

## INDICE

Elementos esenciales de MongoDB: Base De Datos - Colección – Documento	Pag. 2
Insertar documentos mediante los métodos insertOne e insertMany de una colección	Pag. 4
Campo obligatorio _id	Pag. 6
Problemas Propuestos (1.1)	Pag. 8
Borrar bases de datos, colecciones o todos los documentos de una colección	Pag. 9
Problemas Propuestos (1.2)	Pag. 11
Recuperar algunos documentos de una colección con el método find	Pag. 12
Operadores relacionales	Pag. 15
Problemas Propuestos (1.3)	Pag. 17
Borrar documentos de una colección con los métodos deleteOne y deleteMany	Pag. 19
Problemas Propuestos (1.4)	Pag. 22
Modificar un documento mediante el método updateOne	Pag. 23
Problemas Propuestos (1.5)	Pag. 28
Modificar múltiples documentos con el método updateMany	Pag. 29
Problemas Propuestos (1.6)	Pag. 32
Operaciones CRUD	Pag. 33
Operadores lógicos \$and, \$or y \$not	Pag. 35
Problemas Propuestos (1.7)	Pag. 38
Cursores y sus métodos en MongoDB (find-sort-pretty-limit-skip)	Pag. 40
Métodos find con query y projection	Pag. 46
Problemas Propuestos (1.8)	Pag. 48
Documentos embebidos: definición de campos de tipo documento	Pag. 50
Problemas Propuestos (1.9)	Pag. 53
Documentos embebidos: definición de campos de arreglo con elementos de tipo documento	Pag. 55
Problemas Propuestos (1.10)	Pag. 60
Campo _id generado por MongoDB	Pag. 62
Tipo de dato Date en MongoDB	Pag. 65
Problemas Propuestos (1.11)	Pag. 68
Tipo de dato Date en MongoDB	Pag. 69
Indices en MongoDB	Pag. 70
Indices en MongoDB – Simples y Compuestos	Pag. 73
Indices con campos de tipo documento y array	Pag. 82
Indices- eliminación	Pag. 86
MongoDB Shell y JavaScript	Pag. 88
MongoDB Shell cargar y ejecutar un archivo JavaScript *.js	Pag. 93
MongoDB Shell - conectarnos a un servidor remoto	Pag. 97

## 1. Elementos esenciales de MongoDB: Base De Datos - Colección - Documento

Crear primero nuestra primera base de datos:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> use base1
switched to db base1
>
```

Mediante el comando use activamos una base de datos existente o creamos una nueva (en nuestro caso la llamamos base1), queda luego activa la base de datos "base1".

Procedemos ahora a crear la colección libros e insertar el primer documento, la colección se crea en el momento que insertamos el primer documento:

```
db.libros.insertOne(
  {
    codigo: 1,
    nombre: 'El aleph',
    autor: 'Borges',
    editoriales: ['Planeta', 'Siglo XXI']
  }
)
```

En el shell de Mongo tenemos que ingresar:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.insertOne( {  codigo: 1,  nombre: 'El aleph',  autor: 'Borges',
editoriales: ['Planeta', 'Siglo XXI'] } )_
```

Luego de esto ya se ha creado la colección "libros" y se ha insertado el primer documento.

Procedamos a insertar el segundo documento:

```
db.libros.insertOne(  
  {  
    codigo: 2,  
    nombre: 'Martin Fierro',  
    autor: 'Jose Hernandez',  
    editoriales: ['Planeta']  
  }  
)
```

Tengamos en cuenta que mediante el objeto "db" accedemos a la base de datos activa (la misma la activamos con el comando use base1), seguidamente disponemos el nombre de la colección "libros" y finalmente el nombre del método "insertOne" al que le pasamos un objeto en formato JSON.

Ahora nuestra colección "libros" tiene dos documentos, si queremos mostrar los datos almacenados en la colección "libros" podemos llamar al método "find" sin pasar parámetros al mismo:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe  
> db.libros.find()  
{ "_id" : ObjectId("5c27f1edcd0f463fbb3699ce"), "codigo" : 1, "nombre" : "El ale  
ph", "autor" : "Borges", "editoriales" : [ "Planeta", "Siglo XXI" ] }  
{ "_id" : ObjectId("5c27f4ffcd0f463fbb3699cf"), "codigo" : 2, "nombre" : "Martin  
Fierro", "autor" : "Jose Hernandez", "editoriales" : [ "Planeta" ] }  
>
```

Los datos que vemos coinciden con los ingresados al llamar al método insertOne, con la salvedad que se ha agregado un campo llamado \_id en forma automática.

Todos los documentos requiere una clave principal almacenada en el campo \_id. Podemos indicar nosotros el valor a almacenar en el campo \_id, pero si no lo hacemos se crea en forma automática.

## Acotaciones

Cada vez que iniciamos MongoDB shell se activa por defecto la base de datos 'test' mediante el comando 'use' debemos activar la base de datos que necesitamos trabajar. Para saber en todo momento que base de datos se encuentra activa debemos escribir la variable 'db':



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe  
> db  
test  
> use base1  
switched to db base1  
> db  
base1  
>
```

## 2. Insertar documentos mediante los métodos insertOne e insertMany de una colección

Vimos en el concepto anterior como crear una base de datos documental en MongoDB, crear una colección e insertar un documento mediante el método insertOne.

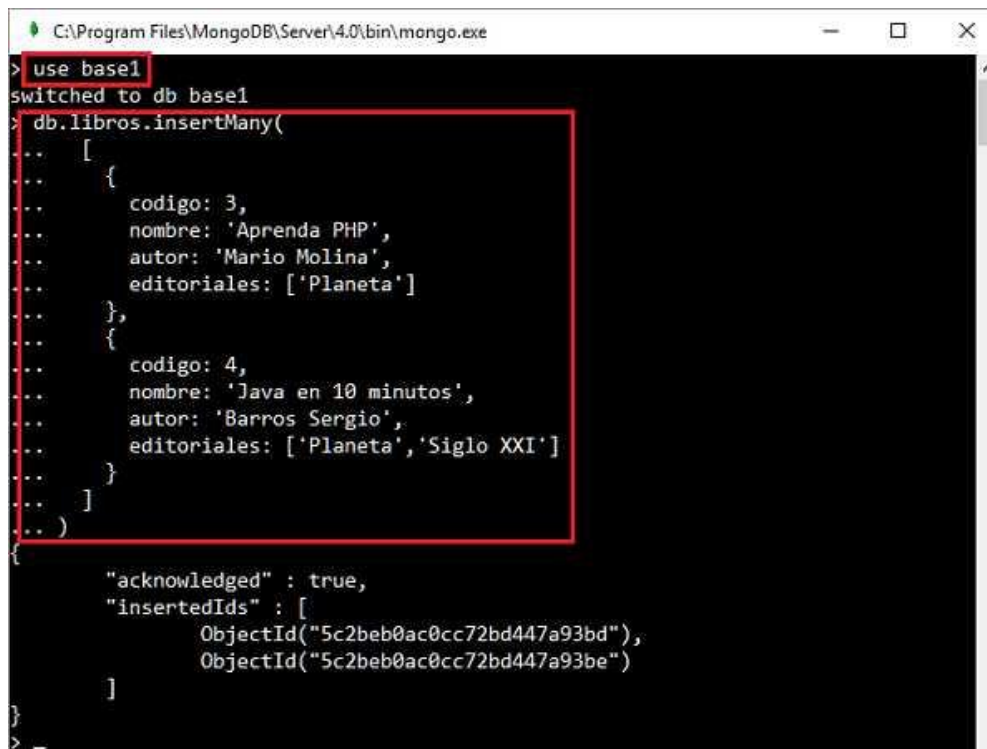
Para insertar un documento o un conjunto de documentos disponemos de los métodos:

- insertOne: Inserta un documento en una colección.
- insertMany: Inserta múltiples documentos en una colección.

Procedamos a insertar más de un documento en la colección "libros" mediante el método insertMany:

```
db.libros.insertMany(  
  [  
    {  
      codigo: 3,  
      nombre: 'Aprenda PHP',  
      autor: 'Mario Molina',  
      editoriales: ['Planeta']  
    },  
    {  
      codigo: 4,  
      nombre: 'Java en 10 minutos',  
      autor: 'Barros Sergio',  
      editoriales: ['Planeta', 'Siglo XXI']  
    }  
  ]  
)
```

En la consola de MongoDB tenemos como resultado:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe  
> use base1  
switched to db base1  
> db.libros.insertMany(  
..  [  
..    {  
..      codigo: 3,  
..      nombre: 'Aprenda PHP',  
..      autor: 'Mario Molina',  
..      editoriales: ['Planeta']  
..    },  
..    {  
..      codigo: 4,  
..      nombre: 'Java en 10 minutos',  
..      autor: 'Barros Sergio',  
..      editoriales: ['Planeta', 'Siglo XXI']  
..    }  
..  ]  
.. )  
{  
  "acknowledged" : true,  
  "insertedIds" : [  
    ObjectId("5c2beb0ac0cc72bd447a93bd"),  
    ObjectId("5c2beb0ac0cc72bd447a93be")  
  ]  
}
```

Tengamos en cuenta que si recién activamos la consola debemos activar la base de datos "base1" mediante el comando "use":

```
use base1
```

Luego llamamos al método "insertMany" y le pasamos un array con todos los documentos a almacenar en la colección "libros".

Tener en cuenta que utilizamos la consola de MongoDB (shell) con el objetivo a aprender los comandos esenciales, luego en la realidad estos datos serán enviados desde nuestras aplicaciones que podrán estar escritas en Python, Ruby, C#, Java etc.

Podemos borrar el contenido de la consola de MongoDB (shell) mediante el comando:

```
cls
```

El mismo resultado lo podemos obtener presionando las teclas: CTRL + L.

Mostremos los documentos almacenados en la colección "libros" mediante el método "find":



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find()
{ "_id" : ObjectId("5c27f1edcd0f463fbb3699ce"), "codigo" : 1, "nombre" : "El aleph", "autor" : "Borges", "editoriales" : [ "Planeta", "Siglo XXI" ] }
{ "_id" : ObjectId("5c27f4ffcd0f463fbb3699cf"), "codigo" : 2, "nombre" : "Martin Fierro", "autor" : "Jose Hernandez", "editoriales" : [ "Planeta" ] }
{ "_id" : ObjectId("5c2beb0ac0cc72bd447a93bd"), "codigo" : 3, "nombre" : "Aprende a PHP", "autor" : "Mario Molina", "editoriales" : [ "Planeta" ] }
{ "_id" : ObjectId("5c2beb0ac0cc72bd447a93be"), "codigo" : 4, "nombre" : "Java en 10 minutos", "autor" : "Barros Sergio", "editoriales" : [ "Planeta", "Siglo XXI" ] }
>
```

Como podemos observar al ejecutar el método "find" nuestra colección "libros" tiene almacenado 4 documentos.

### 3. Campo obligatorio \_id

En MongoDB todo documento requiere un campo clave que se debe llamar `_id`. Si nosotros como desarrolladores no definimos dicho campo el mismo se crea en forma automática y se carga un valor único.

Podemos definir y cargar un valor en el campo `_id` cuando inicializamos un documento:

```
db.clientes.insertOne(  
  {  
    _id: 1,  
    nombre: 'Lopez Marcos',  
    domicilio: 'Colon 111',  
    provincia: 'Cordoba'  
  }  
)
```

Cuando ejecutamos la inserción desde el shell de MongoDB tenemos como resultado:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe  
> use base1  
switched to db base1  
> db.clientes.insertOne(  
  {  
    _id: 1,  
    nombre: 'Lopez Marcos',  
    domicilio: 'Colon 111',  
    provincia: 'Cordoba'  
  }  
)  
{ "acknowledged" : true, "insertedId" : 1 }  
>
```

Cuando se ejecuta el método `insertOne` nos retorna un objeto JSON informando del resultado de la inserción mediante un objeto con dos campos, el primero `acknowledged` que indica si el documento fue admitido en la colección y el `_id` que en este caso lo define el usuario de la base de datos.

Si se produce un error nos genera un objeto JSON con otra estructura, probemos de intentar de ingresar un segundo documento con el mismo `_id`:

```
db.clientes.insertOne(  
  {  
    _id: 1,  
    nombre: 'Perez Ana',  
    domicilio: 'San Martin 222',  
    provincia: 'Santa Fe'  
  }  
)
```



```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.clientes.insertOne(
...   {
...     _id: 1,
...     nombre: 'Perez Ana',
...     domicilio: 'San Martin 222',
...     provincia: 'Santa Fe'
...   }
... )
2019-01-02T09:14:49.708-0300 E QUERY    [js] WriteError: E11000 duplicate key error collection: base1.clientes index: _id_ dup key: { : 1.0 } :
WriteError({
  "index" : 0,
  "code" : 11000,
  "errmsg" : "E11000 duplicate key error collection: base1.clientes index: _id_ dup key: { : 1.0 }",
  "op" : {
    "_id" : 1,
    "nombre" : "Perez Ana",
    "domicilio" : "San Martin 222",
    "provincia" : "Santa Fe"
  }
})
WriteError@src/mongo/shell/bulk_api.js:461:48
Bulk/mergeBatchResults@src/mongo/shell/bulk_api.js:841:49
Bulk/executeBatch@src/mongo/shell/bulk_api.js:906:13
Bulk/this.execute@src/mongo/shell/bulk_api.js:1150:21
DBCollection.prototype.insertOne@src/mongo/shell/crud_api.js:252:9
@(shell):1:1
>

```

Nos retorna un objeto JSON que entre otros campos define uno llamado `errmsg` con el mensaje de error.

Si nuestra aplicación administra el campo `'_id'` hay que tener en cuenta que nunca puede repetirse y en el caso que intentemos ingresar un documento con clave repetida luego dicho documento no se inserta en la colección.

Mostremos todos los documentos almacenados en la colección `libros`:

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.clientes.find()
{ "_id" : 1, "nombre" : "Lopez Marcos", "domicilio" : "Colon 111", "provincia" : "Cordoba" }
>

```

Hay uno solo ya que el segundo intento no se cargó.

## Problemas propuestos (1.1)

1. Insertar 2 documentos en la colección clientes con '\_id' no repetidos
2. Intentar insertar otro documento con clave repetida.
3. Mostrar todos los documentos de la colección libros.



#### 4. Borrar bases de datos, colecciones o todos los documentos de una colección

Hemos visto como se crea una base de datos, una colección y se insertan documentos en la misma.

Si queremos eliminar todos los documentos de una colección debemos utilizar el método "deleteMany" aplicado a una colección existente:

```
use base1
db.libros.deleteMany({})
show collections
```

Debemos pasar un objeto vacío que se indica con las llaves abiertas y cerradas {}. Luego veremos que podemos borrar solo los documentos que cumplen cierta condición.

