombre:{$regex:"OUT", "$options":"i"}},{"nombre":true,"\_id":false});

{ "nombre" : "out run" }

Vamos a buscar todos los documentos cuyo valor de clave termina con el carácter “n”. Para ello se debe indicar el carácter deseado, "n" en este caso, seguido del símbolo **“$”** que indica el final de cadena:

db.juego.find({nombre:{$regex:"n$"}},{"nombre":true,"\_id":false});

{ "nombre" : "pac man" }

{ "nombre" : "out run" }

{ "nombre" : "double dragon" }

db.juego.find({nombre:{$regex:"^d"}},{"nombre":true,"\_id":false});

{ "nombre" : "double dragon" }

### $text

db.apuntes.find( { $text: { $search: "mongodb" } } );

{ "\_id" : 2, "titulo" : "indices en mongodb", "autor" : "pablo", "visitas" : 5 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

{ "\_id" : 8, "titulo" : "mongodb y cassandra", "autor" : "pablo", "visitas" : 10 }

Busca que esten las palabraS:

db.apuntes.find( { $text: { $search: "indices cassandra" } } );

{ "\_id" : 2, "titulo" : "indices en mongodb", "autor" : "pablo", "visitas" : 5 }

{ "\_id" : 3, "titulo" : "indices en oracle", "autor" : "marcelo", "visitas" : 90 }

{ "\_id" : 8, "titulo" : "mongodb y cassandra", "autor" : "pablo", "visitas" : 10 }

substring:

db.apuntes.find( { $text: { $search: "\"programacion en\"" } } )

{ "\_id" : 1, "titulo" : "programacion en oracle", "autor" : "marcelo", "visitas" : 50 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

db.apuntes.find( { $text: { $search: "mongodb cassandra" } } )

{ "\_id" : 2, "titulo" : "indices en mongodb", "autor" : "pablo", "visitas" : 5 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

{ "\_id" : 8, "titulo" : "mongodb y cassandra", "autor" : "pablo", "visitas" : 10 }

db.apuntes.find( { $text: { $search: "mongodb -cassandra" } } )

{ "\_id" : 2, "titulo" : "indices en mongodb", "autor" : "pablo", "visitas" : 5 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

db.apuntes.find( { $text: { $search: "MONGODB", $caseSensitive: false } } )

{ "\_id" : 2, "titulo" : "indices en mongodb", "autor" : "pablo", "visitas" : 5 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

{ "\_id" : 8, "titulo" : "mongodb y cassandra", "autor" : "pablo", "visitas" : 10 }

db.apuntes.find( { $text: { $search: "\"PROGRAMACION EN\"", $caseSensitive: false } } );

{ "\_id" : 1, "titulo" : "programacion en oracle", "autor" : "marcelo", "visitas" : 50 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

db.apuntes.find( { $text: { $search: "MONGODB -CASSANDRA", $caseSensitive: false } } );

{ "\_id" : 2, "titulo" : "indices en mongodb", "autor" : "pablo", "visitas" : 5 }

{ "\_id" : 4, "titulo" : "programacion en mongodb", "autor" : "marcelo", "visitas" : 100 }

### $where

Este operador solo se puede usar en consola. Permite establecer una condición que deben cumplir todos documentos. El formato es del tipo $where:cadena, donde cadena es un string con una expresión JavaScript. El documento actual se referencia como this, y los valores asociados a sus claves como this.clave.

Por ejemplo en la colección de coordenada, devolver todas las que cumplan "x>y":

use numeros;

db.coordenadas.find({$where: "this.x > this.y"}, {\_id:false});

{ "x" : 1, "y" : 0.5 }

{ "x" : 2, "y" : 1 }

{ "x" : 3, "y" : 1.5 }

{ "x" : 4, "y" : 2 }

{ "x" : 5, "y" : 2.5 }

{ "x" : 6, "y" : 3 }

{ "x" : 7, "y" : 3.5 }

{ "x" : 8, "y" : 4 }

{ "x" : 9, "y" : 4.5 }

db.coordenadas.find({$where: "this.x > this.y+2"}, {\_id:false});

{ "x" : 5, "y" : 2.5 }

{ "x" : 6, "y" : 3 }

{ "x" : 7, "y" : 3.5 }

{ "x" : 8, "y" : 4 }

{ "x" : 9, "y" : 4.5 }

## Operadores para gestión de Arrays

$all $elemMatch $size

### $all (opuesto de $in)

El operador $all selecciona los documentos donde el valor de un campo es un array que *contiene todos* los elementos especificados. Este operador se utiliza para mapear todos los valores.

use preferencias;

db.gustos.find({"comida":{ $all: ["helado","chocolate"]}},{nombre:true,"\_id":false});

{ "nombre" : "pablo" }

db.gustos.find({"comida":{ $all:["helado"]}},{nombre:true,"\_id":false});

{ "nombre" : "pablo" }

{ "nombre" : "marcelo" }

Analizar la siguiente situación y obtener conclusiones:

db.gustos.find({"hobby":{ $all:["construccion", "ahorro"]}},{nombre:true,"\_id":false});

{ "nombre" : "francisco" }

db.gustos.find({"hobby":{ $all:["ahorro", "construccion"]}},{nombre:true,"\_id":false});

{ "nombre" : "francisco" }

db.gustos.find({"hobby":{ $all:["construccion"]}},{nombre:true,"\_id":false});

{ "nombre" : "francisco" }

db.gustos.find({"hobby":{ $all:["ahorro"]}},{nombre:true,"\_id":false});

{ "nombre" : "francisco" }

db.gustos.find({"hobby":{ $all:["ahorro", "construccion", "futbol"]}},{nombre:true,"\_id":false});

db.gustos.find({$and:[{"hobby": ["construccion", "ahorro"] } ] },{nombre:true,"\_id":false})

