9 CRUD

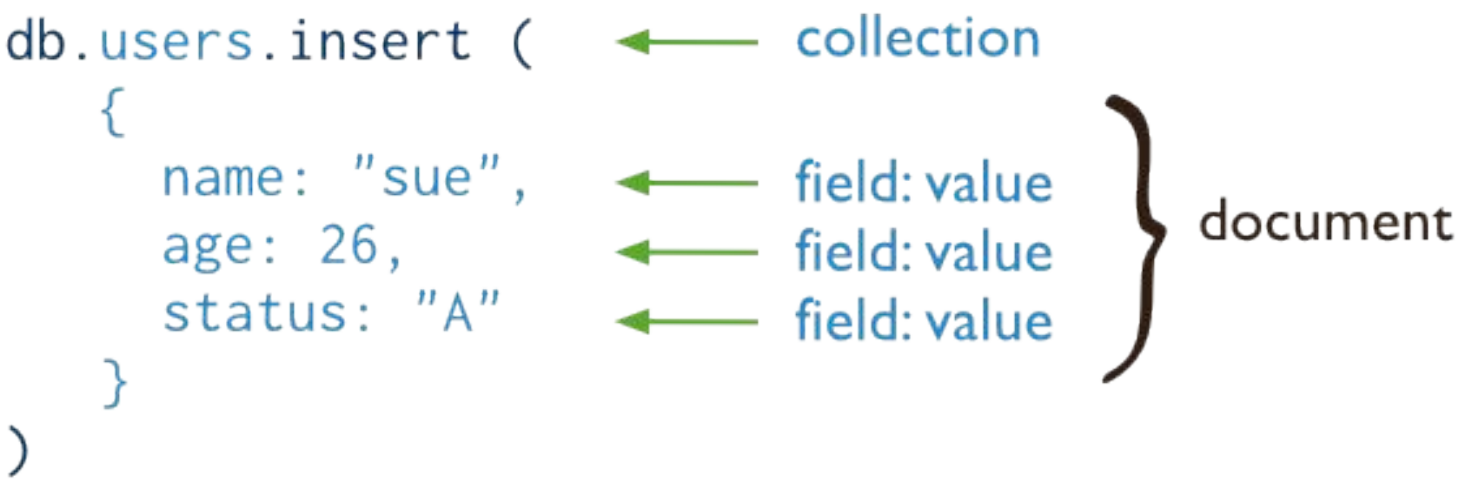
5 CRUB

1. CRUD
2. Inserción de datos
3. Consulta de datos
4. Actualizar información
5. Borrado de documentos

Las operaciones son métodos de una API: NO un lenguaje separado
Los parámetros son documentos **BSON**

OPERACIÓN	MONGO	SQL
Create	insert / save	INSERT
Read	find / findOne	SELECT
Update	update / findAndModiy / save	UPDATE
Delete	remove / drop	DELETE

Dot Notation: Se puede expresar un BSON no sólo con la notación "json" {a : {b :1}}, también se puede expresar como {'a.b' : 1}



ObjectID

Todo documento insertado en MongoDB tiene un campo **_id**, que identifica unívocamente el documento. Podemos darle cualquier valor al _id, por defecto nos genera un valor.

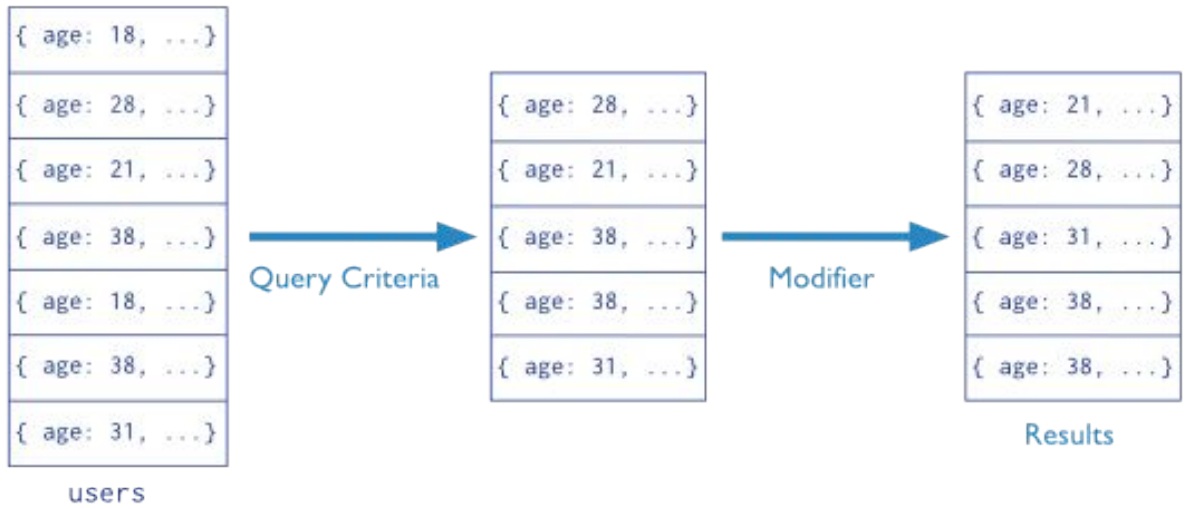
PRACTIQUEMOS

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

- collection
- query criteria
- projection
- cursor modifier

Collection Query Criteria Modifier

```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```



Insertemos