Es importante notar que luego de llamar al método deleteMany la colección "libros" sigue existiendo:

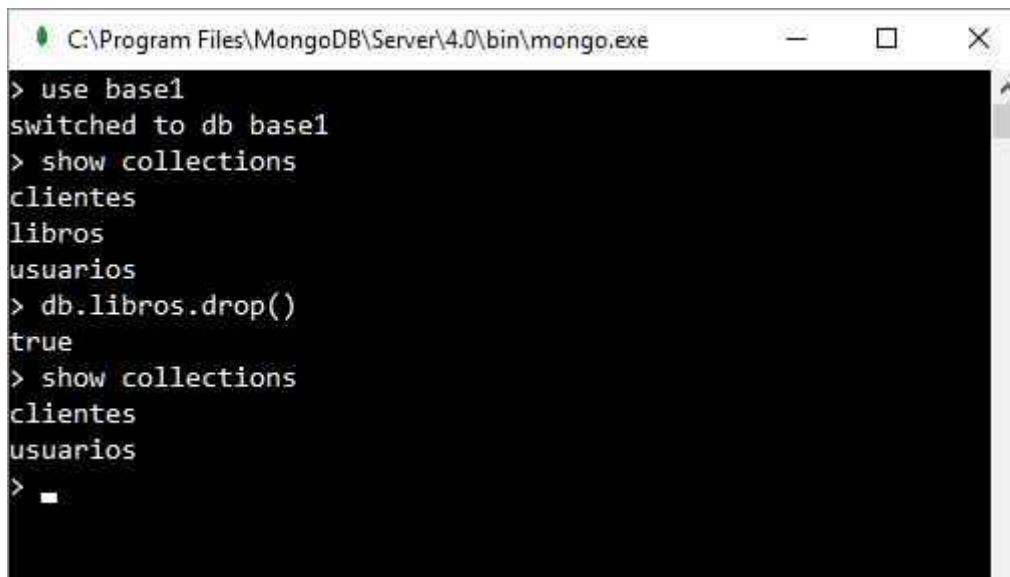


```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> use base1
switched to db base1
> db.libros.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
> show collections
clientes
libros
>
```

Para eliminar los documentos de una colección y la colección propiamente dicha debemos emplear el método "drop":

```
use base1
db.libros.drop()
show collections
```

Luego de llamar al método drop de la colección "libros" la misma deja de existir:

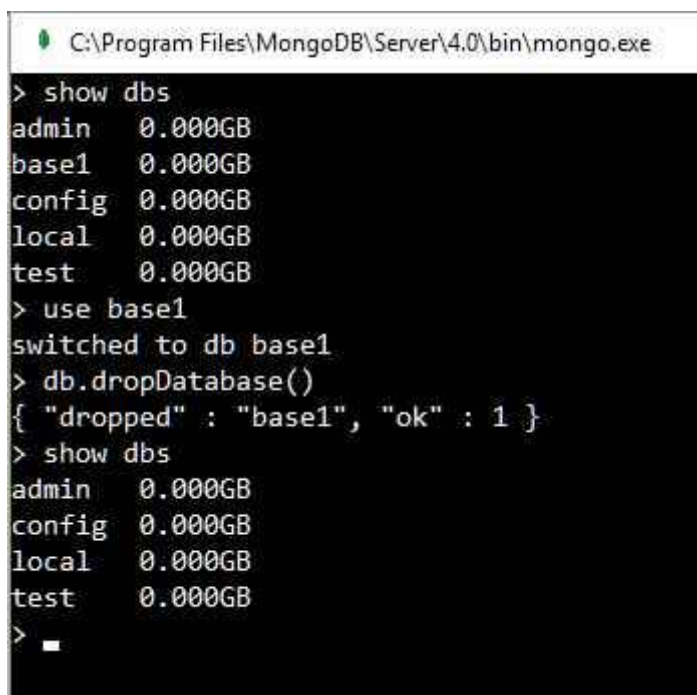


```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> use base1
switched to db base1
> show collections
clientes
libros
usuarios
> db.libros.drop()
true
> show collections
clientes
usuarios
> ■
```

Para eliminar una base de datos en forma completa, es decir todas sus colecciones y documentos debemos emplear el método dropDatabase del objeto "db":

```
show dbs
use base1
db.dropDatabase()
show dbs
```

El método dropDatabase elimina la base de datos activa:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> show dbs
admin    0.000GB
base1    0.000GB
config   0.000GB
local    0.000GB
test     0.000GB
> use base1
switched to db base1
> db.dropDatabase()
{ "dropped" : "base1", "ok" : 1 }
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
test     0.000GB
> ■
```

## Problemas propuestos (1.2)

1. Crear una base de datos llamada "blog".
2. Agregar una colección llamada "posts" e insertar 1 documento con una estructura a su elección.
3. Mostrar todas las bases de datos actuales.
4. Eliminar la colección "posts"
5. Eliminar la base de datos "blog" y mostrar las bases de datos existentes.

## 5. Recuperar algunos documentos de una colección con el método find

Suponiendo que tenemos la siguiente Base de Datos:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)
```

Podemos recuperar todos los documentos de una colección mediante el método find:

```
db.libros.find()
```

El resultado de ejecutar el comando es:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
...   _id: 4,
...   titulo: 'Java en 10 minutos',
...   editorial: ['Siglo XXI'],
...   precio: 45,
...   cantidad: 1
... }
... )
{ "acknowledged" : true, "insertedId" : 4 }
>
db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
```

El método find también nos permite seleccionar solo algunos documentos que cumplen una condición:

```
db.libros.find({_id : 1})
```

Rescatamos el documento que almacena en el campo '\_id' el valor 1:

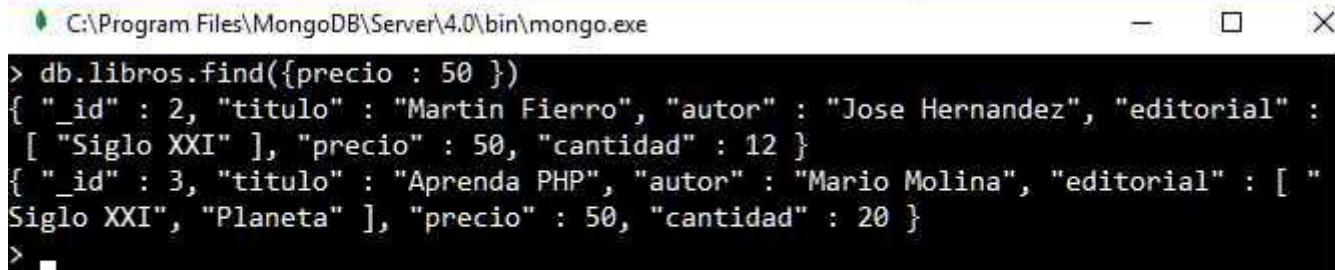
```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find({ _id: 1 })
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
>
```

Si pasamos un valor para el campo '\_id' que no existe luego el método find no regresa un documento.

Podemos rescatar todos los libros que tiene un precio igual a 50:

```
db.libros.find({precio : 50 })
```

Luego se recuperan dos documentos que cumplen la condición:

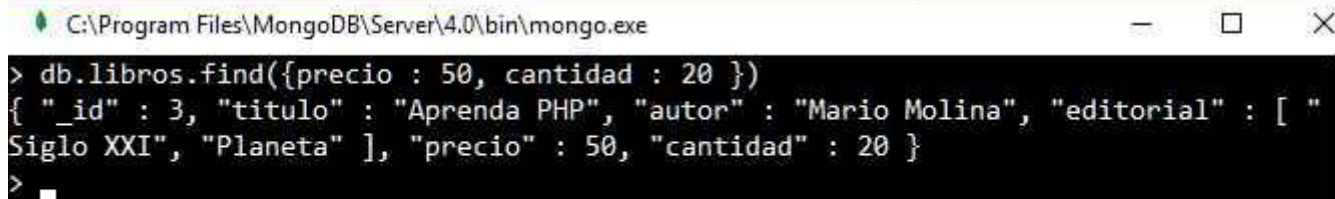


```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find({precio : 50 })
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
>
```

Podemos disponer más de un campo:

```
db.libros.find({precio : 50, cantidad : 20 })
```

Solo hay un documento que cumple estas dos condiciones:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find({precio : 50, cantidad : 20 })
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
>
```

## 6. Operadores relacionales

En el concepto anterior vimos cómo podemos seleccionar mediante el método find algunos documentos que cumplen una condición.

Es decir que cuando llamamos al método find pasamos un objeto literal pasando en el campo precio el valor 50, luego el método find filtra todos los libros cuyo precio sean exactamente igual a 50.

Otra forma de expresar la búsqueda de todos los libros con un precio igual a 50 es:

```
db.libros.find({ precio: { $eq : 50 } })
```

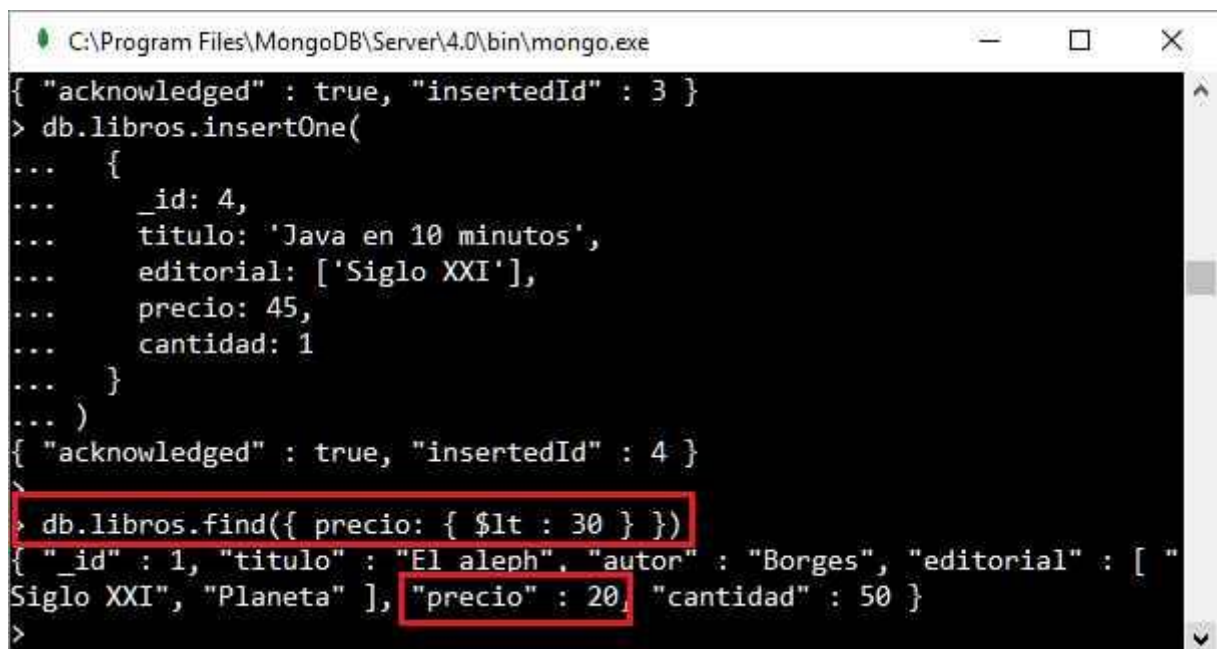
Es decir que luego del campo precio pasamos otro objeto literal iniciando el operador \$eq con el valor 50.

Mostramos esta segunda forma de consultar todos los libros con un precio igual a 50 debido a que cuando tenemos que consultar por ejemplo los libros con un precio inferior a 50, o superior a 50 etc. debemos indicar en forma obligatoria el operador a utilizar.

Para mostrar todos los libros con un precio inferior a 30 tenemos que utilizar el operador \$lt:

```
db.libros.find({ precio: { $lt : 30 } })
```

Tenemos luego como resultado con los datos ya cargados en la colección libros:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
{ "acknowledged" : true, "insertedId" : 3 }
> db.libros.insertOne(
...   {
...     _id: 4,
...     titulo: 'Java en 10 minutos',
...     editorial: ['Siglo XXI'],
...     precio: 45,
...     cantidad: 1
...   }
... )
{ "acknowledged" : true, "insertedId" : 4 }
> db.libros.find({ precio: { $lt : 30 } })
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
>
```

Hay un solo libro que tiene un precio menor a 30.



## Listado de operadores relacionales

- \$eq - equal - igual
- \$lt - low than - menor que
- \$lte - low than equal - menor o igual que
- \$gt - greater than - mayor que
- \$gte - greater than equal - mayor o igual que
- \$ne - not equal - distinto
- \$in - in - dentro de
- \$nin - not in - no dentro de

Veamos con algunos ejemplos como utilizar estos operadores para recuperar documentos que cumplen determinadas condiciones.

Recuperar todos los libros que tienen un precio mayor a 40:

```
db.libros.find({ precio: { $gt:40 } })
```

Recuperar todos los libros que en el campo cantidad tiene 50 o más:

```
db.libros.find( { cantidad: { $gte : 50 } })
```

Recuperar todos los libros que en el campo cantidad hay un valor distinto a 50:

```
db.libros.find( { cantidad: { $ne : 50 } })
```

Recuperar todos los libros cuyo precio estén comprendidos entre 20 y 45:

```
db.libros.find( { precio: { $gte : 20 , $lte : 45 } })
```

Recuperar todos los libros de la editorial 'Planeta':

```
db.libros.find( { editorial: { $in : ['Planeta'] } })
```

Recuperar todos los libros que no pertenezcan a la editorial 'Planeta':

```
db.libros.find( { editorial: { $nin : ['Planeta'] } })
```

Hay que acostumbrarse en un principio a utilizar estos operadores para filtrar documentos de una colección, luego veremos que estos operadores también se emplean cuando efectuemos borrados y modificaciones de documentos.

## Problemas propuestos (1.3)

1. Crear la colección 'articulos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 6 documentos:

```
use base1
db.articulos.drop()

db.articulos.insertOne(
  {
    _id: 1,
    nombre: 'MULTIFUNCION HP DESKJET 2675',
    rubro: 'impresora',
    precio: 3000,
    stock: 20
  }
)
db.articulos.insertOne(
  {
    _id: 2,
    nombre: 'MULTIFUNCION EPSON EXPRESSION XP241',
    rubro: 'impresora',
    precio: 3700,
    stock: 5
  }
)
db.articulos.insertOne(
  {
    _id: 3,
    nombre: 'LED 19 PHILIPS',
    rubro: 'monitor',
    precio: 4500,
    stock: 2
  }
)
db.articulos.insertOne(
  {
    _id: 4,
    nombre: 'LED 22 PHILIPS',
    rubro: 'monitor',
    precio: 5700,
    stock: 4
  }
)
db.articulos.insertOne(
  {
    _id: 5,
    nombre: 'LED 27 PHILIPS',
    rubro: 'monitor',
    precio: 12000,
    stock: 1
  }
)
```

```
db.articulos.insertOne(  
  {  
    _id: 6,  
    nombre: 'LOGITECH M90',  
    rubro: 'mouse',  
    precio: 300,  
    stock: 4  
  }  
)
```

2. Imprimir todos los documentos de la colección 'articulos'.
3. Imprimir todos los documentos de la colección 'articulos' que no son impresoras.
4. Imprimir todos los artículos que pertenecen al rubro de 'mouse'.
5. Imprimir todos los artículos con un precio mayor o igual a 5000.
6. Imprimir todas las impresoras que tienen un precio mayor o igual a 3500.
7. Imprimir todos los artículos cuyo stock se encuentra comprendido entre 0 y 4.

## 7. Borrar documentos de una colección con los métodos deleteOne y deleteMany

Podemos eliminar todos los documentos de una colección mediante el método deleteMany y pasando un objeto literal vacío:

```
db.libros.deleteMany({})
```

Aprendimos también a recuperar algunos documentos con el método find empleando una serie de operadores relacionales, dichos operadores se pueden emplear en forma idéntica con los métodos deleteMany y deleteOne.

Hay dos métodos para eliminar documentos:

- deleteMany: Borra todos los documentos que cumplen la condición que le enviamos.
- deleteOne: Borra el primer documento que cumple la condición que le pasamos.

Almacenaremos una serie de documentos en una colección llamada libros y luego borraremos algunos de sus documentos:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
```

```
        cantidad: 20
      }
    )
    db.libros.insertOne(
      {
        _id: 4,
        titulo: 'Java en 10 minutos',
        editorial: ['Siglo XXI'],
        precio: 45,
        cantidad: 1
      }
    )

    db.libros.find()

    db.libros.deleteOne({_id: 1})

    db.libros.find()

    db.libros.deleteMany({precio : {$gte : 50 }})

    db.libros.find()
```

Si queremos eliminar el documento que almacena en el campo el `_id` con valor 1 luego podemos utilizar la sintaxis:

```
db.libros.deleteOne({_id: 1})
```

Lo más conveniente es utilizar el método 'deleteOne' ya que solo uno puede cumplir esa condición al ser la clave primaria del documento.

Recordemos que la sintaxis alternativa para eliminar el documento con `_id` con valor 1 es:

```
db.libros.deleteOne({_id: { $eq : 1}})
```

La sintaxis anterior es buena recordar ya que los otros operadores relacionales hay que utilizarlos en forma obligatoria y no tienen una sintaxis alternativa como el `$eq`.

Para borrar todos los libros que tienen un precio mayor o igual a 50 tenemos:

```
db.libros.deleteMany({precio : {$gte : 50 }})
```

La ejecución del método `deleteOne` y `deleteMany` informa la cantidad de documentos eliminados:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe

> db.libros.deleteOne({_id: 1})
{ "acknowledged" : true, "deletedCount" : 1 }

> db.libros.find()
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }

> db.libros.deleteMany({precio : { $gte : 50 }})
{ "acknowledged" : true, "deletedCount" : 2 }

> db.libros.find()
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }

> ■
```

## Problemas propuestos (1.4)

1. Crear la colección 'articulos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 6 documentos: Utilizar la colección del problema propuesto anterior (1.3)
2. Imprimir todos los documentos de la colección 'articulos'.
3. Borrar los documentos de la colección 'articulos' cuyo rubro son impresoras, utilizar las dos sintaxis que permite MongoDB.
4. Borrar todos los artículos que tienen un \_id mayor o igual a 5.



## 7. Modificar un documento mediante el método updateOne

Vimos en conceptos anteriores como insertar un documento en una colección, recuperar un documento, borrar un documento y nos está faltando otra operación fundamental que podemos hacer con un documento que es su modificación.

Para modificar un documento en particular disponemos de un método llamado updateOne, veamos con un ejemplo algunas de sus posibilidades:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

db.libros.find()

db.libros.updateOne({_id : {$eq:1}} , {$set : {precio:15,cantidad:1}})

db.libros.find()
```

Hemos llamado al método `updateOne` con dos parámetros, el primero indica el documento a modificar y el segundo parámetro utilizamos el operador `$set` que es un operador de actualización seguidamente con los campos y valores a modificar.