{ "nombre" : "francisco" }

db.gustos.find({$and:[{"hobby": ["ahorro", "construccion"] } ] },{nombre:true,"\_id":false})

db.gustos.find({$and:[{"hobby": ["ahorro"] } ] },{nombre:true,"\_id":false})

>

### $elemMatch

use clubes;

db.club.find({"copas.libertadores":{$elemMatch:{$gte:1984}}}, {\_id:false, club:true});

{ "club" : "club atletico independiente" }

{ "club" : "boca junior" }

{ "club" : "club atletico river plate" }

{ "club" : "sao paulo futebol club" }

{ "club" : "sport club corinthians paulista" }

db.club.find({"copas.libertadores":{$elemMatch:{$gte:1986}}}, {\_id:false, club:true});

{ "club" : "boca junior" }

{ "club" : "club atletico river plate" }

{ "club" : "sao paulo futebol club" }

{ "club" : "sport club corinthians paulista" }

db.club.find({"copas.sudamericana":{$elemMatch:{$gte:2017}}}, {\_id:false, club:true});

{ "club" : "club atletico independiente" }

db.club.find({"copas.libertadores":{$elemMatch:{$gte:1972, $lte: 1975 }}}, {\_id:false, club:true});

{ "club" : "club atletico independiente" }

db.club.find({"copas.libertadores":{$elemMatch:{$gte:2007, $lte: 2017 }}}, {\_id:false, club:true});

{ "club" : "boca junior" }

{ "club" : "club atletico river plate" }

{ "club" : "sport club corinthians paulista" }

### $size

El operador $size no acepta rango de valores.

use clubes;

db.club.find( { "copas.libertadores": { $size: 7 } }, {\_id:false, club:true} );

{ "club" : "club atletico independiente" }

db.club.find( { "copas.libertadores": { $size: 0 } }, {\_id:false, club:true} );

{ "club" : "defensa y justicia" }

db.club.find( { "copas.nicolas": { $size: 1 } }, {\_id:false, club:true} );

{ "club" : "boca juniors" }

{ "club" : "clube de regatas do flamengo" }

db.club.find( { $where: "this.copas.libertadores.length >= 6" },{\_id:false, club:true} );

{ "club" : "boca juniors" }

{ "club" : "club atletico independiente" }

### Ubicación en el array

Podemos consultar cual es la posición de un determinado valor dentro de un array. Ejemplo:

use preferencias;

db.gustos.find({"comida.1":"helado"},{nombre:1,comida:1, \_id:0});

{ "nombre" : "pablo", "comida" : [ "chocolate", "helado", "alfajor", "cafe", "chocotorta", "asado", "m&m" ] }

db.gustos.find({"comida.0":"chocolate"},{nombre:1,comida:1, \_id:0});

{ "nombre" : "pablo", "comida" : [ "chocolate", "helado", "alfajor", "cafe", "chocotorta", "asado", "m&m" ] }

## Operadores de proyección

$ $slice

### $

El operador posicional $ limita el resultado de la consulta del contenido del array para que contenga solo el primer elemento del array que coincida con la consulta, dicho de otra manera, permite que se muestre solo el primer elemento del array que satisface la consulta.  
  
Use $ con el método find() o el método findOne() cuando solo necesita un elemento de un array en particular respecto de los documentos seleccionados.

Tanto el operador $ como el operador $elemMatch proyectan el primer elemento que coincide de un array basado en una condición.

$ ⇨ Proyecta/trae el primer elemento del array que coincide de cada documento en una colección, dependiendo de la condición de búsqueda.

$elemMatch ⇨ Toma un argumento de condición explícita. Esto le permite proyectar basándose en una condición que no está en la consulta, o si necesita proyectar basándose en múltiples campos en los documentos incrustados de la matriz

Limitaciones que presenta

* Solo un operador posicional puede aparecer en un documento.
* Solo un array, el que está limitado con el operador $, debe aparecer en el documento de consulta.

Analicemos las siguientes consultas:

use clubes;

db.club.find({"copas.libertadores" : { $gte: 2000}}, {\_id:false, club:true});

{ "club" : "boca junior" }

{ "club" : "club atletico river plate" }

{ "club" : "sao paulo futebol club" }

{ "club" : "sport club corinthians paulista" }

db.club.find({

"copas.libertadores" : { $gte: 2000}

},

{

\_id: false,

club:true,

"copas.libertadores.$": 1

});

{ "club" : "boca junior", "copas" : { "libertadores" : [ 2000 ] } }

{ "club" : "club atletico river plate", "copas" : { "libertadores" : [ 2015 ] } }

{ "club" : "sao paulo futebol club", "copas" : { "libertadores" : [ 2005 ] } }

{ "club" : "sport club corinthians paulista", "copas" : { "libertadores" : [ 2012 ] } }

db.club.find({"copas.sudamericana":{$elemMatch:{$gte:2000}}}, {\_id:false, club:true});

{ "club" : "club atletico independiente" }

{ "club" : "boca junior" }

{ "club" : "club atletico river plate" }

{ "club" : "sao paulo futebol club" }

### $slice

Muestra partes de un array. Tiene dos opciones de uso:

Opción 1 ⇒ clave : { $slice : n }

muestra las n primeras componentes del array. Si n es negativo muestra las n últimas.

use preferencias;

db.gustos.find({},{\_id:0,nombre:1,comida:{$slice:1},hobby:{$slice:-1}});

{ "nombre" : "pablo", "comida" : [ "chocolate" ], "hobby" : [ "ver futbol" ] }

{ "nombre" : "marcelo", "comida" : [ "champane" ], "hobby" : [ "viajar" ] }

{ "nombre" : "francisco", "comida" : [ "salmón" ], "hobby" : [ "ahorro" ] }

db.gustos.find({},{\_id:0,nombre:1,comida:{$slice:2},hobby:{$slice:-2}});

{ "nombre" : "pablo", "comida" : [ "chocolate", "helado" ], "hobby" : [ "viajar", "remeras de futbol" ] }