```
db.name.insert({a:"1"})
db.name.insert({a:"2", b : null})
db.name.insert({a:1, b:"1"})
db.name.insert({a:2, b:"1", texto:"hola"})
db.name.insert({a:"1", c:true})
db.name.insert({d:{f:1}})
```

Operadores de consulta (comparación)

- JSON

```
db.name.find({a:”1”}) ->
db.name.find({a:”1”,c:true}) ->
db.name.find({a:”1”,b:"1"}) ->
db.name.find({”d.f”:”1”}) ->
db.name.find({d : {f:1}}) ->
```

Operadores de consulta (comparación)

- $ne / $eq

```
db.name.find({a:{$ne : "1"}}) -->  {a:1, b:"1"}, {a:2, b:"1", texto:"hola"},{a:"2"}
db.name.find({a:{$eq : ”1”}}) ->  {a:"1"} , {a:"1", c:true}
```

- $gt / $gte / $lt / lte :

```
db.name.find({a:{$gt : 1}}) -->  {a:2, b:"1", texto:"hola"}
db.name.find({a:{$gt : "1"}}) -->  {a:"2"}
```

- $in / $nin

```
db.name.find({a:{$in :[1,2] }}) -->  {a:1, b:"1"},{a:2, b:"1", texto:"hola"}
db.name.find({a:{$nin :[1] }}) --> {a:"1"},{a:"2"},{a:2, b:"1", texto:"hola"},{a:"1", c:true}
```

PRACTIQUEMOS

Operadores de consulta (comparación)

- $regex :
  - Buscar expresiones regulares (formato **PCRE**)

```
db.name.find({ a: { $regex: /HO.*/i} })
db.name.find({ a: { $regex: /la/, $options: 'i'} })
db.name.find({ a: { $regex: '^h' } })
```

- $text :
  - Búsqueda sobre índices de texto
  - `{ $text: { $search: <string>, $language: <string> } }`
- $exist
  - Devuelve los documentos en los que exista el campo

```
db.name.find({b:{$exists:true}})->{a:"2", b:null},{a:1,b:"1"},{a:2,b:"1",texto:"hola"}
```

Operadores de consulta (lógicos)

- $or :

```
db.name.find({$or : [{a :"1"},{a : {$gt : 1 }}, {c:true}] }) ->
   {a:"1"},{a:2, b:"1", texto:"hola"},{a:"1", c:true}
```

- $and

```
db.name.find({$and : [{a :"1"},{a : {$regex : "^1" }}] }) ->
  {a:"1"},{a:"1", c:true}
```

- $not

```
db.name.find({a : {$gt : 1 } }) ->  {a:2, b:"1", texto:"hola"}
db.name.find({a : {$not : {$gt : 1 } }}) ->  {a:"1"},{a:"2", b : null},{a:1, b:"1"},{a:"1", c:true}
```

- $nor

```
db.name.find({$nor : [{a :"1"},{a : {$gt : 1 }}, {c:true}] }) ->
   {a:"2", b : null}, {a:1, b:"1"}
```

Operadores de consulta (arrays)

```
db.name.insert({a:["1",2]})
db.name.insert({a:[{b:"2"},{b:"3"}]})
```

- Consultas sobre arrays:

```
db.name.find({a:"1"}) -> {a:["1",2]}
db.name.find({"a.b":"2"}) -> {a:[{b:"2"},{b:"3"}]}
db.name.find({'a.1':2}) -> {a:["1",2]}
db.name.find({a:{$gt :1}}) -> {a:["1",2]}
```

- $elementMatch: devuelve los documentos en los que al **menos uno** de los elementos del array satisfaga las condiciones

```
db.name.find({a: {$elemMatch:{$gt :1}}}) -> {a:["1",2]}
```

- $all: devuelve los documentos en los que **todos** los elementos del array satisfagan las condiciones

```
db.name.find({a : {$all:["1"]}}) -> {}
db.name.find({a:{$all:[{$or : [{'a.b':"2"},{'a.b':"3"}]}]}}) -> a:[{b:"2"},{b:"3"}]}
```

Otros operadores de consulta

- Búsqueda geoespacial:
- $type: búsqueda por el tipo de dato
- $mod: valida el campo de acuerdo con una operación de módulo
- $where: valida el campo contra una expresión javascript
- $comment: para añadir comentarios en las queries

Projections

Nos permite filtrar el contenido de los documentos de salida de una consulta.