Si existe el documento con `_id` igual a 1 luego se modifican los campos `precio` y `cantidad`:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.updateOne({_id : {$eq:1}} , {$set : {precio:15,cantidad:1}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 15, "cantidad" : 1 }
{ "_id" : 2, "titulo" : "Martín Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
>
```

Con las bases de datos documentales tengamos en cuenta que los documentos pueden tener distintas cantidades de campos. Por ejemplo si queremos agregar el campo descripción al libro con `'_id'` 4 debemos utilizar la sintaxis:

```
db.libros.updateOne({_id: {$eq:4}} ,{$set : {descripcion: 'Cada unidad trata un tema fundamental de Java desde 0.'}})
```

Luego de ejecutar tenemos el documento con `_id` 4 que contiene un nuevo campo llamado `'descripcion'` con el valor `'Cada unidad trata un tema fundamental de Java desde 0.'`:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.updateOne({_id: {$eq:4}} ,{$set : {descripcion: 'Cada unidad trata un tema fundamental de Java desde 0.'}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 15, "cantidad" : 1 }
{ "_id" : 2, "titulo" : "Martín Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1, "descripcion" : "Cada unidad trata un tema fundamental de Java desde 0." }
>
```

Si queremos eliminar un campo de un documento debemos emplear el operador de actualización \$unset. Probemos ahora de eliminar el campo que acabamos de crear para el documento con \_id 4:

```
db.libros.updateOne({_id : {$eq:4}} , {$unset : {descripcion:''} })
```

Luego de ejecutar el updateOne tenemos que para el documento que tiene el \_id 4 se ha eliminado el campo 'descripcion':



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.updateOne({_id : {$eq:4}} , {$unset : {descripcion:''} })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 15, "cantidad" : 1 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
```

Es importante entender que mediante el operador \$unset eliminamos el campo, en cambio si utilizamos el operador \$set modificamos el contenido del campo, luego si ejecutamos:

```
db.libros.updateOne({_id : {$eq:4}} , {$set : {descripcion:''} })
```

El campo 'descripcion' sigue existiendo y almacena una cadena de texto vacía, y si no existía se crea con una cadena vacía.

Disponemos también de operadores de modificación para arreglos, veamos como podemos agregar y eliminar elementos en el arreglo 'editorial':

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
```

```

db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

db.libros.find()

db.libros.updateOne({_id : {$eq:1}} , {$push : {editorial:'Atlántida'}})

db.libros.find()

```

Podemos ver que luego de ejecutarse el método updateOne el arreglo 'editorial' tiene una nueva componente para el documento con \_id 1:



```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.updateOne({_id : {$eq:1}} , {$push : {editorial:'Atlántida'}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta", "Atlántida" ], "precio" : 20, "cantidad" : 5 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
>

```

De forma similar para eliminar un elemento del arreglo debemos emplear el operador \$pull:

```
db.libros.updateOne({_id : {$eq:1}} , {$pull : {editorial:'Atlántida'}})
```

Podemos consultar todos los operadores de modificación en la página oficial de [MongoDB](#).

## Problemas propuestos (1.5)

1. Crear la colección 'articulos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 6 documentos: Utilizar la colección del problema propuesto anterior (1.3)
2. Imprimir todos los documentos de la colección 'articulos'.
3. Modificar el precio del mouse 'LOGITECH M90'.
4. Fijar el stock en 0 del artículo cuyo \_id es 6.
5. Agregar el campo proveedores con el array ['Martinez','Gutierrez'] para el artículo cuyo \_id es 6.
6. Eliminar el campo proveedores para el artículo cuyo \_id es 6.

## 8. Modificar múltiples documentos con el método updateMany

Vimos en el concepto anterior que MongoDB nos provee de un método que nos permite modificar un único documento llamado updateOne.

El segundo método que nos permite actualizar documentos pero en forma masiva es el método updateMany.

Veamos con un ejemplo algunas variantes del método updateMany:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

db.libros.find()

db.libros.updateMany({_id : {$gt:2}} , {$set : {cantidad:0} })

db.libros.find()
```



```
db.libros.updateMany({cantidad : {$eq:0}} , {$set : {faltantes:true}})

db.libros.find()

db.libros.updateMany({cantidad : {$eq:0}} , {$unset : {faltantes:true}, $set:
{cantidad: 100}} )

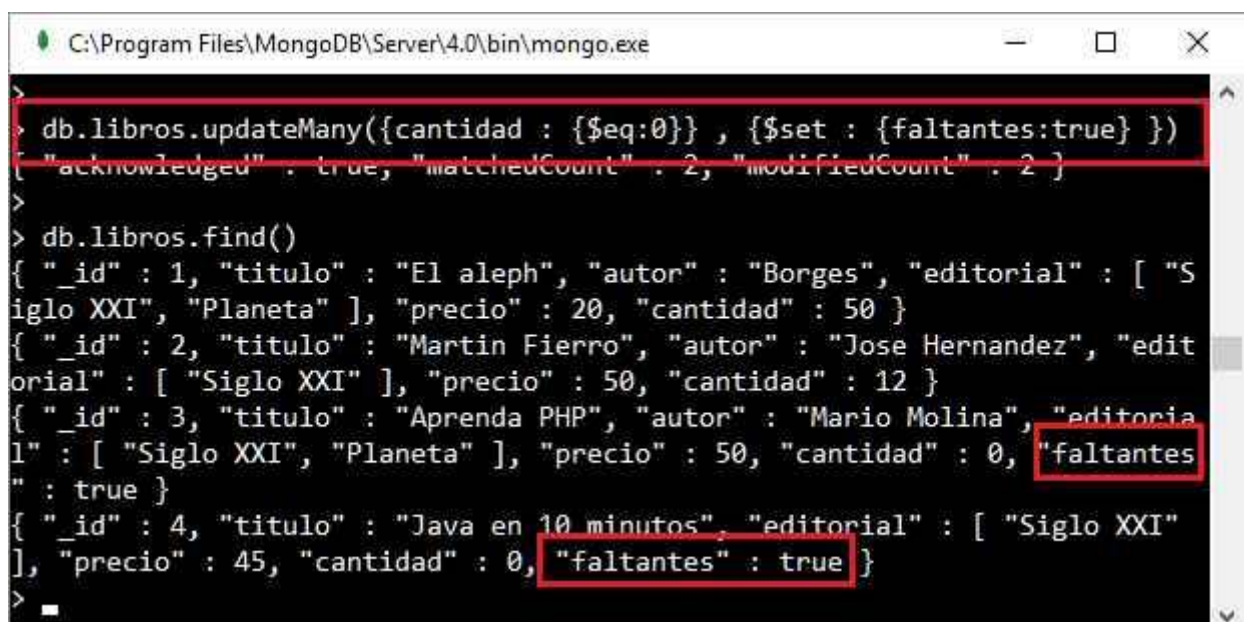
db.libros.find()
```

La primera modificación masiva la hacemos con todos los libros cuyo \_id sean mayores a 2, fijando el campo cantidad con 0:



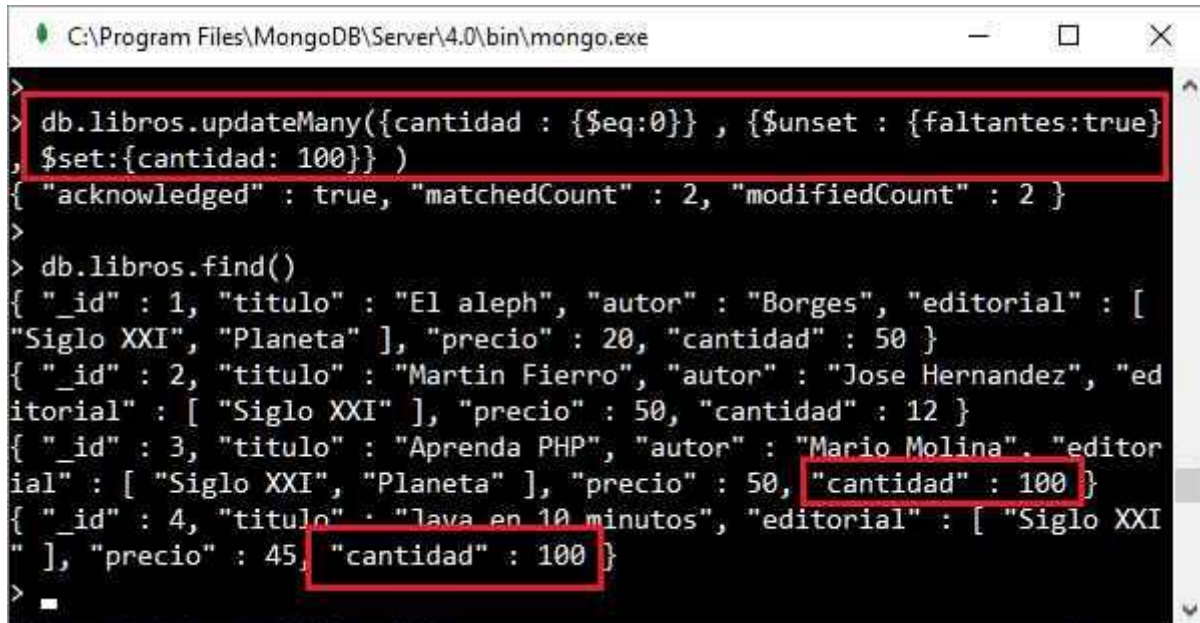
```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.updateMany({_id : {$gt:2}} , {$set : {cantidad:0}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 0 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 0 }
```

La segunda modificación masiva la hacemos con todos los libros que almacenan en el campo 'cantidad' el valor cero, agregando el campo 'faltantes' con el valor true:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.updateMany({cantidad : {$eq:0}} , {$set : {faltantes:true}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 0, "faltantes" : true }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 0, "faltantes" : true }
```

La tercer y última modificación masiva la hacemos con todos los libros que almacenan en el campo 'cantidad' el valor cero, eliminamos el campo faltante y fijamos el campo cantidad con el valor 100:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe

> db.libros.updateMany({cantidad : {$eq:0}} , {$unset : {faltantes:true}
, $set:{cantidad: 100}} )
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
>
> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [
"Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "ed
itorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editor
ial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 100 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI
" ], "precio" : 45, "cantidad" : 100 }
>
```

## Problemas propuestos (1.6)

1. Crear la colección 'articulos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 6 documentos: Utilizar la colección del problema propuesto anterior (1.3)
2. Imprimir todos los documentos de la colección 'articulos'.
3. Fijar el stock en cero para todos los artículos del rubro monitor.
4. Agregar un campo llamado 'pedir' con el valor true para todos los artículos que tienen el campo stock en 0.
5. Eliminar el campo 'pedir' de todos los documentos

## 9. Operaciones CRUD

Hacen referencia a cuatro operaciones básicas (vistas en capítulos anteriores) de toda aplicación: Create (Crear), Read (Leer), Update (Actualizar) y Delete (Borrar).

### Crear

Las operaciones de crear o insertar agregan nuevos documentos a una colección. Si la colección no existe, se intenta crear.

MongoDB provee los siguientes métodos para crear información.

- `db.<colección>.insertOne({})`  
Inserta un solo documento en una colección.
- `db.<colección>.insertMany([{}],{})`  
Inserta varios documentos en una colección.

Ej:

```
db.usuarios.insertOne(
{
  nombre: "Carlos",
  apellido: "Lopez"
})
```

### Leer

Las operaciones de lectura consisten en obtener documentos de una colección.

MongoDB provee los siguientes métodos:

- `db.<colección>.find(<filtro>)`  
Devuelve una vista filtrada de una colección.

Ej:

```
db.usuarios.find(
{
  rol: { $in: ['admin', 'gerente'] }
},
{ nombre: 1 }
)
```

## Actualizar

Las operaciones de actualización consisten en la modificación de documentos ya existentes en una colección.

MongoDB provee los siguientes métodos:

- `db.collection.updateOne()`  
Cambia algunas porciones de un solo documento.
- `db.collection.updateMany()`  
Cambia algunas porciones de muchos documentos.
- `db.collection.replaceOne()`  
Reemplaza totalmente un documento.

Ej:

```
db.usuarios.updateOne(  
  {  
    id: { $eq: 12 }  
  },  
  { $set: { active: false } }  
)
```

## Borrar

Las operaciones de borrado consisten en la eliminación de documentos ya existentes en una colección.

MongoDB provee los siguientes métodos:

- `db.collection.deleteOne()`  
Elimina un solo documento.
- `db.collection.deleteMany()`  
Elimina varios documentos.

Ej:

```
db.usuarios.deleteMany(  
  {  
    active: { $eq: false }  
  }  
)
```

## 10. Operadores lógicos \$and, \$or y \$not

Cuando necesitamos construir consultas que deban cumplir varias condiciones utilizaremos los operadores lógicos.

El operador \$and lo hemos utilizando en forma implícita, por ejemplo si tenemos:

```
db.libros.find({precio : 50, cantidad : 20 })
```

Con la condición anterior se recuperan todos los libros que tienen un precio de 50 y la cantidad es 20. Las dos condiciones deben ser verdaderas para que el documento se recupere.

La sintaxis alternativa para el find es:

```
db.libros.find({$and : [{precio:50}, {cantidad:20}] })
```

El valor para el operador \$and es un arreglo con cada una de las condiciones que debe cumplir.

Para los operadores \$or y \$not no hay una forma de disponer una sintaxis implícita.

Veamos con ejemplos el empleo de los operadores \$or y \$not:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
```

```
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

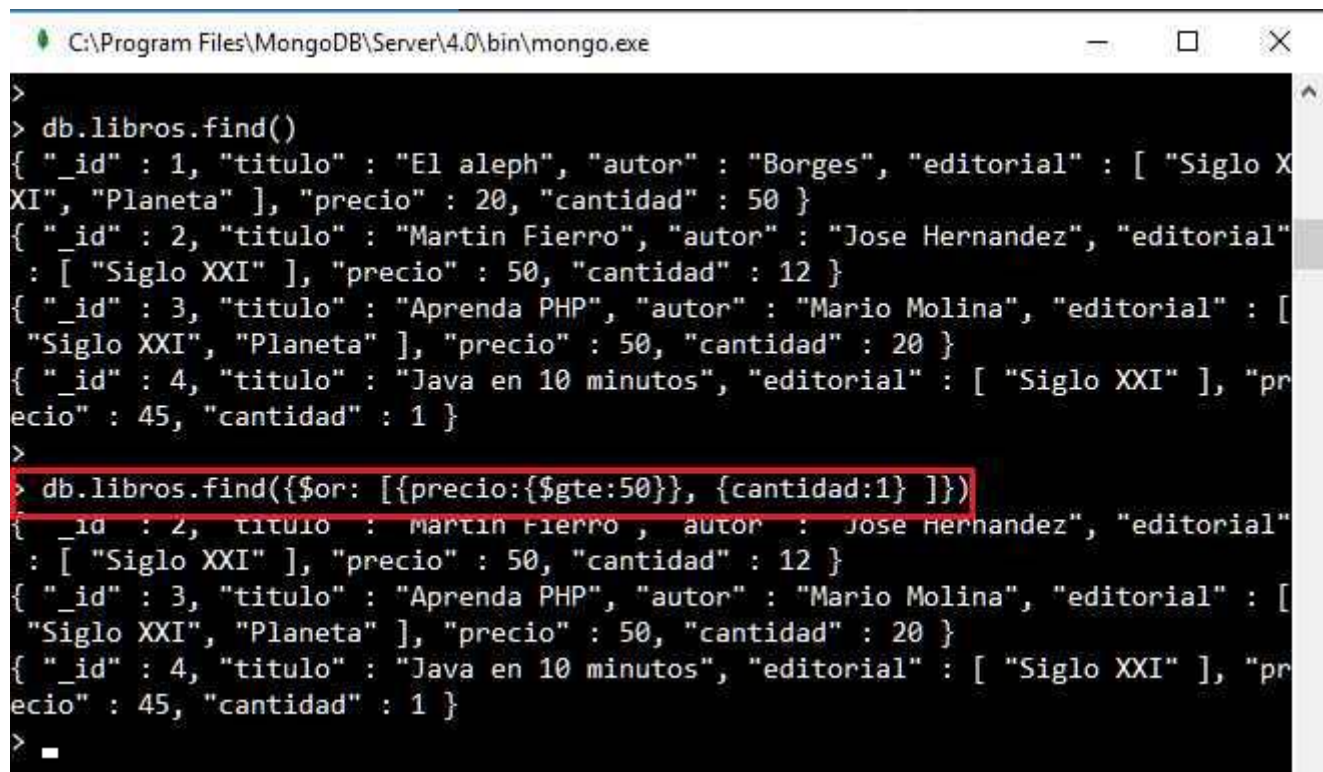
db.libros.find()

db.libros.find({$or: [{precio:{$gte:50}}, {cantidad:1} ]})
```