{ "nombre" : "marcelo", "comida" : [ "champane", "pizza" ], "hobby" : [ "camionetas 4x4", "viajar" ] }

{ "nombre" : "francisco", "comida" : [ "salmón", "budin" ], "hobby" : [ "construccion", "ahorro" ] }

La segunda posibilidad tiene la forma:

Opción 2 ⇒ clave : { $slice : [ n1 , n2 ] }

n1 es la cantidad de valores que se "saltan", y n2 la cantidad de valores consecutivos a mostrar.

db.gustos.find({},{\_id:0,nombre:1,comida:{$slice:[2,3]},hobby:{$slice:[3,2]}});

{ "nombre" : "pablo", "comida" : [ "alfajor", "cafe", "chocotorta" ], "hobby" : [ "viajar", "remeras de futbol" ] }

{ "nombre" : "marcelo", "comida" : [ "helado", "coca zero", "mc donald" ], "hobby" : [ "camionetas 4x4", "viajar" ] }

{ "nombre" : "francisco", "comida" : [ "pastas" ], "hobby" : [ ] }

## Operadores de actualización

## Claves

### $currentDate

El operador [$currentDate](https://docs.mongodb.com/manual/reference/operator/update/currentDate/#up._S_currentDate) establece la fecha actual para una clave, ya sea como una [timestamp](https://docs.mongodb.com/manual/reference/bson-types/#document-bson-type-timestamp) o un [Date](https://docs.mongodb.com/manual/reference/bson-types/#document-bson-type-date). Repasar el concepto de [tiempo unix](https://es.wikipedia.org/wiki/Tiempo_Unix) ([convertidor](https://www.epochconverter.com/)).

use ejemplo;

db.usuario.remove({});

db.usuario.insert({\_id:1, status: "activo", ultimaModificacion: new Date()})

db.usuario.find({"\_id":1}).pretty();

{

"\_id" : 1,

"status" : "activo",

"ultimaModificacion" : ISODate("2018-01-13T00:40:30.451Z")

}

db.usuario.update({\_id:1}, {

$currentDate: {

ultimaModificacion: true,

"cancelacion.fecha": { $type: "timestamp" } },

$set: {

status: "cancelado",

"cancelacion.motivo": "pedido del usuario"

}

} );

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

db.usuario.find({"\_id":1}).pretty();

{

"\_id" : 1,

"status" : "cancelado",

"ultimaModificacion" : ISODate("2018-01-13T00:42:35.047Z"),

"cancelacion" : {

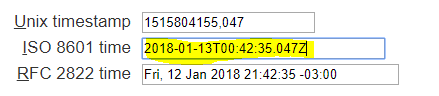
"fecha" : Timestamp(1515804155, 1),

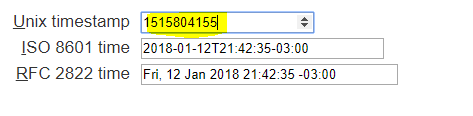
"motivo" : "pedido del usuario"

}

}

[Comparemos los formatos de fechas:](https://coderstoolbox.net/unixtimestamp/)





### $inc

El operador $inc se puede utilizar para incrementar el valor de una clave en una cantidad dada. Ejemplo:

use autos;

db.auto.update({"marca" : "peugeot"}, {$inc : { cantidad: 2}}, {multi : true});

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.auto.update({"marca" : "peugeot"}, {$inc : { cantidad: 1}}, {multi : true});

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "procedencia" : "argentina", "cantidad" : 3 }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017", "procedencia" : "argentina" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero", "procedencia" : "argentina" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark", "procedencia" : "argentina" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206", "procedencia" : "argentina", "cantidad" : 3 }

db.auto.update({"marca" : "peugeot"}, {$inc : { cantidad: 6}}, {multi : true});

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.auto.find({"marca" : "peugeot"});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "procedencia" : "argentina", "cantidad" : 9 }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206", "procedencia" : "argentina", "cantidad" : 9 }

db.auto.update({"marca" : "peugeot"}, {$inc : { cantidad: -3}}, {multi : true});

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.auto.find({"marca" : "peugeot"});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "procedencia" : "argentina", "cantidad" : 6 }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206", "procedencia" : "argentina", "cantidad" : 6 }

**Pregunta.** ¿Tendría sentido utilizar upsert cuando estamos haciendo $unset?

### $min

El operador $min actualiza el valor de una clave a un valor especificado si es que el valor especificado es menor que el valor actual de dicha clave.

use ejemplos;

db.alumno.remove({});

db.alumno.insert({ \_id: 1, nombre: "pablo"});

db.alumno.insert({ \_id: 2, nombre: "marcelo"});

db.alumno.update( {\_id:1}, {$set: {"nota.alta":8, "nota.baja":2}});

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

db.alumno.update( {\_id:2}, {$set: {"nota.alta":9, "nota.baja":6}});

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

db.alumno.update( {\_id:1}, {$min: {"nota.alta":7}});

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

db.alumno.find({\_id:1});

{ "\_id" : 1, "nombre" : "pablo", "nota" : { "alta" : 7, "baja" : 2 } }

db.alumno.update( {\_id:2}, {$min: {"nota.alta":8}});

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

db.alumno.find({\_id:2});

{ "\_id" : 2 , "nombre" : "marcelo", "nota" : { "alta" : 8, "baja" : 6 } }

db.alumno.update( {\_id:2}, {$min: {"nota.alta":9}});

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

db.alumno.find({\_id:2});

{ "\_id" : 2 , "nombre" : "marcelo", "nota" : { "alta" : 8, "baja" : 6 } }

### $max

use ejemplos;

db.alumno.remove({});

db.alumno.insert({ \_id: 1, nombre: "pablo"});

db.alumno.insert({ \_id: 2, nombre: "marcelo"});

db.alumno.update( {\_id:1}, {$set: {"nota.alta":6, "nota.baja":2}});