```
db.name.find({},{a:1}) -> { "_id" : ObjectId("5...5"), "a" : "1" },{ "_id" : ObjectId("5...6"), "a" : "2" },{ "_id" :
ObjectId("5...7"), "a" : 1 },{ "_id" : ObjectId("5...8"), "a" : 2 },{ "_id" : ObjectId("5...9"), "a" : "1" }
```

```
db.name.find({},{a:0}) -> { "_id" : ObjectId("5...5") },{ "_id" : ObjectId("5...6"), "b" : null },{ "_id" : ObjectId
("5...7"), "b" : "1" },{ "_id" : ObjectId("5...8"), "b" : "1", "texto" : "hola" },{ "_id" : ObjectId("5...9"), "c" :
true }
```

El campo **_id** siempre sale... salvo que indiquemos lo contrario

```
db.name.find({},{a:1,_id:0}) -> {"a" : "1"},{"a" : "2"},{"a" : 1},{"a" : 2},{"a" : "1"}
```

Operadores projections:

- $: primer elemento del array que coincide con la query
- $elemMatch : primer elemento del array que coincide con la condición del elemMatch
- $slice: reduce el tamaño del array

Cursor

La salida de una consulta find es siempre un cursor que es necesario iterar para recuperar toda la información.

Estos métodos se puede ejecutar sobre el cursor:

- it : nos devuelve una “página” de la consulta
- next () : nos devuelve el siguiente elemento del cursor
- hasNext () : nos indica si hay más elementos del cursor

Utilizando los cursores (y código javascript) podemos manipular el resultado de una query para realizar operaciones complejas.

Otros métodos del cursor

- count () : cuenta el número de elementos del cursor
- pretty () : formatea la salida del cursor

Recorrer un cursor

```
var cursor = db.grades.find();

while (cursor.hasNext()){
    var grade = cursor.next();
    printjson (grade);
}
```

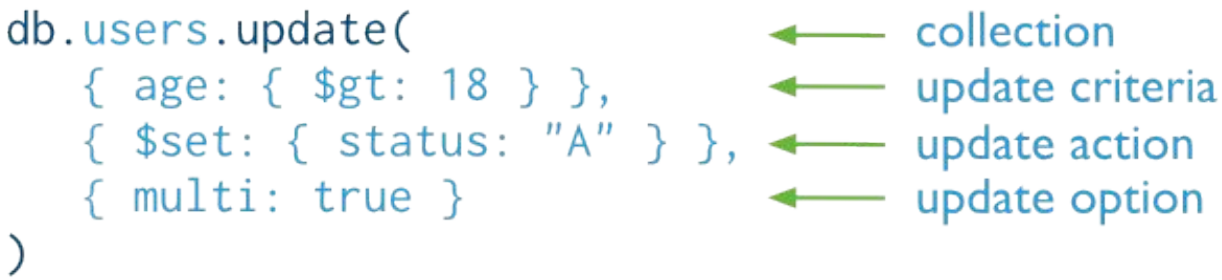
Paginación de resultados

- limit(<num>) -> denota el número máximo de elementos de una query.
- skip(<num>) -> devuelve documento ignorando los <num> primeros.
- sort({<field1>: -1|1, <field2>: -1|1...}) -> ordena los resultados de la consulta.

```
db.name.find({}).limit(2).skip(1).sort({a:-1}) -> {"a":"1"}, {"a":"1", "c":true}
```

Aunque son métodos, tienen que **invocarse** al ejecutar el find.

El orden es indiferente.



Update Criteria

En el criterio de búsqueda se pueden aplicar todos los operadores de búsqueda con la misma sintaxis.

Update action

La actualización a ejecutar. Formato BSON.

Operadores de actualización (campos)

- **JSON** : reemplaza el documento que coincida por el documento dado

```
{a:"1"} -> db.name.update ({a:"1"},{b:"2",c:"3"}) -> {"b" : "2", "c" : "3" }
```

- **$inc** : incrementa el valor de un campo sumándole un valor

```
{a:1} -> db.name.update ({a:1},{ $inc:{a : 1}}) -> {"a" : 2 }
```

- **$mul** : incrementa el valor de un campo multiplicándose por un valor

```
{a:2} -> db.name.update ({a:2},{ $mul:{a : 10}}) -> {"a" : 20 }
```

- **$rename** : cambia el nombre de un campo

```
{a:2} -> db.name.update ({a:2},{ $rename:{a : z}}) -> {z : 2}
```

- **$unset** : elimina el campo

```
{a:2,b:1} -> db.name.update ({a:2},{ $unset:{b:""}}) -> {a : 2}
```

Operadores de actualización (campos)

- **$set : modifica el valor de un campo por otro dado**