Para recuperar los libros que tienen un precio mayor o igual a 50 o la cantidad es 1 debemos implementar mediante un \$or la siguiente sintaxis:

```
db.libros.find({$or: [{precio:{$gte:50}}, {cantidad:1} ]})
```

Hay tres documentos que cumplen la condición:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe

> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
>
> db.libros.find({$or: [{precio:{$gte:50}}, {cantidad:1} ]})
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
>
```



Si queremos recuperar todos los documentos de la colección libros que no tienen un precio mayor o igual a 50 la sintaxis debe ser:

```
db.libros.find({precio: {$not:{$gte:50}} })
```

Los operadores lógicos podemos utilizarlos no solo para recuperar datos, sino también cuando borramos o actualizamos documentos.

Si queremos borrar todos los libros cuyo precio no sean iguales a 50 podemos codificar:

```
db.libros.deleteMany({precio: {$not:{$eq:50}} })
```

Se eliminan dos documentos de la colección libros.

## Problemas propuestos (1.7)

1. Crear la colección 'medicamentos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 6 documentos:

```
use base1
db.medicamentos.drop()

db.medicamentos.insertOne(
{
  _id: 1,
  nombre: 'Sertal',
  laboratorio: 'Roche',
  precio: 5.2,
  cantidad: 100
}
)
db.medicamentos.insertOne(
{
  _id: 2,
  nombre: 'Buscapina',
  laboratorio: 'Roche',
  precio: 4.10,
  cantidad: 200
}
)
db.medicamentos.insertOne(
{
  _id: 3,
  nombre: 'Amoxidal 500',
  laboratorio: 'Bayer',
  precio: 15.60,
  cantidad: 100
}
)
db.medicamentos.insertOne(
{
  _id: 4,
  nombre: 'Paracetamol 500',
  laboratorio: 'Bago',
  precio: 1.90,
  cantidad: 200
}
)
db.medicamentos.insertOne(
{
  _id: 5,
  nombre: 'Bayaspirina',
  laboratorio: 'Bayer',
  precio: 2.10,
  cantidad: 150
}
)
db.medicamentos.insertOne(
{
  _id: 6,
  nombre: 'Amoxidal jarabe',
  laboratorio: 'Bayer',
  precio: 5.10,
  cantidad: 50
}
)
```

2. Imprimir todos los documentos de la colección 'medicamentos'.
3. Recupere los medicamentos cuyo laboratorio sea 'Roche' y cuyo precio sea menor a 5.

4. Recupere los medicamentos cuyo laboratorio sea 'Roche' o cuyo precio sea menor a 5.
5. Muestre todos los medicamentos cuyo laboratorio NO sea "Bayer"
6. Muestre todos los medicamentos cuyo laboratorio sea "Bayer" y cuya cantidad NO sea=100
7. Elimine todos los documentos de la colección medicamentos cuyo laboratorio sea igual a "Bayer" y su precio sea mayor a 10
8. Cambie la cantidad por 200, a todos los medicamentos de "Roche" cuyo precio sea mayor a 5
9. Borre los medicamentos cuyo laboratorio sea "Bayer" o cuyo precio sea menor a 3

## 14. Cursores y sus métodos en MongoDB (find-sort-pretty-limit-skip)

Cada vez que llamamos al método find de una colección el mismo no retorna un objeto de la clase Cursor.

Si no asignamos el valor a una variable en el shell de MongoDB luego se muestran los documentos recuperados y se nos pide que confirmemos cada vez que se muestran 20.

Podemos encadenar la llamada al método find con los métodos de la clase Cursor, por ejemplo si queremos que se muestren todos los libros ordenados por el nombre tenemos que implementar la siguiente lógica:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

db.libros.find().sort({titulo:1})
```

A partir del cursor que retorna el método 'find' llamamos al método 'sort' de la clase Cursor y como condición indicamos por el campo que queremos ordenar (si pasamos un 1 se ordena en forma ascendente y si pasamos un -1 se ordena en forma descendente):



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
...   precio: 45,
...   cantidad: 1
... }
... )
{ "acknowledged" : true, "insertedId" : 4 }
>
db.libros.find().sort({titulo:1})
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
{ "_id" : 2, "titulo" : "Martín Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
>
```

Otro método de la clase Cursor que nos puede ayudar cuando ejecutamos comandos desde el shell de MongoDB es 'pretty', el mismo tiene por objetivo mostrarnos los datos del cursor en forma más legible:

```
db.libros.find().pretty()
```

La salida de los datos del Cursor empleando el método pretty es:

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find().pretty()
{
  "_id" : 1,
  "titulo" : "El aleph",
  "autor" : "Borges",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 20,
  "cantidad" : 50
}
{
  "_id" : 2,
  "titulo" : "Martin Fierro",
  "autor" : "Jose Hernandez",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 50,
  "cantidad" : 12
}
{
  "_id" : 3,
  "titulo" : "Aprenda PHP",
  "autor" : "Mario Molina",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 50,
  "cantidad" : 20
}
{
  "_id" : 4,
  "titulo" : "Java en 10 minutos",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 45,
  "cantidad" : 1
}

```

El método sort también retorna un Cursor con los datos ordenados, luego podemos llamar al método pretty a partir del Cursor devuelto por sort:

```
db.libros.find().sort({titulo:1}).pretty()
```

Otro método útil es el 'limit' que tiene por objetivo limitar a un determinado número de documentos a recuperar del Cursor. Por ejemplo tenemos que la llamada al método find retorna 4 documentos, podemos aplicar el método limit a dicho Cursor para que se limite a recuperar los dos primeros:

```
db.libros.find().limit(2)
```



```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find().limit(2)
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges",
  "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
>
  
```


Nuevamente tener en cuenta que primero podemos ordenar los datos del Cursor y llamar a partir de éste al método limit:

```
db.libros.find().sort({titulo:1}).limit(2)
```

Otro método llamado 'skip' nos permite saltar una determinada cantidad de documentos desde el principio del cursor, por ejemplo:

```
db.libros.find().skip(1)
```

El método find retorna un Cursor con 4 documentos, pero mediante la llamada al método skip indicamos que comience a partir del segundo documento hasta el final:



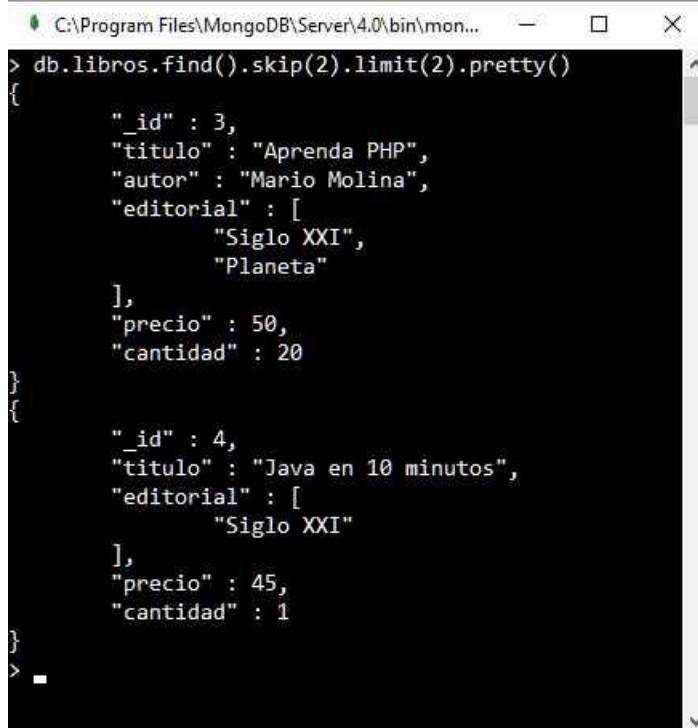
```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find().skip(1)
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
>
  
```

Podemos combinar las llamadas al método skip y limit:

```
db.libros.find().skip(2).limit(2).pretty()
```

Recuperamos a partir del tercer documento del Cursor la cantidad de 2 documentos y se muestran en una forma legible:



```
C:\Program Files\MongoDB\Server\4.0\bin\mon...
> db.libros.find().skip(2).limit(2).pretty()
{
  "_id" : 3,
  "titulo" : "Aprenda PHP",
  "autor" : "Mario Molina",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 50,
  "cantidad" : 20
}
{
  "_id" : 4,
  "titulo" : "Java en 10 minutos",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 45,
  "cantidad" : 1
}
>
```

Existen muchos métodos en la clase `Cursos` que iremos viendo a medida que los necesitemos, podemos consultar todos los métodos en la página oficial de MongoDB.



## 15. Métodos find con query y projection

Hemos visto que el método 'find':

- Si no le pasamos parámetros nos retorna todos los documentos de la colección que hace referencia:

```
db.libros.find()
```

- El primer parámetro en el caso que lo indiquemos filtra la colección y recupera los documentos que cumplen la condición:

```
db.libros.find({precio : 50 })
```

Recuperamos todos los libros que tienen un precio igual a 50. (**query**)

- Hay un segundo parámetro opcional en el cual debemos indicar que campos del documento queremos recuperar:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
```

```
    precio: 45,  
    cantidad: 1  
  }  
)  
  
db.libros.find({precio : 50 },{titulo:1,cantidad:1})
```

En el segundo parámetro del método 'find' debemos especificar cada campo y un valor 1 indicando que se lo quiere recuperar (**projection**). El campo `_id` se recupera por defecto, salvo que indiquemos con un valor 0 para que no se recupere:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe  
>  
> db.libros.find({precio : 50 },{titulo:1,cantidad:1})  
{ "_id" : 2, "titulo" : "Martin Fierro", "cantidad" : 12 }  
{ "_id" : 3, "titulo" : "Aprenda PHP", "cantidad" : 20 }  
>
```

Solo se recuperan los campos titulo,cantidad y `_id`.

## Problemas propuestos (1.8)

1. Crear la colección 'articulos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 6 documentos:

```
use base1
db.articulos.drop()

db.articulos.insertOne(
  {
    _id: 1,
    nombre: 'MULTIFUNCION HP DESKJET 2675',
    rubro: 'impresora',
    precio: 3000,
    stock: 20
  }
)
db.articulos.insertOne(
  {
    _id: 2,
    nombre: 'MULTIFUNCION EPSON EXPRESSION XP241',
    rubro: 'impresora',
    precio: 3700,
    stock: 5
  }
)
db.articulos.insertOne(
  {
    _id: 3,
    nombre: 'LED 19 PHILIPS',
    rubro: 'monitor',
    precio: 4500,
    stock: 2
  }
)
db.articulos.insertOne(
  {
    _id: 4,
    nombre: 'LED 22 PHILIPS',
    rubro: 'monitor',
    precio: 5700,
    stock: 4
  }
)
db.articulos.insertOne(
  {
    _id: 5,
    nombre: 'LED 27 PHILIPS',
    rubro: 'monitor',
    precio: 12000,
    stock: 1
  }
)

db.articulos.insertOne(
```

```
{
+   _id: 6,
  nombre: 'LOGITECH M90',
  rubro: 'mouse',
  precio: 300,
  stock: 4
}
```

2. Imprimir todos los documentos de la colección 'articulos', mostrar solo los campos `_id` y nombre.
3. Imprimir todos los documentos de la colección 'articulos' que son impresoras, mostrar solo los campos nombre y precio.
4. Imprimir todas las impresoras que tienen un precio mayor o igual a 3500. Solo mostrar los campos `_id`, nombre, precio y stock
5. Imprimir todos los documentos de la colección 'articulos' que son monitor, mostrar solo los campos nombre y precio ordenados de menor a mayor.

## 16. Documentos embebidos: definición de campos de tipo documento

Hasta ahora hemos trabajado definiendo documentos con una serie de campos que almacenan tipo de datos simples como enteros, reales, cadenas de caracteres y datos compuestos de tipo arreglo.

Según el tipo de consultas que haremos a nuestros documentos debemos definir el esquema de campos de los mismos. Supongamos que tenemos que almacenar el dato de clientes y luego hacer consultas de donde viven y discriminar por calle y número, luego puede tener sentido tener un campo dirección que sea un documento donde se almacenen por separado la calle, el número y el código postal.

Veamos con un ejemplo como creamos una colección de documentos utilizando documentos embebidos.

```
use base1
db.clientes.drop()

db.clientes.insertOne(
  {
    _id: 1,
    nombre: 'Martinez Victor',
    mail: 'mvictor@gmail.com',
    direccion: {
      calle: 'Colon',
      numero: 620,
      codigopostal: 5000
    }
  }
)
db.clientes.insertOne(
  {
    _id: 2,
    nombre: 'Alonso Carlos',
    mail: 'acarlos@gmail.com',
    direccion: {
      calle: 'Colon',
      numero: 150,
      codigopostal: 5000
    }
  }
)
db.clientes.insertOne(
  {
    _id: 3,
    nombre: 'Gonzalez Marta',
    mail: 'gmarta@outlook.com',
    direccion: {
      calle: 'Colon',
      numero: 1200,
      codigopostal: 5000
    }
  }
)
)
```

```
db.clientes.insertOne(
  {
    _id: 4,
    nombre: 'Ferrero Ariel',
    mail: 'fariel@yahoo.com',
    direccion: {
      calle: 'Dean Funes',
      numero: 23,
      codigopostal: 5002
    }
  }
)
db.clientes.insertOne(
  {
    _id: 5,
    nombre: 'Fernandez Diego',
    mail: 'fdiego@gmail.com',
    direccion: {
      calle: 'Dean Funes',
      numero: 561,
      codigopostal: 5002
    }
  }
)

db.clientes.find()
```

Para recuperar todos los clientes que viven en la calle 'Colon' tenemos que plantear la siguiente consulta:

```
db.clientes.find({'direccion.calle': 'Colon'}).pretty()
```

Es obligatorio disponer las comillas cuando hacemos referencia a un subcampo de un documento: 'direccion.calle'.

El resultado de ejecutar la consulta es:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.clientes.find({'direccion.calle':'Colon'}).pretty()
{
  "_id" : 1,
  "nombre" : "Martinez Victor",
  "mail" : "mvictor@gmail.com",
  "direccion" : {
    "calle" : "Colon",
    "numero" : 620,
    "codigopostal" : 5000
  }
}
{
  "_id" : 2,
  "nombre" : "Alonso Carlos",
  "mail" : "acarlos@gmail.com",
  "direccion" : {
    "calle" : "Colon",
    "numero" : 150,
    "codigopostal" : 5000
  }
}
{
  "_id" : 3,
  "nombre" : "Gonzalez Marta",
  "mail" : "gmarta@outlook.com",
  "direccion" : {
    "calle" : "Colon",
    "numero" : 1200,
    "codigopostal" : 5000
  }
}
>
```

Para recuperar todos los clientes que viven en la calle 'Colon' y su número está comprendido entre 1 y 1000 tenemos que plantear la siguiente consulta:

```
db.clientes.find({ 'direccion.calle':'Colon', 'direccion.numero':{'$gte:0'}, 'direccion.numero':{'$lte:1000'} }).pretty()
```

Si ejecutamos la consulta podemos comprobar que hay dos documentos que cumplen la condición impuesta al método 'find':

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.clientes.find({'direccion.calle':'Colon', 'direccion.numero':{'$gte:0'}, 'direccion.numero':{'$lte:1000'}}).pretty()
{
  "_id" : 1,
  "nombre" : "Martinez Victor",
  "mail" : "mvictor@gmail.com",
  "direccion" : {
    "calle" : "Colon",
    "numero" : 620,
    "codigopostal" : 5000
  }
}
{
  "_id" : 2,
  "nombre" : "Alonso Carlos",
  "mail" : "acarlos@gmail.com",
  "direccion" : {
    "calle" : "Colon",
    "numero" : 150,
    "codigopostal" : 5000
  }
}
>
```

## Problemas propuestos (1.9)

1. Crear la colección 'libros' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 4 documentos:

```
use base1
db.libros.drop()
db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: {
      nombre: 'Borges',
      nacionalidad: 'Argentina'
    },
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: {
      nombre: 'Jose Hernandez',
      nacionalidad: 'Argentina'
    },
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: {
      nombre: 'Mario Molina',
      nacionalidad: 'Española'
    },
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    autor: {
      nombre: 'Java en 10 minutos',
      nacionalidad: 'Española'
    },
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
})
```



2. Imprimir todos los documentos de la colección 'libros'.
3. Imprimir todos los libros de autores de nacionalidad 'Argentina'.
4. Imprimir los libros de 'Borges'.
5. Imprimir todos los libros de nacionalidad 'Española' que cuestan 50 o más.

## 17. Documentos embebidos: definición de campos de arreglo con elementos de tipo documento

Una de las ventajas fundamentales del gestor de base de datos MongoDB es la posibilidad de crear esquemas de documentos muy flexibles y no tener que guardar datos relacionados en distintos documentos.