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

db.alumno.update( {\_id:2}, {$set: {"nota.alta":7, "nota.baja":6}});

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

db.alumno.update( {\_id:2}, {$max: {"nota.alta":8}});

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

db.alumno.find({\_id:2});

{ "\_id" : 2 , "nombre" : "marcelo", "nota" : { "alta" : 8, "baja" : 6 } }

db.alumno.update( {\_id:2}, {$max: {"nota.alta":7}});

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

db.alumno.find({\_id:2});

{ "\_id" : 2 , "nombre" : "marcelo", "nota" : { "alta" : 8, "baja" : 6 } }

db.alumno.update({}, {$max: {"nota.baja":4}});

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

db.alumno.find({});

{ "\_id" : 1 , "nombre" : "pablo", "nota" : { "alta" : 6, "baja" : 4 } }

{ "\_id" : 2 , "nombre" : "marcelo", "nota" : { "alta" : 8, "baja" : 6 } }

### $mul

El operador $mul multiplica el valor de una clave por un número. La clave debe tener un valor numérico. Si la clave no existe, el operador se encargará de crearlo.

use ejemplos;

db.producto.remove({});

db.producto.insert({ \_id: 1, nombre: "nueces"});

db.producto.insert({ \_id: 2, nombre: "almendras", precio: 150});

db.producto.insert({ \_id: 3, nombre: "chia"});

db.producto.update(

{ \_id: 1 },

{ $mul: { precio: 1.25 } }

);

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

db.producto.find({\_id:1});

{ "\_id" : 1, "nombre" : "nueces", "precio" : 0 }

db.producto.update(

{ \_id: 2 },

{ $mul: { precio: 2 } }

);

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

db.producto.find({\_id:2});

{ "\_id" : 2, "nombre" : "almendras", "precio" : 300 }

db.producto.update(

{ },

{ $mul: { precio: 3 } }

);

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

db.producto.find({ });

{ "\_id" : 2, "nombre" : "almendras", "precio" : 900 }

{ "\_id" : 3, "nombre" : "chia" }

{ "\_id" : 1, "nombre" : "nueces", "precio" : 0 }

### $rename

Se utiliza para renombrar la clave de un documento.

El operador $rename realiza un $unset y un $set de la clave del documento. Por lo tanto, dicha operación puede no garantizar el orden de las claves en el documento. Si el documento tiene la clave que se pretende renombrar, el operador la renombrar, ahora si no existe, no hace nada.

use ejemplos;

db.tema.remove({});

db.tema.insert({ \_id: 1, nobre: "take on me", grup: "a-ha"});

db.tema.find({\_id:1});

{ "\_id" : 1, "nobre" : "take on me", "grup" : "a-ha" }

db.tema.update( { \_id: 1 }, { $rename: { "nobre": "nombre", "grup": "banda" } } )

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

db.tema.find({\_id:1});

{ "\_id" : 1, "nombre" : "take on me", "banda" : "a-ha" }

db.tema.update( { \_id: 1 }, { $rename: { "anio": "año"} } )

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

db.tema.find({\_id:1});

{ "\_id" : 1, "nombre" : "take on me", "banda" : "a-ha" }

db.tema.update( { \_id: 1 }, {$set : { "album.name": "Hunting High and Low", "album.anio": 1985}});

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

db.tema.find({\_id:1}).pretty();

{

"\_id" : 1,

"nombre" : "take on me",

"banda" : "a-ha",

"album" : {

"name" : "Hunting High and Low",

"anio" : 1985

}

}

db.tema.update( { \_id: 1 }, { $rename: { "album.name": "album.nombre", "album.anio": "album.año" } } );

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

db.tema.find({\_id:1}).pretty();

{

"\_id" : 1,

"nombre" : "take on me",

"banda" : "a-ha",

"album" : {

"nombre" : "Hunting High and Low",

"año" : 1985

}

}

### $set

Para utilizar el update de la manera en que lo conocemos en cualquier RDBMS nos obliga a repetir toda la información del documento. Como podemos ver, no es una opción muy práctica sobre todo si consideramos que un documento puede ser complejo. Para ello, podemos usar el operador $set.

Y consideremos los siguientes updates:

use automoviles;

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact" }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

db.auto.update({\_id:1}, {$set : { año: 2012}});

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

db.auto.find({});

{ "\_id" : 1, "marca" : "peugeot", "modelo" : "207 compact", "año" : 2012 }

{ "\_id" : 2, "marca" : "ford", "modelo" : "fiesta 2017" }

{ "\_id" : 3, "marca" : "renault", "modelo" : "sandero" }

{ "\_id" : 4, "marca" : "chevrolet", "modelo" : "spark" }

{ "\_id" : 5, "marca" : "peugeot", "modelo" : "206" }

Si no se pusiera el operador $set en el update, el segundo documento reemplaza completamente al primero.

Si la clave que se especifica en $set no existe en el documento en cuestión, entonces se agrega, lo que hace del update una herramienta muy útil para insertar nuevas claves fácilmente.

use ejemplos;

db.tema.remove({});

db.tema.insert({ \_id:1});

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

db.tema.update( {\_id: 1} , { $set :{ nombre: "bara bará" } } );

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

db.tema.update( {\_id: 1} , { $set :{ cantante: "cristiano araujo" } } );

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

db.tema.update( {\_id: 1} , { $set :{ "tema.album": "ao vivo em goiania", "tema.año": 2012}})

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

db.tema.update( {\_id: 1} , { $set :{ "tema.invitados": "bruno e marrone"}})

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

db.tema.update( {\_id: 1} , { $set :{ tipo: "sertanejo" } } );

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

db.tema.find().pretty();

{

"\_id" : 1,

"nombre" : "bara bará",

"cantante" : "cristiano araujo",

"tema" : {

"album" : "ao vivo em goiania",

"año" : 2012,

"invitados" : "bruno e marrone"

},

"tipo" : "sertanejo"