```
{a:"1",b:"2"} -> db.name.update ({},{$set : { c:"3"}}) -> {a:"1",b:"2",c:"3"}

{a:"1",b:"2"} -> db.name.update ({},{$set : { b:7}}) -> {a:"1",b:7}

{a:"1",b:"2"} -> db.name.update ({},{$set : { b:10,c:"1"}}) -> {a:"1",b:10,c:"1"}
```

Operadores de actualización (campos)

- Otros operadores:
  - **$min** : sólo actualiza si el valor dado es menor que el mínimo del valor actual
  - **$max** : sólo actualiza si el valor dado es mayor que el máximo del valor actual
  - **$currentDate** : actualiza en un campo la fecha actual
  - **$setOnInsert** : sólo añade el campo si se trata de una creación

PRACTIQUEMOS

Operadores de actualización (arrays)

- **$** : modificar la posición del array que coincide con el criterio de búsqueda. Sólo se puede indicar un $ por campo en la actualizació

```
{a:["1","2"],b:[{c:1}]} -> db.name.update ({a:"2"},{$set : { a.$:"3"}}) -> {a:["1","3"],b:[{c:1}]}

{a:["1","2"],b:[{c:1}]} -> db.name.update ({a:"2"},{$set : { a.0:"3"}}) -> {a:["3","2"],b:[{c:1}]}

{a:["1","2"],b:[{c:1}]} -> db.name.update ({a:"2"},{$set : { b.$:"3"}}) -> {a:["1","2"],b:[{c:1}, "3"]}

{a:["2","2"],b:[{c:1}]} -> db.name.update ({a:"2"},{$set : { a.$:"3"}}) -> {a:["3","2"],b:[{c:1}]}
```

- **$addToSet** : añade un valor al array sólo si el valor no existe

```
{a:["1","2"],b:[{c:1}]} -> db.name.update ({a:"2"},{ $addToSet: : {a : "3"}}) -> {a:["1","2","3"],b:[{c:
1}]}

{a:["1","2"],b:[{c:1}]} -> db.name.update ({a:"2"},{ $addToSet: : {a : "2"}}) -> {a:["1","2"],b:[{c:1}]}
```

Operadores de actualización (arrays)

- **$pop** : elimina el primer o el último elemento de un array

```
{a:["1","2"]} -> db.name.update ({a:"2"},{ $pop: : {a : -1}}) -> {a:["2"]}
{a:["1","2"]} -> db.name.update ({a:"2"},{ $pop: : {a : 1}}) -> {a:["1"]}
```

- **$pullAll** : elimina elementos del array que vengan como valor

```
{a:["1","2"]} -> db.name.update ({a:"2"},{$pullAll: : {a : ["3"]}}) -> {a:["1","2"]}
{a:["1","2","2"]} -> db.name.update ({a:"2"},{$pullAll: : {a : ["2"]}}) -> {a:["1"]}
{a:["1","2","2"]} -> db.name.update ({a:"2"},{$pullAll: : {a : ["2","1"]}}) -> {a:[]}
```

- **$pull** : elimina elementos del array que satisfagan las condiciones dadas

```
{a:["1","2"],b:["7"]} -> db.name.update ({a:"2"},{$pull: : {a : "3",b:"7"}}) ->  {a:["1","2"],b:[]}
{a:["1","2"],b:["7"]} -> db.name.update ({a:"2"},{$pull: : {a : {$in : ["3","2"],b:"7"}}) ->  {a:["1"],b:
[]}
```

Operadores de actualización (arrays)

- **$push** : añade un valor a un array

```
{a:["1","2"]} -> db.name.update ({a:"2"},{$push: : {a : "3"}}) -> {a:["1","2","3"]}
```

  - Modificadores de push:
    - **$each** : permite manipular múltiples items por actualización
    - **$slice** : modifica el tamaño del array en la actualización
    - **$sort** : ordena los elementos del array
    - **$position** : indica la posición para la modificación del array

Modificadores de actualización

- **upsert** : con este parámetro indicamos que si ningún elemento coincide con el criterio de búsqueda cree un nuevo documento en la colección.
- **multi** : por defecto el método update sólo actualiza un único documento, con este parámetro le indicamos que afecte a todos los documentos que cumplan con el criterio.

Otras operaciones de actualización

- **save** : inserta o actualiza un nuevo documento. Lo actualiza si encuentra coincidencia con el campo **_id** del documento enviado como parámetro

- **findAndModify** : actualiza y devuelve el documento actualizado.

Remove