Desde los primeros ejemplos hemos visto que podemos definir campos en un documento de tipo arreglo:

```
db.libros.insertOne(
  {
    codigo: 1,
    nombre: 'El aleph',
    autor: 'Borges',
    editoriales: ['Planeta', 'Siglo XXI']
  }
)
```

El campo 'editoriales' es de tipo arreglo y almacena dos cadenas de caracteres.

Podemos definir campos de tipo arreglo y almacenar en los mismos documentos embebidos, por ejemplo:

```
use base1
db.posts.drop()

db.posts.insertOne(
  {
    _id: 1,
    titulo: 'Lenguaje Java',
    contenido: 'Uno de los lenguajes más utilizados es ...',
    comentarios: [{
      autor: 'Marcos Paz',
      mail: 'pazm@gmail.com',
      contenido: 'Me parece un buen...'
    },
    {
      autor: 'Ana Martinez',
      mail: 'martineza@gmail.com',
      contenido: 'Todo ha cambiado en...'
    },
    {
      autor: 'Luiz Blanco',
      mail: 'blanco1@outlook.com',
      contenido: 'Afirmo que es...'
    }
  ]
})

db.posts.find().pretty()
```

El campo comentarios almacena un arreglo de tres elementos, cada elemento es un documento.

La ejecución de este bloque:

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> use base1
switched to db base1
> db.posts.drop()
true
>
> db.posts.insertOne(
...   {
...     _id: 1,
...     titulo: 'Lenguaje Java',
...     contenido: 'Uno de los lenguajes más utilizados es ...',
...     comentarios: [{
...       autor: 'Marcos Paz',
...       mail: 'pazm@gmail.com',
...       contenido: 'Me parece un buen...'
...     }],
...     {
...       autor: 'Ana Martinez',
...       mail: 'martineza@gmail.com',
...       contenido: 'Todo ha cambiado en...'
...     },
...     {
...       autor: 'Luiz Blanco',
...       mail: 'blanco1@outlook.com',
...       contenido: 'Afirmo que es...'
...     }
...   }
... )
{ "acknowledged" : true, "insertedId" : 1 }
>
> db.posts.find().pretty()
{
  "_id" : 1,
  "titulo" : "Lenguaje Java",
  "contenido" : "Uno de los lenguajes más utilizados es ...",
  "comentarios" : [
    {
      "autor" : "Marcos Paz",
      "mail" : "pazm@gmail.com",
      "contenido" : "Me parece un buen..."
    },
    {
      "autor" : "Ana Martinez",
      "mail" : "martineza@gmail.com",
      "contenido" : "Todo ha cambiado en..."
    },
    {
      "autor" : "Luiz Blanco",
      "mail" : "blanco1@outlook.com",
      "contenido" : "Afirmo que es..."
    }
  ]
}

```

La idea fundamental en MongoDB es disponer de todos los datos agrupados dependiendo de los accesos futuros a la información, es

evidente que cuando queremos acceder a una entrada de un bloq también necesitemos acceder a los comentarios de dicha entrada.

Esta forma de organizar los datos nos permite implementar aplicaciones muy eficientes en comparación al modelo de base de datos relacional (MySQL, Oracle, SQL Server etc.)

Podemos generar documentos embebidos con más niveles, por ejemplo definir un campo en el arreglo que sea de tipo documento.

## Consultas de los documentos embebidos

Ejecutemos el siguiente bloque de comandos en MongoDB para analizar las consultas en documentos embebidos dentro de arreglos:

```
use base1
db.posts.drop()

db.posts.insertOne(
  {
    _id: 1,
    titulo: 'Lenguaje Java',
    contenido: 'Uno de los lenguajes más utilizados es ...',
    comentarios: [{
      autor: 'Marcos Paz',
      mail: 'pazm@gmail.com',
      contenido: 'Me parece un buen...'
    },
    {
      autor: 'Ana Martinez',
      mail: 'martineza@gmail.com',
      contenido: 'Todo ha cambiado en...'
    },
    {
      autor: 'Luiz Blanco',
      mail: 'blanco1@outlook.com',
      contenido: 'Afirmo que es...'
    }
  ]
})

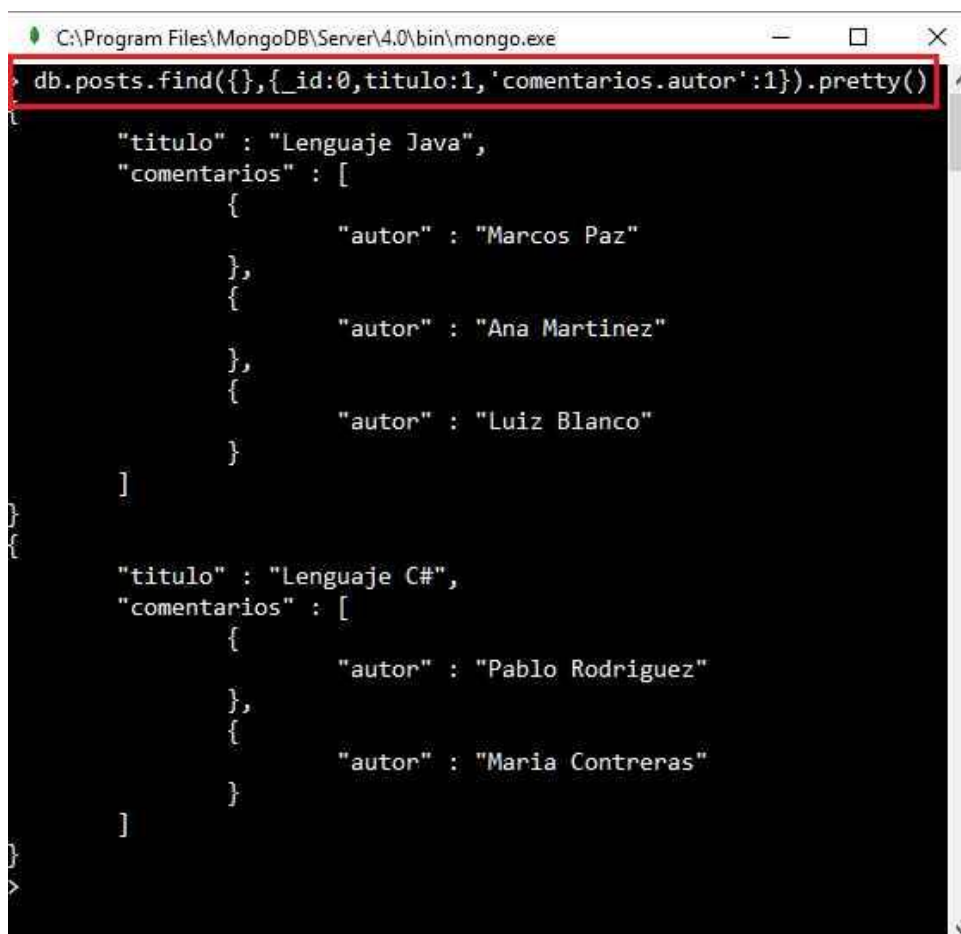
db.posts.insertOne(
  {
    _id: 2,
    titulo: 'Lenguaje C#',
    contenido: 'Microsoft desarrolla el lenguaje C# con el objetivo ...',
    comentarios: [{
      autor: 'Pablo Rodriguez',
      mail: 'rodriguezp@gmail.com',
      contenido: 'Correcta idea.'
    },
    {
      autor: 'Maria Contreras',
      mail: 'contrerasm@gmail.com',
      contenido: 'Buen punto de vista...'
    }
  ]
})
```

```
)
db.posts.find({}, {_id:0,titulo:1,'comentarios.autor':1}).pretty()
db.posts.find({'comentarios.autor':'Pablo Rodriguez'}).pretty()
db.posts.find({'comentarios.0.autor':'Pablo Rodriguez'}).pretty()
```

Si queremos recuperar los títulos de todos los posts y los nombres de los usuarios que hicieron comentarios en cada post, debemos implementar la siguiente consulta:

```
db.posts.find({}, {_id:0,titulo:1,'comentarios.autor':1}).pretty()
```

Y como resultado tenemos:



The screenshot shows a MongoDB console window with the command `db.posts.find({}, {_id:0,titulo:1,'comentarios.autor':1}).pretty()` entered. The output displays two documents. The first document is for a post titled "Lenguaje Java" with three comments by "Marcos Paz", "Ana Martinez", and "Luiz Blanco". The second document is for a post titled "Lenguaje C#" with two comments by "Pablo Rodriguez" and "Maria Contreras".

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.posts.find({}, {_id:0,titulo:1,'comentarios.autor':1}).pretty()
{
  "titulo" : "Lenguaje Java",
  "comentarios" : [
    {
      "autor" : "Marcos Paz"
    },
    {
      "autor" : "Ana Martinez"
    },
    {
      "autor" : "Luiz Blanco"
    }
  ]
}
{
  "titulo" : "Lenguaje C#",
  "comentarios" : [
    {
      "autor" : "Pablo Rodriguez"
    },
    {
      "autor" : "Maria Contreras"
    }
  ]
}

```

Para recuperar todos los posts donde ha comentado el usuario 'Pablo Rodriguez':

```
db.posts.find({'comentarios.autor':'Pablo Rodriguez'}).pretty()
```

Y como resultado tenemos:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.posts.find({'comentarios.autor':'Pablo Rodriguez'}).pretty()
{
  "_id" : 2,
  "titulo" : "Lenguaje C#",
  "contenido" : "Microsoft desarrolla el lenguaje C# con el ob
jetivo ...",
  "comentarios" : [
    {
      "autor" : "Pablo Rodriguez",
      "mail" : "rodriguezp@gmail.com",
      "contenido" : "Correcta idea."
    },
    {
      "autor" : "Maria Contreras",
      "mail" : "contrerasm@gmail.com",
      "contenido" : "Buen punto de vista..."
    }
  ]
}
```

Podemos utilizar el subíndice del arreglo para analizar un documento en particular, por ejemplo si queremos recuperar todos los posts que comentó primero 'Rodriguez Pablo':

```
db.posts.find({'comentarios.0.autor':'Pablo Rodriguez'}).pretty()
```

## Acotaciones

Cuando definimos documentos embebidos debemos tener en cuenta que un documento en MongoDB no puede tener un tamaño mayor a 16MB.

No podemos crear una colección 'ciudades' y almacenar documentos con el nombre de la ciudad y embeber documentos con los nombres de todos sus ciudadanos (podría haber millones por cada ciudad)

En el ejemplo de la colección 'posts' los comentarios por cada post serán una cantidad limitada y no millones.

Cuando uno define el esquema de los documentos a almacenar en MongoDB debe tener en cuenta cuanto pueden crecer los documentos embebidos para tomar la decisión de separar los mismos en documentos independientes.

## Problemas propuestos (1.10)

1. Crear la colección 'libros' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 4 documentos:

```
db.series.insertOne(
{
  _id: 1,
  titulo: 'The big bang theory',
  productor: 'Chuck Lorre',
  actores: ['Johnny Galecki','Jim Parsons','Kaley Cuoco','Kunal Nayyar','Simon Helberg','Mayim Bialik','Melissa Rauch'],
  temporada1: [
    {
      capitulo1:{
        titulo:'Piloto',
        audiencia:8300000
      }
    },
    {
      capitulo2:{
        titulo:'La hipótesis del Gran Cerebro',
        audiencia:8700000
      },
    },
    {
      capitulo3:{
        titulo:'El Corolario de el Gato con Botas',
        audiencia:9200000
      }
    }
  ],
  temporada2: [
    {
      capitulo1:{
        titulo:'El paradigma del pescado malo',
        audiencia:10000000
      }
    },
    {
      capitulo2:{
        titulo:'La topología de la bragueta',
        audiencia:11000000
      }
    }
  ]
}
)

db.series.insertOne(
{
  _id: 2,
  titulo: 'The Walking Dead',
  productor: 'Robert Kirkman',
  actores: ['Andrew Lincoln','Jon Bernthal','Sarah Wayne Callies','Laurie Holden','Jeffrey DeMunn','Steven Yeun'],
```

```

temporada1: [
  {
    capitulo1:{
      titulo:'TS 19',
      audiencia:7000000
    }
  },
  {
    capitulo2:{
      titulo:'Wildfire',
      audiencia:8200000
    }
  },
  {
    capitulo3:{
      titulo:'Díselo a las ranas',
      audiencia:9100000
    }
  }
],
temporada2: [
  {
    capitulo1:{
      titulo:'Lo que queda por delante',
      audiencia:12000000
    }
  },
  {
    capitulo2:{
      titulo:'Sangría',
      audiencia:13000000
    }
  }
]
)

```

2. Imprimir todos los documentos de la colección 'series'.
3. Imprimir los datos de todas las temporadas de la serie 'The big bang theory'.
4. Imprimir los datos de todos los capítulos de la primer temporada de la serie de 'The big bang theory'.
5. Imprimir solo el primer capítulo de la primer temporada de 'The Walking Dead'.



## 18. Campo \_id generado por MongoDB

Cuando creamos un documento podemos o no iniciar el campo `_id`, en la mayoría de los ejemplos hemos definido nosotros el campo `_id` y su valor, recordemos que no pueden repetirse.

Si no definimos nosotros el campo `_id` se genera en forma automática:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)

db.libros.find()
```

Tenemos como resultado:

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.libros.find().pretty()
{
  "_id" : ObjectId("5c39e2c7687b2322fe53eba6"),
  "titulo" : "El aleph",
  "autor" : "Borges",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 20,
  "cantidad" : 50
},
{
  "_id" : ObjectId("5c39e2c7687b2322fe53eba7"),
  "titulo" : "Martin Fierro",
  "autor" : "Jose Hernandez",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 50,
  "cantidad" : 12
}

```

El valor que se genera es de la clase ObjectId y tiene la característica de poder generarse con un valor único aunque se esté generando en servidores distintos.

Uno de los objetivos fundamentales de emplear MongoDB es poder tener servidores distribuidos, la posibilidad de definir una clave primaria única y no tener que comunicarse entre los servidores se resuelve utilizando la clase ObjectId.

No se puede generar un campo numérico autoincremental como en los gestores de bases de datos relacionales, esto obligaría a sincronizar dicho campo con otros servidores y generaría cuellos de botella y excesivo tráfico.

MongoDB fue diseñado para ser una base de datos distribuida, es fundamental poder generar identificadores únicos en un entorno fragmentado con múltiples servidores.

Un objeto de la clase ObjectId requiere 12 bytes y representan:

- Los primeros 4 bytes indican la cantidad de segundos desde el 1 de enero de 1970 (época unix)  
Cuando se ordena por el \_id luego los documentos se encontrarán organizados según el tiempo que se lo insertó.
- Los 3 bytes siguientes del ObjectId son un identificador único de la máquina en la que se generó, esto nos garantiza que los diferentes servidores no generarán ObjectId iguales.

- Los 2 bytes siguientes evitan que haya colisiones de ObjectId en el mismo servidor, el mismo se genera con el Id del número de proceso que lo generó.
- Los primeros 9 bytes de un ObjectId garantizan su unicidad en todas las máquinas y procesos durante un segundo. Los últimos 3 bytes son simplemente un contador incremental que es responsable de la unicidad dentro de un segundo en un solo proceso. Esto permite generar hasta 16.777.216 ObjectIds únicos por proceso en un solo segundo.

## 19. Tipo de dato Date en MongoDB

Hemos utilizado hasta ahora varios tipos de datos cuando inicializamos campos de un documento:

- String : Permiten almacenar cadenas de caracteres en formato UTF-8
- Integer32 : Valores entero numérico
- Integer64 : Valores entero numérico
- Double : Almacena valores de punto flotante
- Object : Almacena un documento embebido
- Array : Permite almacenar un arreglo con elementos de distinto tipo
- Boolean : Permite almacenar un valor true o false

Veamos un ejemplo como podemos almacenar un tipo de dato Date tomando la fecha y hora actual del servidor.

## Problema

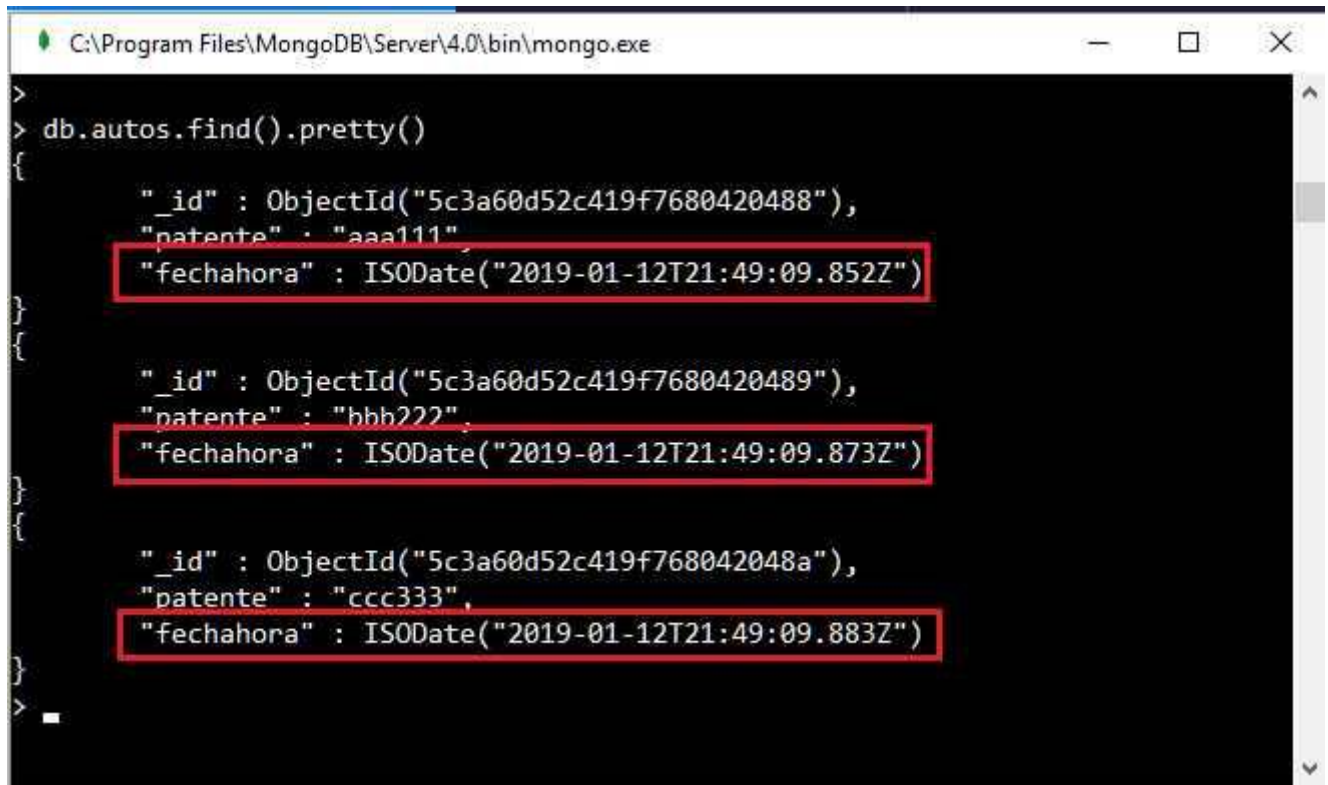
Una playa de estacionamiento cada vez que ingresa un vehículo crea un documento donde almacena la patente y la fecha y hora de ingreso .