}

### $unset

Continuamos con el ejemplo anterior:

db.tema.update( {\_id: 1} , { $unset :{ tipo: "sertanejo" } } );

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

db.tema.update( {\_id: 1} , { $unset :{ "tema.invitados": "rojo rey de copas" }})

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

db.tema.update( {\_id: 1} , { $unset :{ "tema.album": "rojo rey de copas" }})

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

db.tema.update( {\_id: 1} , { $unset :{ "tema.año": "rojo rey de copas" }})

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

db.tema.update( {\_id: 1} , { $unset :{ "tema": "rojo rey de copas" }})

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

db.tema.update( {\_id: 1} , { $unset :{ "cantante": "rojo rey de copas" }})

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

db.tema.update( {\_id: 1} , { $unset :{ "nombre": "rojo rey de copas" }})

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

db.tema.update( {\_id: 1} , { $unset :{ "\_id": "rojo rey de copas" }})

WriteResult({

"nMatched" : 0,

"nUpserted" : 0,

"nModified" : 0,

"writeError" : {

"code" : 66,

"errmsg" : "After applying the update to the document {\_id: 1.0

, ...}, the (immutable) field '\_id' was found to have been altered to \_id: ObjectId('5a5a5661932942700319bb18')"

}

})

### $setOnInsert

No se puede modificar la clave en un update. Veamos un ejemplo:

use ejemplos;

db.tema.remove({});

db.tema.update(

{\_id: 1} ,

{

$set :{ nombre: "não precisa mudar" },

$setOnInsert: { canta: "ivete sangalo" }

},

{ upsert: true }

);

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

db.tema.update(

{\_id: 1} ,

{

$set :{ nombre: " tche tche rere" },

$setOnInsert: { canta: "ivete sangalo" }

},

{ upsert: true }

);

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

db.tema.find().pretty();

{ "\_id" : 1, "nombre" : " tche tche rere", "canta" : "ivete sangalo" }

db.tema.update(

{\_id: 1} ,

{

$set: { canta: "gusttavo lima" }

},

{ upsert: true }

);

db.tema.find().pretty();

{ "\_id" : 1, "nombre" : " tche tche rere", "canta" : "gusttavo lima" }

## Arrays

### $ (update)

El operador $ busca un elemento del array y lo actualiza sin especificar explícitamente la posición del elemento en el array. Modifica el primer documento que coincide con la condición. Es útil utilizarlo cuando se quiere actualizar un único elemento y no se sabe la posición en un array.

use ejemplos;

db.alumno.remove({});

db.alumno.insert( [

{\_id: 1, notas: [ 8, 3, 7, 9, 3 ] },

{\_id: 2, notas: [ 4, 5, 1, 1, 3 ] },

{\_id: 3, notas: [ 9, 10, 8, 4, 8 ] },

]);

BulkWriteResult({

"writeErrors" : [ ],

"writeConcernErrors" : [ ],

"nInserted" : 3,

"nUpserted" : 0,

"nMatched" : 0,

"nModified" : 0,

"nRemoved" : 0,

"upserted" : [ ]

})

db.alumno.find();

{ "\_id" : 1, "notas" : [ 8, 3, 7, 9, 3 ] }

{ "\_id" : 2, "notas" : [ 4, 5, 1, 1, 3 ] }

{ "\_id" : 3, "notas" : [ 9, 10, 8, 4, 8 ] }

db.alumno.update(

{ \_id: 1, notas: 3},

{ $set: { "notas.$" :4 } }

);

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

db.alumno.find();

{ "\_id" : 1, "notas" : [ 8, 4, 7, 9, 3 ] }

{ "\_id" : 2, "notas" : [ 4, 5, 1, 1, 3 ] }

{ "\_id" : 3, "notas" : [ 9, 10, 8, 4, 8 ] }

db.alumno.update(

{ \_id: 1, notas: 3},

{ $set: { "notas.$" :4 } }

);

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

db.alumno.update(

{ \_id: 1, notas: 3},

{ $set: { "notas.$" :4 } }

);

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

db.alumno.find();

{ "\_id" : 1, "notas" : [ 8, 4, 7, 9, 4 ] }

{ "\_id" : 2, "notas" : [ 4, 5, 1, 1, 3 ] }

{ "\_id" : 3, "notas" : [ 9, 10, 8, 4, 8 ] }

db.alumno.update( { \_id: 2, notas: 1}, { $set: { "notas.$" :4 } });

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

db.alumno.update( { \_id: 3, notas: 4}, { $set: { "notas.$" :7 } });

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

db.alumno.find();

{ "\_id" : 1, "notas" : [ 8, 4, 7, 9, 4 ] }

{ "\_id" : 2, "notas" : [ 4, 5, 4, 1, 3 ] }

{ "\_id" : 3, "notas" : [ 9, 10, 8, 7, 8 ] }

### $addToSet

#### $each

El operador $addToSet agrega un valor a un array a menos que el valor ya exista dentro del array, en cuyo caso $addToSet no hará nada.

El operador solo asegura que no se van a insertar elementos duplicados pero no asegura que no los haya en el array. Si la clave no es un array, la operación falla. Si el valor es un array, el operador lo agrega.

use ejemplos;

db.letra.remove({});

db.letra.insert({ \_id: 1, letras: ["a", "b"] });

db.letra.update(

{ \_id: 1 },

{ $addToSet: {letras: {$each: [ "c", "d" ] }} }

);

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

db.letra.find();

{ "\_id" : 1, "letras" : [ "a", "b", "c", "d" ] }

db.letra.update(

{ \_id: 1 },

{ $addToSet: {letras: [ "d", "e" ] } }

);

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

db.letra.find();

{ "\_id" : 1, "letras" : [ "a", "b", "c", "d", [ "d", "e" ] ] }

db.letra.update( { \_id: 1 }, { $addToSet: {letras: [ "d", "e" ] } } );

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

db.letra.find();