```
db.users.remove(          <-- collection
   { status: "D" }        <-- remove criteria
)
```

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

Drop

Elimina una colección entera, incluyendo documentos e índices.

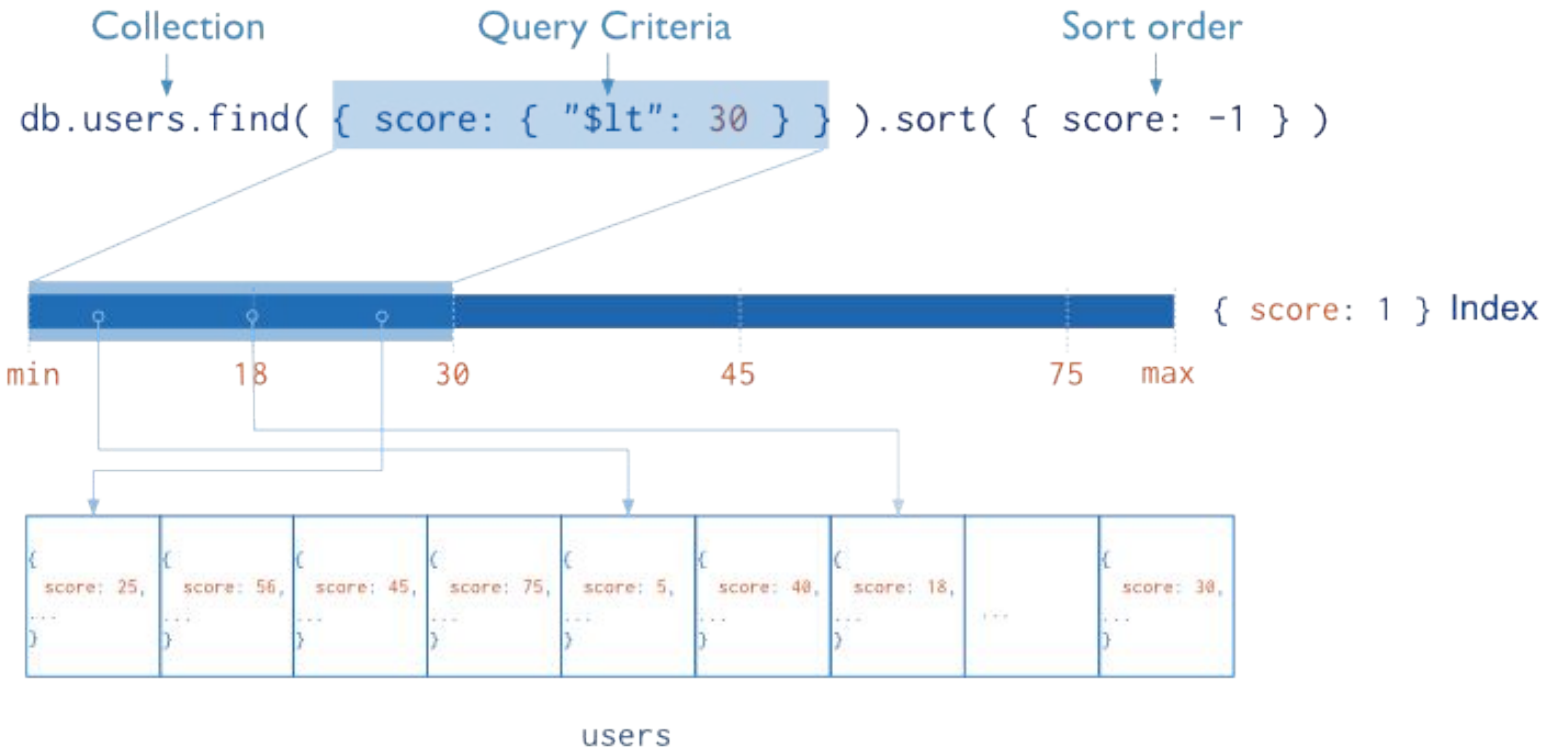
```
db.collection.drop()
```

10 Índices

7 Índices

1. Índice
2. Tipos de índice
3. Monoclave
4. Compuestos
5. Multikey
6. Otros
7. Covered query
8. Operaciones sobre índices

El concepto de índice que maneja MongoDB es el mismo que el que se maneja en otros repositorios de información.



Consideraciones generales

- Ofrecen grandes mejoras en las operaciones de lecturas
- Pueden penalizar las operaciones de escritura
- Se almacenan en memoria
- Por defecto se crea un índice único sobre el campo **_id**
- No sólo se indican los campos del índice sino el orden del mismo
- Los operadores de negación no utilizan índices
- Los índices los emplean las querys y las “agregaciones”

2 Tipos de índices

```
db.collection.createIndex( {  <field_1>: <index type>, <field_2>: <index type>  ... } {<options>})
```

Los índices pueden ser:

- **monoclave :** sólo indexan por un campo de los documentos de la colección
- **compuestos :** indexan por varios campos de los documentos de la colección
- **hashed :** indexan de forma clave valor
- **text :** índices de texto (tipo solr)
- **geoespaciales :** indexan por coordenadas espaciales

Según las opciones que se definan los índices pueden tener las propiedades:

- **unique :** sólo puede haber una correspondencia entre entrada del índice y documento
- **sparse :** no indexa campos con valores nulos
- **TTL**: elimina documentos de la colección cuando pasa cierto tiempo.

Índices monoclave

```
db.collection.createIndex( { <field>: <-1|1> } )
```

- Sólo afectan a un campo de búsqueda. Indicar -1 o 1 indica si el índice es ascendente o descendente para un campo es irrelevante.
- Índices sobre entidades embebidas: sólo se produce un acierto en el índice si las entidades son exactamente iguales, incluyendo orden.