```
use base1
db.autos.drop()

db.autos.insertOne(
  {
    patente : 'aaa111',
    fechahora : new Date()
  }
)
db.autos.insertOne(
  {
    patente : 'bbb222',
    fechahora : new Date()
  }
)
db.autos.insertOne(
  {
    patente : 'ccc333',
    fechahora : new Date()
  }
)

db.autos.find().pretty()
```

La representación de los datos de tipo Date cuando llamamos al método find es:



```
> C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
>
> db.autos.find().pretty()
{
  "_id" : ObjectId("5c3a60d52c419f7680420488"),
  "patente" : "aaa111",
  "fechahora" : ISODate("2019-01-12T21:49:09.852Z")
}
{
  "_id" : ObjectId("5c3a60d52c419f7680420489"),
  "patente" : "bbb222",
  "fechahora" : ISODate("2019-01-12T21:49:09.873Z")
}
{
  "_id" : ObjectId("5c3a60d52c419f768042048a"),
  "patente" : "ccc333",
  "fechahora" : ISODate("2019-01-12T21:49:09.883Z")
}
>
```

Para almacenar la fecha se utiliza el estándar [ISO 8601](https://www.iso.org/standard/52083.html) que tiene un formato:

YYYY-MM-DDTHH:MM:SS

Podemos almacenar una fecha particular cuando creamos el objeto de la clase Date:

```
use base1
db.empleados.drop()

db.empleados.insertOne(
{
  _id : 20456234,
  nombre : 'Rodriguez Pablo',
  fechaingreso : new Date(2010,0,31)
}
)
db.empleados.insertOne(
{
  _id : 17488834,
  nombre : 'Gomez Ana',
  fechaingreso : new Date(2001,11,1)
}
)
db.empleados.insertOne(
{
  _id : 23463564,
  nombre : 'Juarez Carla',
```

```

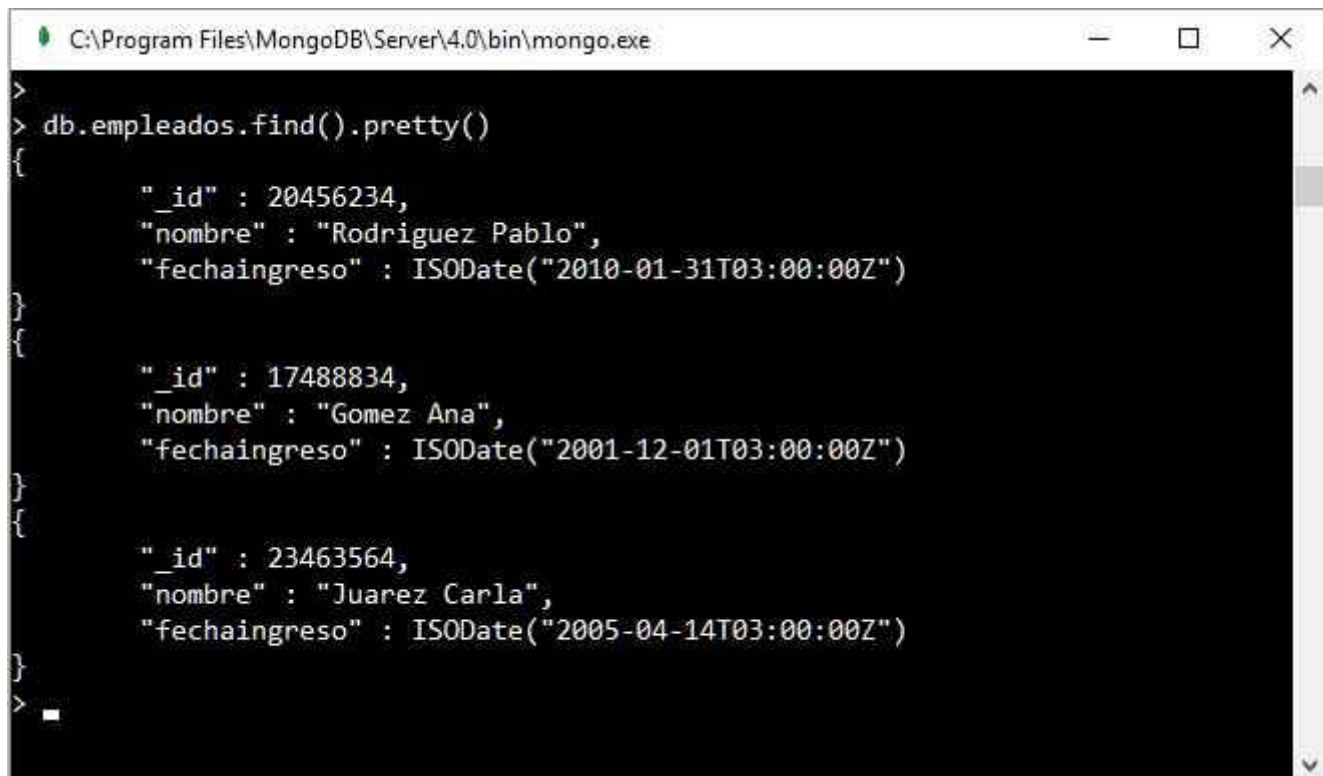
    fechaingreso : new Date(2005,3,14)
  }
)

db.empleados.find().pretty()

```

Como el shell de MongoDB está implementado en JavaScript debemos indicar al crear un objeto de la clase Date para el mes un valor comprendido entre 0 y 11.

Podemos ver las fechas almacenadas en el campo 'fechaingreso', y que la parte de la hora está en cero por no pasarlas cuando creamos el objeto de la clase Date:



```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
>
> db.empleados.find().pretty()
{
  "_id" : 20456234,
  "nombre" : "Rodriguez Pablo",
  "fechaingreso" : ISODate("2010-01-31T03:00:00Z")
}
{
  "_id" : 17488834,
  "nombre" : "Gomez Ana",
  "fechaingreso" : ISODate("2001-12-01T03:00:00Z")
}
{
  "_id" : 23463564,
  "nombre" : "Juarez Carla",
  "fechaingreso" : ISODate("2005-04-14T03:00:00Z")
}
>

```

Si necesitamos que los datos de empleados se recuperen en forma ordenada por el campo 'fechaingreso' debemos codificar la siguiente consulta:

```
db.empleados.find().pretty().sort({fechaingreso:1})
```

## Problemas propuestos (1.11)

1. Crear la colección 'alumnos' en la base de datos 'base1' (eliminar la colección previamente), cargar luego 3 documentos:

```
use base1

db.alumnos.drop()

db.alumnos.insertOne(
  {
    _id: 20456123,
    apellido: 'Gonzalez',
    nombre: 'Ana',
    domicilio: 'Colon 123',
    fechanacimiento: new Date(1990,7,15)
  }
)
db.alumnos.insertOne(
  {
    _id: 45123845,
    apellido: 'Juarez',
    nombre: 'Bernardo',
    domicilio: 'Sucre 456',
    fechanacimiento: new Date(1964,0,1)
  }
)
db.alumnos.insertOne(
  {
    _id: 16567512,
    apellido: 'Perez',
    nombre: 'Laura',
    domicilio: '21 de Septiembre 3233',
    fechanacimiento: new Date(1972,3,2)
  }
)

db.alumnos.find().pretty()
```

2. Imprimir todos los documentos de la colección alumnos.
3. Imprimir solo el apellido y la fecha de nacimiento.
4. Imprimir todos los datos ordenados por la fecha de nacimiento de mayor a menor.
5. Imprimir todos los alumnos que nacieron a partir de 1970.

## 20. Tipo de dato Date en MongoDB

MongoDB permite definir la carga de datos binarios (por ejemplo archivos jpg, png, pdf etc.) en un campo de un documento, siempre y cuando no supere el límite de 16Mb definido por el gestor de base de datos por motivos de eficiencia. Si tenemos que almacenar archivos muy grandes como podrían ser un archivo mp4 con una película debemos utilizar otra técnica, como por ejemplo que en el campo del documento se almacene la URL donde se encuentra la película.

Desde el shell de Mongo estamos bastante limitados para la carga de un campo de tipo binario pero podemos hacerlo creando una variable y almacenando el valor retornado por BinData:

```
use base1

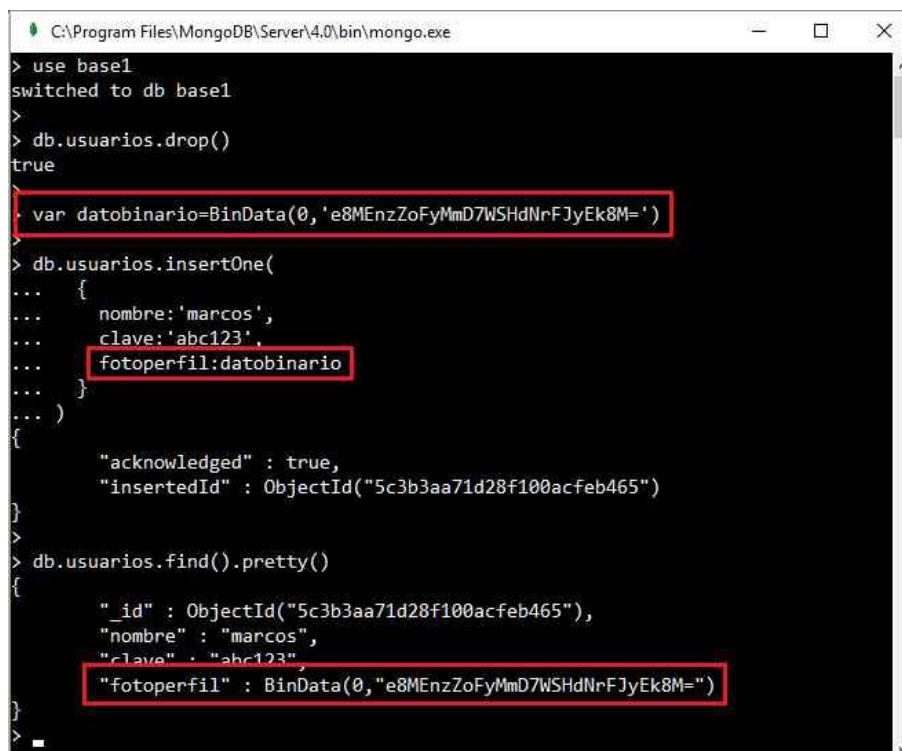
db.usuarios.drop()

var datobinario=BinData(0,'e8MEnzZoFyMmD7WSHdNrFJyEk8M=')

db.usuarios.insertOne(
{
  nombre:'marcos',
  clave:'abc123',
  fotoperfil:datobinario
}
)

db.usuarios.find().pretty()
```

La ejecución del bloque anterior nos almacena un documento con un campo con formato 'Binary Data':



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> use base1
switched to db base1
>
> db.usuarios.drop()
true
> var datobinario=BinData(0,'e8MEnzZoFyMmD7WSHdNrFJyEk8M=')
> db.usuarios.insertOne(
... {
...   nombre:'marcos',
...   clave:'abc123',
...   fotoperfil:datobinario
... }
... )
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5c3b3aa71d28f100acfeb465")
}
> db.usuarios.find().pretty()
{
  "_id" : ObjectId("5c3b3aa71d28f100acfeb465"),
  "nombre" : "marcos",
  "clave" : "abc123",
  "fotoperfil" : BinData(0,"e8MEnzZoFyMmD7WSHdNrFJyEk8M=")
}
```



Recordemos que en el mundo real cuando almacenemos campos de tipo Binary Data lo haremos desde una aplicación creada con alguno de los tantos lenguajes que soporta MongoDB como pueden ser: C, C++, C#, Go, Java, Node.js, Perl, PHP, Python, Ruby o Scala.

Las ventajas de tener los datos binarios dentro de los documentos son:

- Proporciona el motor de base de datos alta disponibilidad y replicación de datos.
- Una arquitectura de aplicación más simple al tener centralizado todos los datos.
- Cuando se usan conjuntos de réplicas distribuidas geográficamente, MongoDB distribuirá automáticamente los datos a centros de datos geográficamente distintos.
- El almacenamiento de datos en la base de datos aprovecha los mecanismos de autenticación y seguridad de MongoDB.
- Mejor rendimiento en los accesos a datos. Accediendo al documento contamos con todos los datos y no tenemos que hacer referencias a otras locaciones con datos.

Hay que tener en cuenta que un documento puede crecer solo hasta 16Mb, en el caso de ser archivos binarios con tamaños superiores hay que utilizar otra técnica para su almacenamiento, lo más común es almacenarlos la referencia mediante una URL.

Hay otra técnica de trocear el archivo y almacenarlo en una sucesión de documentos, puede consultar en la página oficial de [MongoDB](#).

## 21. Indices en MongoDB

Otro elemento esencial en MongoDB son los índices.

Los índices sirven para acceder a los documentos de una colección rápidamente, acelerando la localización de la información.

Los índices se emplean para facilitar la obtención de información de una colección. El índice de una colección desempeña la misma función que el índice de un libro: permite encontrar datos rápidamente; en el caso de las colecciones, localiza documentos.

Mongo accede a los documentos de dos maneras:

1) recorriendo las colecciones; comenzando desde el principio y extrayendo los documentos que cumplen las condiciones de la consulta; lo cual implica posicionar las cabezas lectoras, leer el dato, controlar si coincide con lo que se busca (como si pasáramos una a una las páginas de un libro buscando un tema específico).

2) empleando índices; recorriendo la estructura de árbol del índice para localizar los documentos y extrayendo los que cumplen las condiciones de la consulta (comparando con un libro, diremos que es como leer el índice y luego de encontrar el tema buscado, ir directamente a la página indicada).

Un índice posibilita el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, MongoDB debe recorrer secuencialmente todos los documentos de una colección para encontrar un documento en particular.

Los índices son estructuras asociadas a las colecciones, una colección almacena los campos indexados y se crean para acelerar las consultas.

Entonces, el objetivo de un índice es acelerar la recuperación de información. La indexación es una técnica que optimiza el acceso a los datos, mejora el rendimiento acelerando las consultas y otras operaciones como pueden ser las modificaciones y borrados. Es útil cuando las colecciones contienen cientos de miles o millones de documentos.

La desventaja es que consume espacio en el disco y genera costo de mantenimiento (tiempo y recursos) cuando se efectúan inserciones y modificaciones.

Es importante identificar el o los campos por los que sería útil crear un índice.

No se recomienda crear índices sobre campos que no se usan con frecuencia en consultas o en colecciones muy pequeñas.