{ "\_id" : 1, "letras" : [ "a", "b", "c", "d", [ "d", "e" ] ] }

### $pop

El operador $pop elimina el primer o el último elemento de un array. Se le puede pasar como parámetro alguno de los siguientes valores:

**1** ⇒ elimina el primer elemento del array.

**-1** ⇒ elimina el último elemento del array.

use ejemplos;

db.superliga.remove({});

db.superliga.insert({\_id:1,equipos:["boca", "independiente", "defensa y justicia", "racing", "river"] });

db.superliga.insert({ \_id: 2, equipos: "independiente campeón de la sudamericana 2017"});

Eliminamos el primer elemento de array:

db.superliga.update( { \_id: 1 }, { $pop: { equipos: 1 } } )

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

db.superliga.find();

{ "\_id" : 1, "equipos" : [ "independiente", "defensa y justicia", "racing", "river" ] }

Eliminamos el último elemento de array:

db.superliga.update( { \_id: 1 }, { $pop: { equipos: -1 } } )

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

db.superliga.find();

{ "\_id" : 1, "equipos" : [ "independiente", "defensa y justicia", "racing" ] }

¿Qué sucede cuando no es un array lo que le pasamos al operador $pop?

db.superliga.update( { \_id: 2 }, { $pop: { equipos: 1 } } )

WriteResult({

"nMatched" : 0,

"nUpserted" : 0,

"nModified" : 0,

"writeError" : {

"code" : 16837,

"errmsg" : "Can only $pop from arrays. {\_id: 2.0} has the field

'equipos' of non-array type String"

}

})

### $pull

El operador $pull elimina todas las instancias existentes de un array que coincidan con la condición.

Si se especifica una condición y el array tiene documentos embebidos, entonces el operador $pull aplica la condición incluyendo a los documentos embebidos.

En el caso de querer eliminar un array,

el operador respeta hasta el orden de los elementos, es decir, debe ser exacto.

Si lo que queremos eliminar es un documento, entonces $pull elimina los elementos en el array que tienen exactamente los mismos claves y valores. El orden de las claves pueden diferir.

Eliminar todos los elementos que coinciden con un valor específico:

use ejemplos;

db.frutas.remove({});

db.frutas.insert({\_id:1, fruta: ["manzana", "pera", "mandarina", "fresa", "membrillo"]});

WriteResult({ "nInserted" : 1 })

db.frutas.insert({\_id:2, fruta: ["arandanos", "pera", "frutilla", "fresa", "mango", "higo"]});

WriteResult({ "nInserted" : 1 })

db.frutas.insert({\_id:3, fruta: ["ciruela", "coco", "frutilla", "fresa", "avellana", "almendra"]});

WriteResult({ "nInserted" : 1 })

db.frutas.update({}, {$pull: { fruta: "manzana"}});

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

db.frutas.find({});

{ "\_id" : 1, "fruta" : [ "pera", "mandarina", "fresa", "membrillo" ] }

{ "\_id" : 2, "fruta" : [ "arandanos", "pera", "frutilla", "fresa", "mango", "higo" ] }

{ "\_id" : 3, "fruta" : [ "ciruela", "coco", "frutilla", "fresa", "avellana", "almendra" ] }

db.frutas.update({}, {$pull: { fruta: "pera"}});

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

db.frutas.find({});

{ "\_id" : 1, "fruta" : [ "mandarina", "fresa", "membrillo" ] }

{ "\_id" : 2, "fruta" : [ "arandanos", "pera", "frutilla", "fresa", "mango", "higo" ] }

{ "\_id" : 3, "fruta" : [ "ciruela", "coco", "frutilla", "fresa", "avellana", "almendra" ] }

db.frutas.update({}, {$pull: { fruta: "frutilla"}}, {multi:true} );

WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 2 })

db.frutas.find({});

{ "\_id" : 1, "fruta" : [ "mandarina", "fresa", "membrillo" ] }

{ "\_id" : 2, "fruta" : [ "arandanos", "pera", "fresa", "mango", "higo" ] }

{ "\_id" : 3, "fruta" : [ "ciruela", "coco", "fresa", "avellana", "almendra" ] }

db.frutas.remove({});

WriteResult({ "nRemoved" : 3 })

db.frutas.insert ( {

\_id: 1,

frutas: [ "manzana", "pera", "naranja", "uva", "mandarina" ],

hortalizas: [ "brocoli", "alcachofa", "lechuga", "calabaza" ]

}) ;

WriteResult({ "nInserted" : 1 })

db.frutas.insert ( {

\_id: 2,

frutas: [ "melon", "sandia", "coco", "kiwi", "mango" ],

hortalizas: [ "arveja" , "maiz", "acelga", "apio" ]

}) ;

WriteResult({ "nInserted" : 1 })

db.frutas.update(

{ },

{ $pull: { frutas: { $in: [ "naranja", "manzana" ] }, hortalizas: "brocoli" } },

{ multi: true }

) ;

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 1 })

db.frutas.find({});

{ "\_id" : 1, "frutas" : [ "pera", "uva", "mandarina" ], "hortalizas" : [ "alcachofa", "lechuga", "calabaza" ] }

{ "\_id" : 2, "frutas" : [ "melon", "sandia", "coco", "kiwi", "mango" ], "hortalizas" : [ "arveja", "maiz", "acelga", "apio" ] }

Eliminar todos los elementos que coinciden con una condición

use ejemplos;

db.numeros.remove({});

db.numeros.insert({\_id:1, numero: [ 0, 1, 2, 4, 6, 8, 10]});

WriteResult({ "nInserted" : 1 })

db.numeros.update({\_id:1}, {$pull: {numero: {$gte: 5}}});

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

db.numeros.find({});

{ "\_id" : 1, "numero" : [ 0, 1, 2, 4 ] }

Eliminar elementos de un array de documentos:

use ejemplos;

db.encuesta.remove({});

db.encuesta.insert ( {

\_id: 1,

resultados: [

{ producto: "A", calificación: 5 },

{ producto: "B", calificación: 8, comentario: "totalmente de acuerdo" }

]