```
Documento: { metro: {  city: "New York",  state: "NY"  }, name: "Giant Factory"}
Índice: db.factories.createIndex( { metro: 1 } )
Query 1: db.factories.find({metro:{city:"New York",state:"NY"}})  -> encuentra documento
Query 2: db.factories.find({metro:{state:"NY",city:"New York"}})  -> NO encuentra documento
```

Índices compuestos

```
db.collection.createIndex( { <field_1>: <-1|1>,<field_2>: <-1|1> } )
```

El -1 índica ordenación descendente y el 1 ordenación ascendente

```
Índice: db.name.createIndex( { a: 1, b : -1 } )

Query 1: db.factories.find({}).sort({a:1,b:-1}) -> se ejecuta contra el índice

Query 2: db.factories.find({}).sort({a:-1,b:-1}) -> NO se ejecuta contra el índice

Query 3: db.factories.find({}).sort({b:-1,a:1}) -> NO se ejecuta contra el índice

Query 4: db.factories.find({}).sort({a:1}) -> se ejecuta contra el índice

Query 5: db.factories.find({}).sort({a:-1}) -> se ejecuta contra el índice
```

Índices multiclave *

Son índices en los que un elemento del índice hace referencia un campo que tiene valores de array

Pueden ser índices de un único campo o compuestos

Se crean tantas entradas del índice como valores contenta el array.

Sólo uno de los elementos del índice puede tener valores de array.

Índices únicos

```
db.collection.createIndex( { <field>: <index_type> },{unique: <true|false>} )
```

Indicar que cada valor del índice debe corresponder con un único valor.

Se puede utiliar tanto para índices de un sólo campo o multicampo

Índices con valores nulos (sparse)

```
db.collection.createIndex( { <field>: <index_type> },{sparse: <true|false>} )
```

Es un índice de valores únicos que admite valores **nulos.**

Los valores nulos no se introducen en el índice. Una búsqueda que utilice este índice **no devolverá** documentos que no estén en el índice

```
Índice: db.factories.createIndex( { a: 1 },{sparse:true} )
```

```
Query: db.factories.find({a : null}).sort(a:-1)  -> NO encontrará ningún documento
```

COVERED QUERY: query que se resuelve contra un índice sin consultar datos de la colección.

```
Documento: { a: "hola", b : "adios", c:"saludo", _id:ObjectId(...) } )

Índice: db.name.createIndex( { a: 1, b : -1 } )

Query 1: db.factories.find({a:"hola", b:"adios"}, {a:1,b:1,_id:0}) -> covered

Query 2: db.factories.find({a:"hola", b:"adios"}, {a:1,b:1}) -> NO covered

Query 3: db.factories.find({a:"hola", b:"adios"}, {c:0,_id:0}) -> covered

Query 4: db.factories.find({a:"hola"},{a:1,b:1,_id:0}).sort({c:1}) -> NO covered

Query 5: db.factories.find({},{a:1,b:1,_id:0}) -> NO covered

Query 6: db.factories.find({a:"hola"},{a:1,b:1,_id:0}).sort( {a:-1,b:-1}) -> NO covered
```

Borrado de índice