Los cambios sobre las colecciones, como inserción, actualización o eliminación de documentos, son incorporados automáticamente en los archivos índices.

En MongoDB el primer índice que se crea en forma automática corresponde al campo `_id`, recordemos que dicho campo podemos asignarle un valor nosotros o hacer que se cree automáticamente.

Los otros campos a los cuales se le crearán índices deberán ser seleccionados por nosotros dependiendo de las consultas de nuestra aplicación a la base de datos.

Los índices que se crean en MongoDB funcionan de manera similar a como funcionan en gestores de bases de datos relacionales (MySQL, SqlServer, PostgreSQL etc)

## 22. Indices en MongoDB – Simples y Compuestos

Un índice es simple cuando se hace por un único campo del documento, debemos utilizar el método 'createIndex' e indicar el campo por el cual queremos generar el archivo índice.

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

db.libros.createIndex( {titulo : 1} )

db.libros.find({}, { titulo:1 }).sort({ titulo:1 }).pretty()
```

Cuando ejecutamos el bloque anterior al crearse el índices nos informa de los índices anteriores que tenía la colección y la nueva cantidad de índices:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe

> db.libros.createIndex( {titulo : 1} )
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

Para indexar en orden inverso cuando se crea el índice debemos especificar un -1:

```
db.libros.createIndex( {titulo : -1} )
```

Los datos quedan ordenados de la 'z' a la 'a'.

## Indices compuestos

Un índice es compuesto cuando se hace por dos o más campos del documento, debemos utilizar también el método 'createIndex' e indicar los campos por los cuales generar el archivo índice.

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
```

```
}
)
db.libros.insertOne(
{
  _id: 4,
  titulo: 'Java en 10 minutos',
  editorial: ['Siglo XXI'],
  precio: 45,
  cantidad: 1
}
)

db.libros.createIndex( {titulo : 1, autor : 1} )

db.libros.find({titulo:'Aprenda PHP',autor:'Mario Molina'}).pretty()
```

Hemos creado un índice en la colección libros por los campos titulo y autor:

```
db.libros.createIndex( {titulo : 1, autor : 1} )
```

Luego si la colección almacena millones de documentos, una consulta por los campos título y autor es casi instantánea:

```
db.libros.find({titulo:'Aprenda PHP',autor:'Mario Molina'}).pretty()
```

## Acceso solo a los datos de los índices.

Para maximizar la eficiencia de una consulta podemos filtrar por campos que se encuentran indexados y recuperar datos directamente del índice. En estos casos MongoDB no debe acceder a los documentos de la colección, solo debe acceder a los datos que se encuentran en el archivo índice.

```
use base1
db.articulos.drop()

db.articulos.insertOne(
{
  _id: 1,
  nombre: 'MULTIFUNCION HP DESKJET 2675',
  rubro: 'impresora',
  precio: 3000,
  stock: 20
}
)
db.articulos.insertOne(
{
  _id: 2,
  nombre: 'MULTIFUNCION EPSON EXPRESSION XP241',
  rubro: 'impresora',
  precio: 3700,
  stock: 5
}
)
db.articulos.insertOne(
{
```

```
_id: 3,
nombre: 'LED 19 PHILIPS',
rubro: 'monitor',
precio: 4500,
stock: 2
}
)
db.articulos.insertOne(
{
  _id: 4,
  nombre: 'LED 22 PHILIPS',
  rubro: 'monitor',
  precio: 5700,
  stock: 4
}
)
db.articulos.insertOne(
{
  _id: 5,
  nombre: 'LED 27 PHILIPS',
  rubro: 'monitor',
  precio: 12000,
  stock: 1
}
)
db.articulos.insertOne(
{
  _id: 6,
  nombre: 'LOGITECH M90',
  rubro: 'mouse',
  precio: 300,
  stock: 4
}
)
db.articulos.createIndex( {rubro : 1, _id : 1} )
db.articulos.find({rubro:'monitor'},{rubro:1, _id:1})
```

## Conocer estadísticas de la consulta.

Si queremos conocer información sobre los requerimientos reales de una consulta podemos utilizar el método explain:

```
db.articulos.find({rubro:'monitor'},{rubro:1, _id:1}).explain('executionStats')
```

Nos retorna muchos datos que pueden ser útiles para planificar que estructuras de índices definir:

```
> db.articulos.find({rubro:'monitor'},{rubro:1, _id:1}).explain('executionStats')
{
  "queryPlanner" : {
```

```

"plannerVersion" : 1,
"namespace" : "base1.articulos",
"indexFilterSet" : false,
"parsedQuery" : {
  "rubro" : {
    "$eq" : "monitor"
  }
},
"winningPlan" : {
  "stage" : "PROJECTION",
  "transformBy" : {
    "rubro" : 1,
    "_id" : 1
  },
  "inputStage" : {
    "stage" : "IXSCAN",
    "keyPattern" : {
      "rubro" : 1,
      "_id" : 1
    },
    "indexName" : "rubro_1__id_1",
    "isMultiKey" : false,
    "multiKeyPaths" : {
      "rubro" : [ ],
      "_id" : [ ]
    },
    "isUnique" : false,
    "isSparse" : false,
    "isPartial" : false,
    "indexVersion" : 2,
    "direction" : "forward",
    "indexBounds" : {
      "rubro" : [
        "[\"monitor\\", \"monitor\"]"
      ],
      "_id" : [
        "[MinKey, MaxKey]"
      ]
    }
  },
  "rejectedPlans" : [ ]
},
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 3,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 3,
  "totalDocsExamined" : 0,
  "executionStages" : {
    "stage" : "PROJECTION",
    "nReturned" : 3,
    "executionTimeMillisEstimate" : 0,
    "works" : 4,
    "advanced" : 3,
    "needTime" : 0,
    "needYield" : 0,
    "saveState" : 0,
    "restoreState" : 0,

```



```

        "isEOF" : 1,
        "invalidates" : 0,
        "transformBy" : {
            "rubro" : 1,
            "_id" : 1
        },
        "inputStage" : {
            "stage" : "IXSCAN",
            "nReturned" : 3,
            "executionTimeMillisEstimate" : 0,
            "works" : 4,
            "advanced" : 3,
            "needTime" : 0,
            "needYield" : 0,
            "saveState" : 0,
            "restoreState" : 0,
            "isEOF" : 1,
            "invalidates" : 0,
            "keyPattern" : {
                "rubro" : 1,
                "_id" : 1
            },
            "indexName" : "rubro_1__id_1",
            "isMultiKey" : false,
            "multiKeyPaths" : {
                "rubro" : [ ],
                "_id" : [ ]
            },
            "isUnique" : false,
            "isSparse" : false,
            "isPartial" : false,
            "indexVersion" : 2,
            "direction" : "forward",
            "indexBounds" : {
                "rubro" : [
                    "["monitor\\", \\monitor\\"]"
                ],
                "_id" : [
                    "[MinKey, MaxKey]"
                ]
            },
            "keysExamined" : 3,
            "seeks" : 1,
            "dupsTested" : 0,
            "dupsDropped" : 0,
            "seenInvalidated" : 0
        }
    },
    "serverInfo" : {
        "host" : "diego-PC",
        "port" : 27017,
        "version" : "4.0.5",
        "gitVersion" : "3739429dd92b92d1b0ab120911a23d50bf03c412"
    },
    "ok" : 1
}

```

Entre los datos podemos encontrar:

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 3,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 3,
  "totalDocsExamined" : 0,
```

Podemos ver que en el campo "totalDocsExamined" almacena un cero, esto quiere decir que no se consultaron documentos, sino directamente los datos almacenados en los índices (esto aumenta mucho la eficiencia de la consulta cuando tenemos millones de documentos).

Si cambiamos la consulta y recuperamos también el precio:

```
db.articulos.find({rubro:'monitor'},{rubro:1, _id:1, precio:1}).explain('executionStats')
```

Luego podemos comprobar que si se accedieron a los documentos a parte del archivo índice:

```
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 3,
  "executionTimeMillis" : 0,
  "totalKeysExamined" : 3,
  "totalDocsExamined" : 3,
```

## Indices únicos

Podemos asegurarnos que un documento tiene valores distintos para uno o varios campos definiendo un índice único. La sintaxis para la definición de un índice único:

```
use base1

db.clientes.drop()

db.clientes.insertOne(
  {
    _id: 1,
    nombre: 'Perez Ana',
    dni: '20439455',
    domicilio: 'San Martin 222',
    provincia: 'Santa Fe'
  }
)
```

```
db.clientes.insertOne(
  {
    _id: 2,
    nombre: 'Garcia Juan',
    dni: '21495834',
    domicilio: 'Rivadavia 333',
    provincia: 'Buenos Aires'
  }
)

db.clientes.insertOne(
  {
    _id: 3,
    nombre: 'Perez Luis',
    dni: '20888722',
    domicilio: 'Sarmiento 444',
    provincia: 'Buenos Aires'
  }
)

db.clientes.createIndex({dni:1},{unique:true})

db.clientes.find()

db.clientes.insertOne(
  {
    _id: 4,
    nombre: 'Peña Lucas',
    dni: '20888722',
    domicilio: 'General Paz 323',
    provincia: 'Buenos Aires'
  }
)
```

Para crear un índice único debemos pasar un segundo parámetro al método 'createIndex':

```
db.clientes.createIndex({dni:1},{unique:true})
```

Luego si intentamos ingresar un documento en la colección 'clientes' con un 'dni' repetido, MongoDB no lo permite:

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.clientes.createIndex({dni:1},{unique:true})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
>
> db.clientes.find()
{ "_id" : 1, "nombre" : "Perez Ana", "dni" : "20439455", "domicilio" : "San Martin 222", "provincia" : "Santa Fe" }
{ "_id" : 2, "nombre" : "Garcia Juan", "dni" : "21495834", "domicilio" : "Rivadavia 333", "provincia" : "Buenos Aires" }
{ "_id" : 3, "nombre" : "Perez Luis", "dni" : "20888722", "domicilio" : "Sarmiento 444", "provincia" : "Buenos Aires" }
>
> db.clientes.insertOne(
...   {
...     _id: 4,
...     nombre: 'Peña Lucas',
...     dni: '20888722',
...     domicilio: 'General Paz 323',
...     provincia: 'Buenos Aires'
...   }
... )
2019-01-14T15:43:12.093-0300 E QUERY [js] WriteError: E11000 duplicate key error collection: base1.clientes index:
dni_1 dup key: { : "20888722" } :
WriteError({
  "index" : 0,
  "code" : 11000,
  "errmsg" : "E11000 duplicate key error collection: base1.clientes index: dni_1 dup key: { : \"20888722\" }",
  "op" : {
    "_id" : 4,
    "nombre" : "Peña Lucas",
    "dni" : "20888722",
    "domicilio" : "General Paz 323",
    "provincia" : "Buenos Aires"
  }
})
WriteError@src/mongo/shell/bulk_api.js:461:48
Bulk/mergeBatchResults@src/mongo/shell/bulk_api.js:841:49
Bulk/executeBatch@src/mongo/shell/bulk_api.js:906:13

```

## 23. Indices con campos de tipo documento y array

MongoDB permite acceder a sus documentos y crear índices en campos de tipo documento. Los campos de tipo documento se pueden combinar con campos de nivel superior en índices compuestos y, aunque tienen algunas particularidades en algunos aspectos, se comportan principalmente como se comportan los campos de índice "normales".

Podemos crear un índice simple de un subcampo, como por ejemplo la 'calle' del campo 'direccion':

```
use base1
db.clientes.drop()

db.clientes.insertOne(
  {
    _id: 1,
    nombre: 'Martinez Victor',
    mail: 'mvictor@gmail.com',
    direccion: {
      calle: 'Colon',
      numero: 620,
      codigopostal: 5000
    }
  }
)
db.clientes.insertOne(
  {
    _id: 2,
    nombre: 'Alonso Carlos',
    mail: 'acarlos@gmail.com',
    direccion: {
      calle: 'Colon',
      numero: 150,
      codigopostal: 5000
    }
  }
)
db.clientes.insertOne(
  {
    _id: 3,
    nombre: 'Gonzalez Marta',
    mail: 'gmarta@outlook.com',
    direccion: {
      calle: 'Colon',
      numero: 1200,
      codigopostal: 5000
    }
  }
)
db.clientes.insertOne(
  {
    _id: 4,
    nombre: 'Ferrero Ariel',
    mail: 'fariel@yahoo.com',
    direccion: {
      calle: 'Dean Funes',
      numero: 23,
```

```
        codigopostal: 5002
      }
    }
  )
  db.clientes.insertOne(
    {
      _id: 5,
      nombre: 'Fernandez Diego',
      mail: 'fdiego@gmail.com',
      direccion: {
        calle: 'Dean Funes',
        numero: 561,
        codigopostal: 5002
      }
    }
  )

  db.clientes.createIndex({'direccion.calle':1})

  db.clientes.find({'direccion.calle':'Dean Funes'})
```

Podemos también crear un índice que incluya todos los subcampos:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: {
      nombre: 'Borges',
      nacionalidad: 'Argentina'
    },
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: {
      nombre: 'Jose Hernandez',
      nacionalidad: 'Argentina'
    },
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: {
      nombre: 'Mario Molina',
```

```

        nacionalidad: 'Española'
    },
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
}
)
db.libros.insertOne(
{
    _id: 4,
    autor: {
        nombre: 'Java en 10 minutos',
        nacionalidad: 'Española'
    },
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
}
)

db.libros.createIndex({autor:1})

db.libros.find({'autor.nombre':'Java en 10 minutos','autor.nacionalidad':'Española'})

```

El índice se utiliza solo en los casos que en la consulta indiquemos ambos subcampos: 'nombre' y 'nacionalidad'.

Podemos también incluir un campo de tipo arreglo cuando se define un índice.

## Operador \$text

El operador \$ text query realiza búsquedas de texto en una colección con un índice de texto.

\$text tokenizará la cadena de búsqueda utilizando espacios en blanco y la mayoría de la puntuación como delimitadores, y realizará un OR lógico de todos esos tokens en la cadena de búsqueda.

Por ejemplo, puede utilizar la consulta siguiente para encontrar todas las tiendas que contengan términos de la lista "coffe", "shop" y "java":

```
db.stores.find( { $text: { $search: "java coffee shop" } } )
```

También puede buscar frases exactas al envolverlas en comillas dobles. Por ejemplo, a continuación encontrará todos los documentos que contienen "java" o "coffe shop"

```
db.stores.find( { $text: { $search: "java \"coffee shop\"" } } )
```

Para excluir una palabra, puede añadir un carácter "-". Por ejemplo, para encontrar todas las tiendas que contienen "java" o "shop" pero no "coffee", utilice lo siguiente:

```
db.stores.find( { $text: { $search: "java shop -coffee" } } )
```

MongoDB devolverá sus resultados no ordenado de forma predeterminada. Sin embargo, las consultas de búsqueda de texto calcularán una puntuación de pertinencia para cada documento que especifica qué tan bien un documento coincide con la consulta.

Para ordenar los resultados en orden de puntuación de relevancia, debe proyectar explícitamente el campo \$meta (Retorna el metadato asociado con un documento) con "textScore" (operador que asigna una puntuación a cada documento que contiene el término de búsqueda en los campos indexados. La puntuación representa la relevancia de un documento para una consulta de búsqueda de texto determinada.)