} );

WriteResult({ "nInserted" : 1 })

db.encuesta.insert ( {

\_id: 2,

resultados: [

{ producto: "C", calificación: 8, comentario: "totalmente de acuerdo" },

{ producto: "A", calificación: 4 },

]

} );

WriteResult({ "nInserted" : 1 })

db.encuesta.find().pretty();

{

"\_id" : 1,

"resultados" : [

{

"producto" : "A",

"calificación" : 5

},

{

"producto" : "B",

"calificación" : 8,

"comentario" : "totalmente de acuerdo"

}

]

}

{

"\_id" : 2,

"resultados" : [

{

"producto" : "C",

"calificación" : 8,

"comentario" : "totalmente de acuerdo"

},

{

"producto" : "A",

"calificación" : 4

}

]

}

La siguiente operación eliminará del array resultados, todos los elementos que contienen tanto una calificación igual a 8 y además la clave producto es igual a "B".

db.encuesta.update(

{ },

{ $pull: { resultados: { calificación: 8 , producto: "B" } } },

{ multi: true }

) ;

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 1 })

db.encuesta.find().pretty();

{

"\_id" : 1,

"resultados" : [

{

"producto" : "A",

"calificación" : 5

}

]

}

{

"\_id" : 2,

"resultados" : [

{

"producto" : "C",

"calificación" : 8,

"comentario" : "totalmente de acuerdo"

},

{

"producto" : "A",

"calificación" : 4

}

]

}

Supongamos ahora los siguientes documentos:

use ejemplos;

db.encuesta.remove({});

db.encuesta.insert( {

\_id: 1,

resultados: [

{ parte: "A", calificación: 5, respuestas: [ { p: 1, r: 4 }, { p: 2, r: 6 } ] },

{ parte: "B", calificación: 8, respuestas: [ { p: 1, r: 8 }, { p: 2, r: 9 } ] }

]

} );

db.encuesta.insert( {

\_id: 2,

resultados: [

{ parte: "C", calificación: 8, respuestas: [ { p: 1, r: 8 }, { p: 2, r: 7 } ] },

{ parte: "B", calificación: 4, respuestas: [ { p: 1, r: 0 }, { p: 2, r: 8 } ] },

]

} );

db.encuesta.update(

{ },

{ $pull: { resultados: { respuestas: { $elemMatch: { p: 2, r: { $gte: 8 } } } } } },

{ multi: true }

) ;

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.encuesta.find();

{ "\_id" : 1, "resultados" : [ { "parte" : "A", "calificación" : 5, "respuestas" : [ { "p" : 1, "r" : 4 }, { "p" : 2, "r" : 6 } ] } ] }

{ "\_id" : 2, "resultados" : [ { "parte" : "C", "calificación" : 8, "respuestas" : [ { "p" : 1, "r" : 8 }, { "p" : 2, "r" : 7 } ] } ] }

### $push

El operador $push adiciona una clave:valor específico a un array. Si no existe, lo crea.

use ejemplos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:1, notas: [7, 8] });

db.alumnos.insert( {\_id:2, notas: [4, 5] });

db.alumnos.update(

{ \_id: 1 },

{ $push: { notas: 7 } }

);

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

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 7, 8, 7 ] }

{ "\_id" : 2, "notas" : [ 4, 5 ] }

db.alumnos.update({ \_id: 1 },{ $push:{ nombre: "pablo" } } );

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

db.alumnos.update({ \_id: 2 },{ $push:{ nombre: "marcelo" } } );

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

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 7, 8, 7 ], "nombre" : [ "pablo" ] }

{ "\_id" : 2, "notas" : [ 4, 5 ], "nombre" : [ "marcelo" ] }

db.alumnos.update({ \_id: 2 },{ $push:{ nombre: "martin" } } );

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

db.alumnos.update({ \_id: 2 },{ $push:{ nombre: "marcelo" } } );

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

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 7, 8, 7 ], "nombre" : [ "pablo" ] }

{ "\_id" : 2, "notas" : [ 4, 5 ], "nombre" : [ "marcelo", "martin", "marcelo" ] }

db.alumnos.update({},{ $push:{ promedio: 7 } } );

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

db.alumnos.find();

{ "\_id" : 2, "notas" : [ 4, 5 ], "nombre" : [ "marcelo", "martin", "marcelo" ] }

{ "\_id" : 1, "notas" : [ 7, 8, 7 ], "nombre" : [ "pablo" ], "promedio" : [ 7 ] }

db.alumnos.update({},{ $push:{ promociona: "si" } }, {multi:true} );

WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 7, 8, 7 ], "nombre" : [ "pablo" ], "promedio" : [ 7 ],

"promociona" : [ "si" ] }

{ "\_id" : 2, "notas" : [ 4, 5 ], "nombre" : [ "marcelo", "martin", "marcelo" ],

"promociona" : [ "si" ] }

Agregar muchos valores a un array

use ejemplos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:1, notas: [7, 8] });

WriteResult({ "nInserted" : 1 })

db.alumnos.insert( {\_id:2, notas: [4, 5] });

WriteResult({ "nInserted" : 1 })

db.alumnos.update({\_id: 1 }, { $push: { notas: { $each: [ 9, 9, 6 ] } } } )

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

db.alumnos.update({\_id: 2 }, { $push: { notas: [ 9, 9, 6 ] } } )

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

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 7, 8, 9, 9, 6 ] }

{ "\_id" : 2, "notas" : [ 4, 5, [ 9, 9, 6 ] ] }

Utilizando el operador $push con múltiples modificadores.

db.alumnos.insert( {\_id:5, notas: [ {examen: 1, nota: 10}, {examen: 2, nota: 8

}, {examen: 3, nota: 5}, {examen: 4, nota:6}] });

db.alumnos.find().pretty();