```
db.accounts.dropIndex( { "tax-id": 1 } )
```

Regeneración del índice

```
db.accounts.reIndex()
```

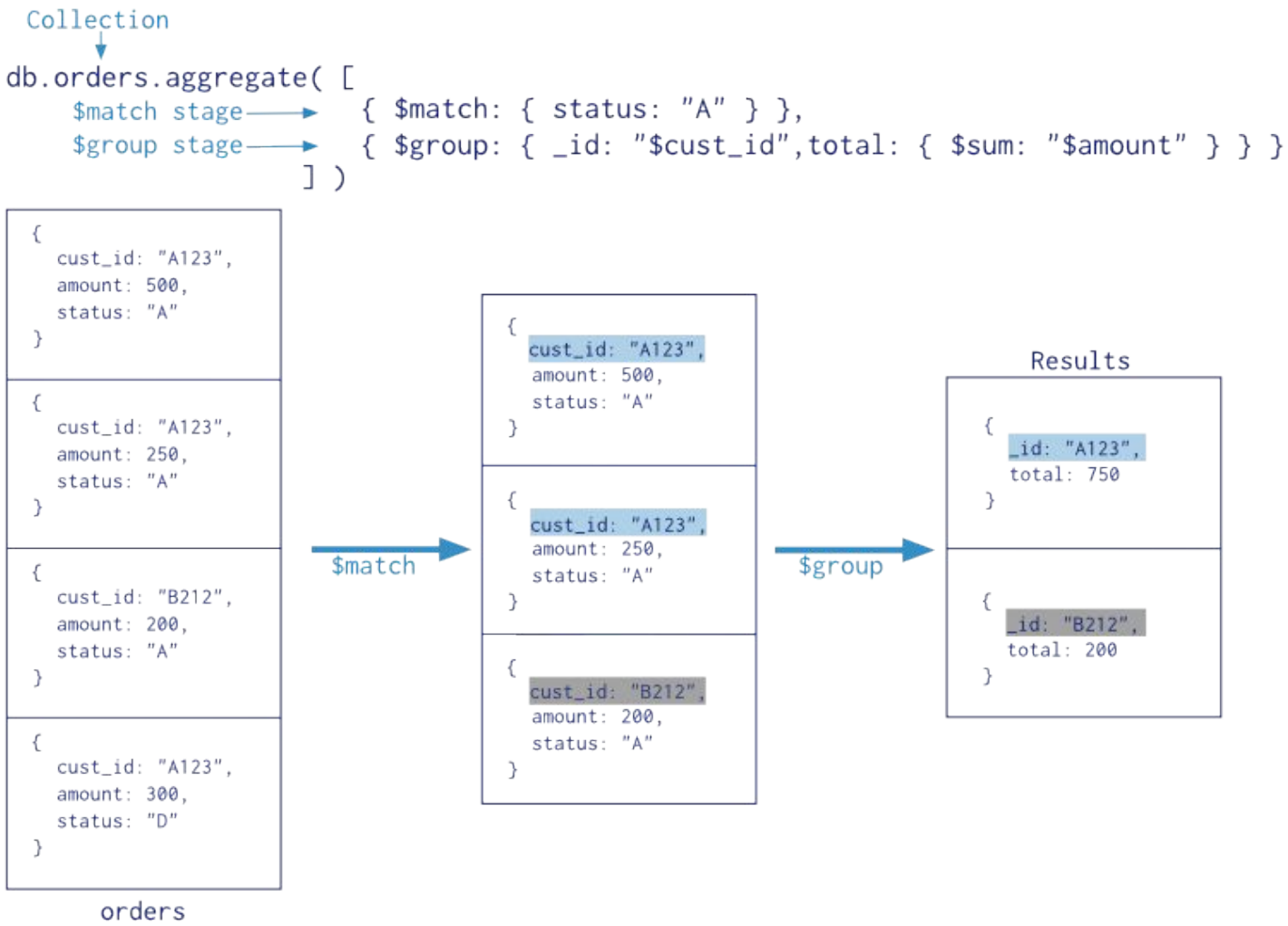
Listado de índices

```
db.accounts.getIndexes()
```

11 Agregaciones

6 Agregaciones

1. Agregaciones
2. Pipeline
3. Operadores
4. Operadores $group


Collection
db.orders.aggregate([
$match stage
{ $match: { status: "A" } },
$group stage
{ $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
])
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
$match
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
$group
Results
{ _id: "A123", total: 750 }
{ _id: "B212", total: 200 }

Nos permite definir una conjunto de operaciones que se ejecuten de manera secuencial sobre una colección.
Cada operación toma el conjunto de datos manipulados por la operación anterior. La primera operación se ejecuta sobre la colección que se indique.
La salida de la operación de agreggate es un **cursor**.

Según la operación se pueden dar los siguientes casos:

- **n** documentos de entrada -> **n** documentos de salida
- **n** documentos de entrada -> **m** documentos de salida (**n>m**)
- **n** documentos de entrada -> **m** documentos de salida (**m>n**)

$project

Manipula cada documento, permitiendo eliminar campos, añadir algunos o realizar transformaciones sobre campos existentes.
Es una operación que mantiene el mismo número de documentos.
Sintasix:

{$project : {<field>: <1 or true>} } -> añadir un campo del documento.

{$project : {<field>: <0 or false>} } -> elimina un campo del documento.

{$project : {<field>: <expression>} } -> añade un campo nuevo, o modifica el valor.

$project

```
{
    "city" : "ACMAR",
    "loc" : [
        -86.51557,
        33.584132
    ],
    "pop" : 6055,
    "state" : "AL",
    "_id" : "35004"
}
```



```
db.zips.aggregate([{$project
:                    {"_id":0,
   "city":1,
   "pop":1,
   "state":1,
   "zip":"_id"
}}])
```