{ "\_id" : 1, "notas" : [ 7, 8, 9, 9, 6 ] }

{ "\_id" : 2, "notas" : [ 4, 5, [ 9, 9, 6 ] ] }

{

"\_id" : 5,

"notas" : [

{

"examen" : 1,

"nota" : 10

},

{

"examen" : 2,

"nota" : 8

},

{

"examen" : 3,

"nota" : 5

},

{

"examen" : 4,

"nota" : 6

}

]

}

db.alumnos.update(

{ \_id: 5 },

{

$push: {

notas: {

$each: [ { examen: 5, nota: 8 }, { examen: 6, nota: 7 }, { examen:

7, nota: 6 } ],

$sort: { nota: -1 },

$slice: 3

}

}

}

);

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

db.alumnos.find().pretty();

{ "\_id" : 1, "notas" : [ 7, 8, 9, 9, 6 ] }

{ "\_id" : 2, "notas" : [ 4, 5, [ 9, 9, 6 ] ] }

{

"\_id" : 5,

"notas" : [

{

"examen" : 1,

"nota" : 10

},

{

"examen" : 2,

"nota" : 8

},

{

"examen" : 5,

"nota" : 8

}

]

}

El update anterior usa los siguientes operadores:

$each ⇒ agregar múltiples elementos al array.  
$sort ⇒ ordena todos los elementos del array modificados por notas en orden descendente.  
$slice ⇒ mantiene solo los primeros tres elementos ordenados del array de notas.

### $pullAll

El operador $pullAll elimina todas las instancias de los valores especificados el array existente. A diferencia del operador $pull que elimina elementos al especificar una consulta, $pullAll elimina los elementos que coinciden con los valores listados.

use ejemplos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:1, notas: [ 3, 4, 7, 3, 8, 4, 9, 4 ] });

db.alumnos.update( { \_id: 1 }, { $pullAll: { notas: [ 4, 3 ] } } );

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 7, 8, 9 ] }

### $each

El operador $each está disponible y podemos combinarlo con los operadores: $addToSet ([ver ejemplo](#_d5d57f5ili10)) y $push ([ver ejemplo](#_rmkc671f8xkr)).

### $position

use ejemplos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:1, notas: [4, 8] });

db.alumnos.insert( {\_id:2, notas: [4, 9] });

Agregar elementos al comienzo del array.

db.alumnos.update(

{ \_id: 1 },

{

$push: {

notas: {

$each: [ 1, 2, 3 ],

$position: 0

}

}

}

)

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

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 1, 2, 3, 4, 8 ] }

{ "\_id" : 2, "notas" : [ 4, 9 ] }

Agregar elementos en el medio del array.

db.alumnos.update(

{ \_id: 1 },

{

$push: {

notas: {

$each: [ 5, 6, 7 ],

$position: 4

}

}

}

)

db.alumnos.find();

{ "\_id" : 2, "notas" : [ 4, 9 ] }

{ "\_id" : 1, "notas" : [ 1, 2, 3, 4, 5, 6, 7, 8 ] }

### $slice

El operador $slice limita el número de elementos del array durante una operación de $push.  
Para poder usar el modificador $slice, debe aparecer junto con $each.

Cortar el array desde el comienzo

use ejemplos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:1, notas: [4, 5, 6] });

db.alumnos.update(

{ \_id: 1 },

{

$push: {

notas: {

$each: [ 7, 8, 9 ],

$slice: -5

}

}

}

);

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 5, 6, 7, 8, 9 ] }

Cortar el array desde el final

use ejemplos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:1, notas: [5, 6] });

db.alumnos.update(

{ \_id: 1 },

{

$push: {

notas: {

$each: [ 2, 3, 4 ],

$slice: 3

}

}

}

);

db.alumnos.find();

{ "\_id" : 1, "notas" : [ 5, 6, 2 ] }

Actualizar un array usando solo $slide

use alumnos;

db.alumnos.remove({});

db.alumnos.insert( {\_id:4, notas: [4, 5, 6, 7] });

WriteResult({ "nInserted" : 1 })

db.alumnos.update(

{ \_id: 4 },

{

$push: {

notas: {

$each: [ ],

$slice: -3

}

}

}

);

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

db.alumnos.find();

{ "\_id" : 4, "notas" : [ 5, 6, 7 ] }

# **Fechas**

En el caso de querer registrar la fecha del día, podemos usar la siguiente función:

{

fecha : new Date();

}

Veamos un ejemplo práctico:

use ejemplos;

db.ejemplo.remove({});

db.ejemplo.insert({

\_id : 1,

fecha : new Date(),

});

db.ejemplo.find ({}) {

"\_id" : 1,

"fecha" : ISODate("2017-12-30T00:25:19.007Z")

}

# **Remove**

Para borrar elementos de una colección se puede usar el método remove que recibe como parámetro una query y elimina todos los elementos de la colección que cumplen con dicha condición de búsqueda:.

Borra todos los documentos de la colección nombres de la base de datos ejemplos:

use ejemplos;

db.nombres.find({}).count();

5

db.nombres.remove({});

WriteResult({ "nRemoved" : 5 })

Borra el/los documentos (en este caso 3) que tienen como valor a "pablo" en la clave nombre de la colección nombres de la base de datos ejemplos:

use ejemplos;

db.nombres.find({nombre:"pablo"}).count();

3

db.nombres.remove({nombre:"pablo"});

WriteResult({ "nRemoved" : 3 })

# **Drop**

El método drop se utiliza para borrar una colección de una base de datos. Este método borra también todos los indices asociados a dicha colección. Su ejecución puede retornar únicamente dos valores:

true → cuando se borra satisfactoriamente una colección.

false → cuando no existe la colección que se quiere borrar.

show collections;

alumno

alumnos

collection

ejemplo

letra

letras

nombres

producto

productos

students

superliga

system.indexes

tema

db.nombres.drop();

true

db.sosrojo.drop();

false