```
{
    "city" : "acmar",
    "pop" : 6055,
    "state" : "AL",
    "zip" : "35004"
}
```

$match

Filtra los documentos en base a un criterio de búsqueda. El criterio de búsqueda se define igual que el criterio de un find.
Es una operación que disminuye el número de documentos.
Sintasix:

```
{ $match: { <query> } }
```

$sort

Ordena todos los documentos.
Es una operación que mantiene el mismo número de documentos.
Sintasix:

```
{ $sort: { <field1>: <1|-1>, <field2>: <1|-1> ... } } -> 1 denota orden ascendente y -1 descendente
```

$limit

Limita el número máximo de documentos que maneja el pipeline.
Es una operación que disminuye el número de documentos.
Sintasix:

```
{ $limit: <positive integer> }
```

$skip

Elimina los primeros n documentos que maneja el framwork.
Es una operación que disminuye el número de documentos.
Sintasix:

```
{ $skip: <positive integer> }
```

$unwind

Destruye un array creando tantos documentos nuevos como elementos tiene el array.
Es una operación que aumenta número de documentos.
Sintasix:

```
{ $unwind: <field path> }
```

```
{a: ["1","2"],b:"3"} -> db.names.aggregate ([{$unwind :  "$a"}]) -> {a: "1",b:"3"},{a: "2",b:"3"}
```

$out

Permite escribir la salida de la agregación en una colección.
Cuando se utilice tiene que ser el último comando.
Sintasix:

```
{ $out: "<output-collection>" }
```

$group

Agrupa documentos, en base a la coincidencia de valores de un campo y permite definir operaciones sobre cómo definir la información agregada.
Es una operación que disminuye el número de documentos.
Sintasix:

```
{ $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }
```

El campo _id es obligatorio y es el que indica los campos del documento sobre los que se realiza el agrupamiento:

- _id: "$<field>" -> agrupamos sobre un único campo
- _id: {<field_name_1>: "$<field_1>",<field_name_2>: "$<field_2>"} -> agrupamos sobre varios campos
- _id: null -> agrupamos sobre todos los datos de la agregación

Los acumuladores que se utilizan con la operación de **$group** son:

- **$sum** : suma todos los valores de un campo de todos los documentos

```
{a: "1",b:5},{a: "1",b:1},{a: "1"} ->
db.names.aggregate ([{$group :  {_id:"$a"},value:{$sum:"$b"}}]) -> {_id: "a",value:6}
db.names.aggregate ([{$group :  {_id:"$a"},value:{$sum:1}}]) -> {_id: "a",value:3}
```

- **$avg** : calcula la media de todos los valores de un campo de todos los documentos

```
{a: "1",b:5},{a: "1",b:1},{a: "1"} ->
db.names.aggregate ([{$group :  {_id:"$a"},value:{$avg:"$b"}}]) -> {_id: "a",value:2}
```

- **$first** : Coge el valor de un campo del primer documento que se agrega

```
{a: "1",b:5},{a: "1",b:1},{a: "1"} ->
db.names.aggregate ([{$sort : {b:1 }}{$group :  {_id:"$a"},value:{$first:"$b"}}]) -> {_id: "a",value:5}
```

Los acumuladores que se utilizan con la operación de $group son:

- **$last** : Coge el valor de un campo del último documento que se agrega

```
{a: "1",b:5},{a: "1",b:1},{a: "1"} ->
db.names.aggregate ([{$sort : {b:1 }},{$group :  {_id:"$a"},value:{$last:"$b"}}]) -> {_id: "a",value:null}
```

- **$max** : Coge el valor máximo de un campo de todos los documentos

```
{a: "1",b:5},{a: "1",b:1},{a: "1"} ->
db.names.aggregate ([{$group :  {_id:"$a"},value:{$max:"$b"}}]) -> {_id: "a",value:5}
```

- **$min** : Coge el valor mínimo de un campo de todos los documentos

```
{a: "1",b:5},{a: "1",b:1},{a: "1"} ->
db.names.aggregate ([{$group :  {_id:"$a"},value:{$min:"$b"}}]) -> {_id: "a",value:1}
```

Los acumuladores que se utilizan con la operación de $group son:

- **$push** : Añade a un array un valor por cada documento de la agregación.

```
{a: "1",b:5},{a: "1",b:1},{a: "1"},{a: "1",b:1} ->
db.names.aggregate ([{$group :  {_id:"$a"},value:{$push:"$b"}}]) -> {_id: "a",value:[5,1,1]}
```

- **$addToSet** : Añade a un array un valor por cada documento de la agregación, eliminando los valores repetidos.

```
{a: "1",b:5},{a: "1",b:1},{a: "1"},{a: "1",b:1} ->
db.names.aggregate ([{$group :  {_id:"$a"},value:{$push:"$b"}}]) -> {_id: "a",value:[5,1]}
```