```
db.stores.find(
    { $text: { $search: "java coffee shop" } },
    { score: { $meta: "textScore" } }
).sort( { score: { $meta: "textScore" } } )
```

Los índices de texto admiten consultas de búsqueda de texto en campos que contienen contenido de cadena. Los índices de texto mejoran el rendimiento al buscar palabras o frases específicas dentro del contenido de una cadena.

Una colección sólo puede tener **un** índice de texto, pero ese índice puede cubrir varios campos.

La indexación de campos comúnmente consultados aumenta las posibilidades de cubrir esas consultas. Las consultas cubiertas son consultas que pueden satisfacerse completamente utilizando un índice, sin examinar ningún documento. Esto optimiza el rendimiento de las consultas. Para crear un índice de texto, utilice el siguiente prototipo:

```
db.<collection>.createIndex(
  {
    <field1>: "text",
    <field2>: "text",
    ...
  }
)
```

Ej: `db.libros.createIndex({titulo:"text"},{default_language: "spanish"})`

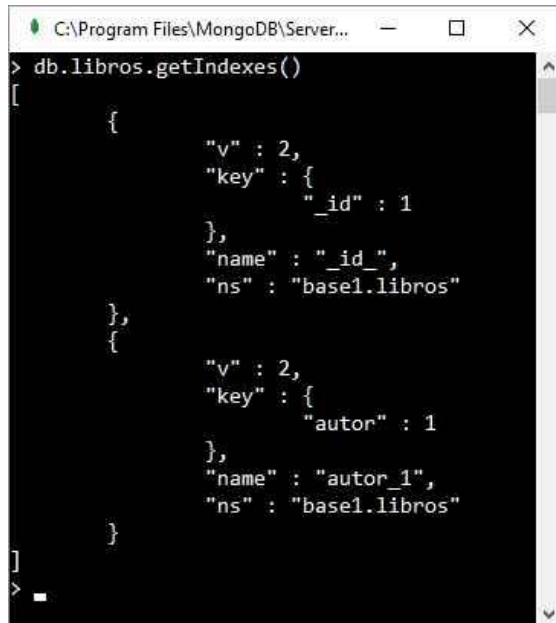


## 24. Indices- eliminación

Para conocer los índices que tiene una colección hacemos uso del método 'getIndexes', nos devuelve información de cada uno de los índices:

```
db.libros.getIndexes()
```

Tenemos como resultado una salida similar a:



```
C:\Program Files\MongoDB\Server...
> db.libros.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "base1.libros"
  },
  {
    "v" : 2,
    "key" : {
      "autor" : 1
    },
    "name" : "autor_1",
    "ns" : "base1.libros"
  }
]
```

Para eliminar un índice constamos con el método 'dropIndex' al cual le debemos pasar el nombre del índice a eliminar:

```
db.libros.dropIndex('autor_1')
```

Luego si consultamos nuevamente los índices presentes en la colección veremos que ha desaparecido 'autor\_1':



```
C:\Program Files\MongoDB\Server...
> db.libros.dropIndex('autor_1')
{ "nIndexesWas" : 2, "ok" : 1 }
> db.libros.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "base1.libros"
  }
]
```

Tengamos en cuenta que no podemos eliminar el índice que crea MongoDB sobre el campo \_id.

Para eliminar un índice si no queremos llamar a `getIndexes` podemos indicar los campos por los que se creó el índice. Crearemos un índice y luego lo eliminaremos:

```
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)
db.libros.insertOne(
  {
    _id: 3,
    titulo: 'Aprenda PHP',
    autor: 'Mario Molina',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 50,
    cantidad: 20
  }
)
db.libros.insertOne(
  {
    _id: 4,
    titulo: 'Java en 10 minutos',
    editorial: ['Siglo XXI'],
    precio: 45,
    cantidad: 1
  }
)

db.libros.createIndex( {titulo : 1} )

db.libros.getIndexes()

db.libros.dropIndex( {titulo : 1} )

db.libros.getIndexes()
```

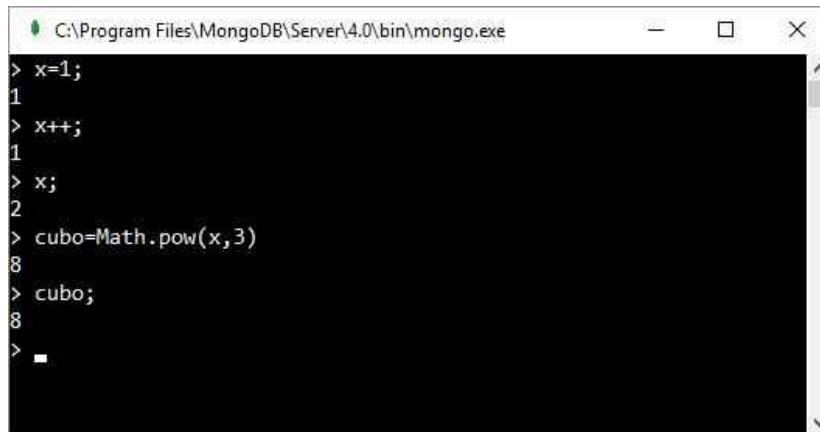
## 25. MongoDB Shell y JavaScript

Desde los primeros conceptos hemos utilizado el programa MongoDB shell para comunicarnos con nuestro servidor.

En MongoDB shell podemos ejecutar todas las funciones del lenguaje JavaScript, para comprobar esto ejecutemos:

```
x=1;
x++;
x;
cubo=Math.pow(x,3)
cubo;
```

Tenemos como resultado al ejecutar el bloque de comandos JavaScript en el MongoDB shell:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> x=1;
1
> x++;
1
> x;
2
> cubo=Math.pow(x,3)
8
> cubo;
8
> _
```

Podemos incluso codificar una función y luego llamarla:

```
function mayor(x1,x2) {
  if (x1>x2)
    return x1;
  else
    return x2;
};

mayor(10,3);
mayor(6,34);
```

Tenemos como resultado al ejecutar el bloque de comandos JavaScript en el MongoDB shell:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> function mayor(x1,x2) {
...   if (x1>x2)
...     return x1;
...   else
...     return x2;
... };
>
> mayor(10,3);
10
> mayor(6,34);
34
>
```

## Personalización del prompt de MongoDB shell

Ahora que sabemos que tenemos un entorno en JavaScript en MongoDB shell podemos personalizar por ejemplo el shell modificando la variable 'prompt' asignando una función:

```
prompt = function() {
  return (new Date())+"> ";
};
```

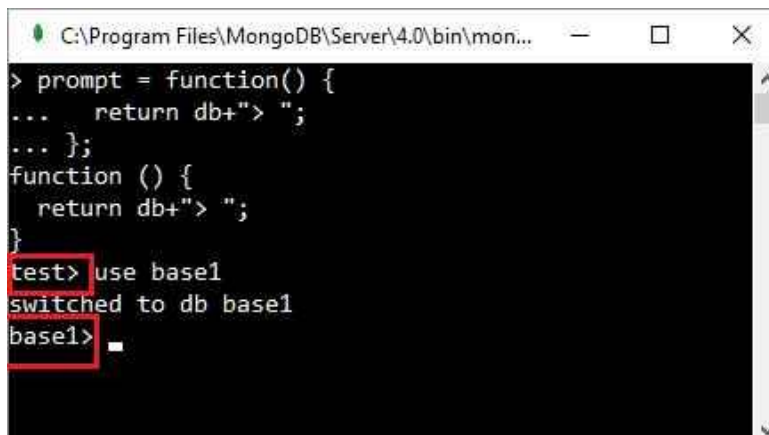
El prompt de MongoDB shell nos muestra la fecha y hora actual:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> prompt = function() {
...   return (new Date())+"> ";
... };
function () {
  return (new Date())+"> ";
}
Mon Jan 21 2019 12:55:27 GMT-0300> use base1
switched to db base1
Mon Jan 21 2019 12:55:34 GMT-0300> 
```

Algo más útil es que el prompt muestre el nombre de la base de datos activa, esto lo logramos asignando la siguiente función:

```
prompt = function() {
  return db+"> ";
};
```

El prompt de MongoDB shell nos muestra la base de datos en uso:



```
> prompt = function() {  
...   return db+"> ";  
... };  
function () {  
   return db+"> ";  
}  
test> use base1  
switched to db base1  
base1> 
```

## Crear una gran colección de datos mediante código JavaScript

Podemos utilizar la funcionalidad de JavaScript en el shell de MongoDB para poblar una colección con datos de prueba por ejemplo:

```
use base1  
db.articulos.drop()  
for(i = 1; i <= 10; i++) {  
  db.articulos.insertOne(  
    {  
      _id: i,  
      nombre: 'nombre'+i  
    }  
  );  
}  
db.articulos.find().pretty();
```

Como resultado tenemos:

```

C:\Program Files\MongoDB\Server\4.0\bin\...
> use base1
switched to db base1
> db.articulos.drop()
true
> for(i = 1; i <= 10; i++) {
...   db.articulos.insertOne(
...     {
...       _id: i,
...       nombre: 'nombre'+i
...     }
...   );
... }
{ "acknowledged" : true, "insertedId" : 10 }
> db.articulos.find().pretty();
{ "_id" : 1, "nombre" : "nombre1" }
{ "_id" : 2, "nombre" : "nombre2" }
{ "_id" : 3, "nombre" : "nombre3" }
{ "_id" : 4, "nombre" : "nombre4" }
{ "_id" : 5, "nombre" : "nombre5" }
{ "_id" : 6, "nombre" : "nombre6" }
{ "_id" : 7, "nombre" : "nombre7" }
{ "_id" : 8, "nombre" : "nombre8" }
{ "_id" : 9, "nombre" : "nombre9" }
{ "_id" : 10, "nombre" : "nombre10" }
>

```

## Desplegar el código JavaScript de los métodos de MongoDB

Podemos desplegar el código JavaScript de cada método visto en conceptos anteriores simplemente indicando su nombre desde el shell:

```

use base1
db.articulos.insertOne

```

Muy útil si tenemos conocimientos de JavaScript para conocer su implementación:

```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> use base1
switched to db base1
> db.articulos.insertOne
function (document, options) {
    var opts = Object.extend({}, options || {});

    // Add _id ObjectId if needed
    document = this.addIdIfNeeded(document);

    // Get the write concern
    var writeConcern = this._createWriteConcern(opts);

    // Result
    var result = {acknowledged: (writeConcern && writeConcern.w == 0) ? false : true};

    // Use bulk operation API already in the shell
    var bulk = this.initializeOrderedBulkOp();
    bulk.insert(document);

    try {
        // Execute insert
        bulk.execute(writeConcern);
    } catch (err) {
        if (err instanceof BulkWriteError) {
            if (err.hasWriteErrors()) {
                throw err.getWriteErrorAt(0);
            }

            if (err.hasWriteConcernError()) {
                throw err.getWriteConcernError();
            }
        }

        throw err;
    }

    if (!result.acknowledged) {
        return result;
    }

    // Set the inserted id
    result.insertedId = document._id;

    // Return the result
    return result;
}
>

```

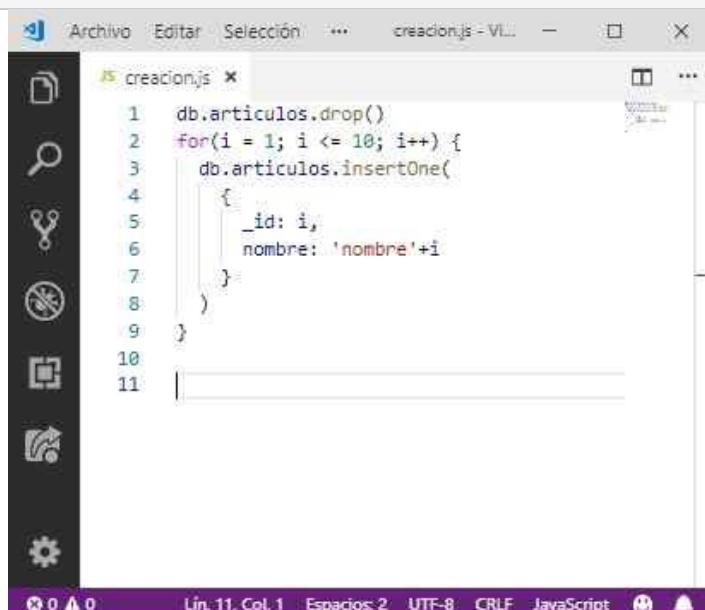
## 26. MongoDB Shell cargar y ejecutar un archivo JavaScript \*.js

Vimos en el concepto anterior que desde la aplicación de consola de MongoDB (shell) podemos ejecutar comandos JavaScript, ahora veremos que dichos comandos podemos tenerlos en un archivo \*.js y cargarlos mediante un comando.

Si tenemos que implementar script de mediana o gran complejidad lo más adecuado es utilizar un editor de texto y grabarlo en un archivo \*.js

Por ejemplo codifiquemos con nuestro editor favorito el siguiente bloque en un archivo llamado 'creacion.js' y lo guardemos en la carpeta c:\\scriptmongodb\\:

```
db.articulos.drop()
for(i = 1; i <= 10; i++) {
  db.articulos.insertOne(
    {
      _id: i,
      nombre: 'nombre'+i
    }
  )
}
```



Luego de haber grabado el archivo con el nombre 'creacion.js' en la carpeta c:\\scriptmongodb procedamos a ejecutar el comando load desde el shell de MongoDB:

```
load("c:\\scriptmongodb\\creacion.js")
```

Luego de ejecutar la función load nos informará si se ejecutó el bloque de comandos en forma exitosa:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> load("c:\\scriptmongodb\\creacion.js")
true
>
```

Podemos comprobar que la colección artículos se creó correctamente y se cargaron los 10 documentos:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> load("c:\\scriptmongodb\\creacion.js")
true
> db.articulos.find().pretty()
{ "_id" : 1, "nombre" : "nombre1" }
{ "_id" : 2, "nombre" : "nombre2" }
{ "_id" : 3, "nombre" : "nombre3" }
{ "_id" : 4, "nombre" : "nombre4" }
{ "_id" : 5, "nombre" : "nombre5" }
{ "_id" : 6, "nombre" : "nombre6" }
{ "_id" : 7, "nombre" : "nombre7" }
{ "_id" : 8, "nombre" : "nombre8" }
{ "_id" : 9, "nombre" : "nombre9" }
{ "_id" : 10, "nombre" : "nombre10" }
>
```

Hemos pasado el path absoluto (c:\\scriptmongodb\\) donde se encuentra el archivo 'creacion.js', si el archivo \*.js se encuentra en la misma carpeta desde donde iniciamos MongoDB shell podemos especificar solamente el nombre del archivo al llamar a load:

```
load("creacion.js")
```

Si queremos ejecutar mongo.exe desde cualquier carpeta de Windows 10, debemos configurar una variable de entorno con la carpeta donde hemos instalado MongoDB.

Solo así podemos ejecutar 'mongo.exe' desde cualquier carpeta del sistema operativo simplemente indicando su nombre:

```

Node.js command prompt - mongo
C:\scriptmongodb>mongo
MongoDB shell version v4.0.5
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("857df0bf-3281-49c9-b137-d35007dbb735") }
MongoDB server version: 4.0.5
Server has startup warnings:
2019-01-21T23:08:16.485-0300 I CONTROL [initandlisten]
2019-01-21T23:08:16.485-0300 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-01-21T23:08:16.485-0300 I CONTROL [initandlisten] **      Read and write access to data and configuration i
s unrestricted.
2019-01-21T23:08:16.485-0300 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>

```

## Comandos de MongoDB en un archivo \*.js

En un script no podemos utilizar los comandos show dbs, use, show collections etc. pero podemos sustituirlos llamando a métodos:

use base1	<code>db = db.getSiblingDB('base1')</code>
show dbs, show databases	<code>db.adminCommand('listDatabases')</code>
show collections	<code>db.getCollectionNames()</code>
show users	<code>db.getUsers()</code>
show roles	<code>db.getRoles({showBuiltinRoles: true})</code>
show log	<code>db.adminCommand({ 'getLog' : '' })</code>
show logs	<code>db.adminCommand({ 'getLog' : '*' })</code>
it	<pre> cursor = db.collection.find(); while ( cursor.hasNext() ) {     printjson( cursor.next() ); } </pre>

En un script siempre que necesitemos hacer salidas por pantalla debemos utilizar la función print y printjson, por ejemplo:

```
printjson(db.adminCommand('listDatabases'))
```

Modifiquemos el archivo 'creacion.js' con el siguiente código y luego volvamos a cargarlo mediante la función 'load':

```
printjson(db.adminCommand('listDatabases'))
db = db.getSiblingDB('base1')

print(db.getCollectionNames())
db.articulos.drop()
for(i = 1; i <= 10; i++) {
  db.articulos.insertOne(
    {
      _id: i,
      nombre: 'nombre'+i
    }
  )
}
cursor = db.articulos.find();
while ( cursor.hasNext() ) {
  printjson(cursor.next());
}
```

Podemos comprobar que se muestran los datos:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> load("c:\\scriptmongodb\\creacion.js")
{
  "databases" : [
    {
      "name" : "admin",
      "sizeOnDisk" : 32768,
      "empty" : false
    },
    {
      "name" : "base1",
      "sizeOnDisk" : 311296,
      "empty" : false
    },
    {
      "name" : "config",
      "sizeOnDisk" : 98304,
      "empty" : false
    },
    {
      "name" : "local",
      "sizeOnDisk" : 73728,
      "empty" : false
    },
    {
      "name" : "test",
      "sizeOnDisk" : 65536,
      "empty" : false
    }
  ],
  "totalSize" : 581632,
  "ok" : 1
}
alumnos,articulos,autos,clientes,empleados,ingreso,libros,medicamentos,usuarios
{ "_id" : 1, "nombre" : "nombre1" }
{ "_id" : 2, "nombre" : "nombre2" }
{ "_id" : 3, "nombre" : "nombre3" }
{ "_id" : 4, "nombre" : "nombre4" }
{ "_id" : 5, "nombre" : "nombre5" }
{ "_id" : 6, "nombre" : "nombre6" }
{ "_id" : 7, "nombre" : "nombre7" }
{ "_id" : 8, "nombre" : "nombre8" }
{ "_id" : 9, "nombre" : "nombre9" }
{ "_id" : 10, "nombre" : "nombre10" }
true
>
```

## 27. MongoDB Shell - conectarnos a un servidor remoto

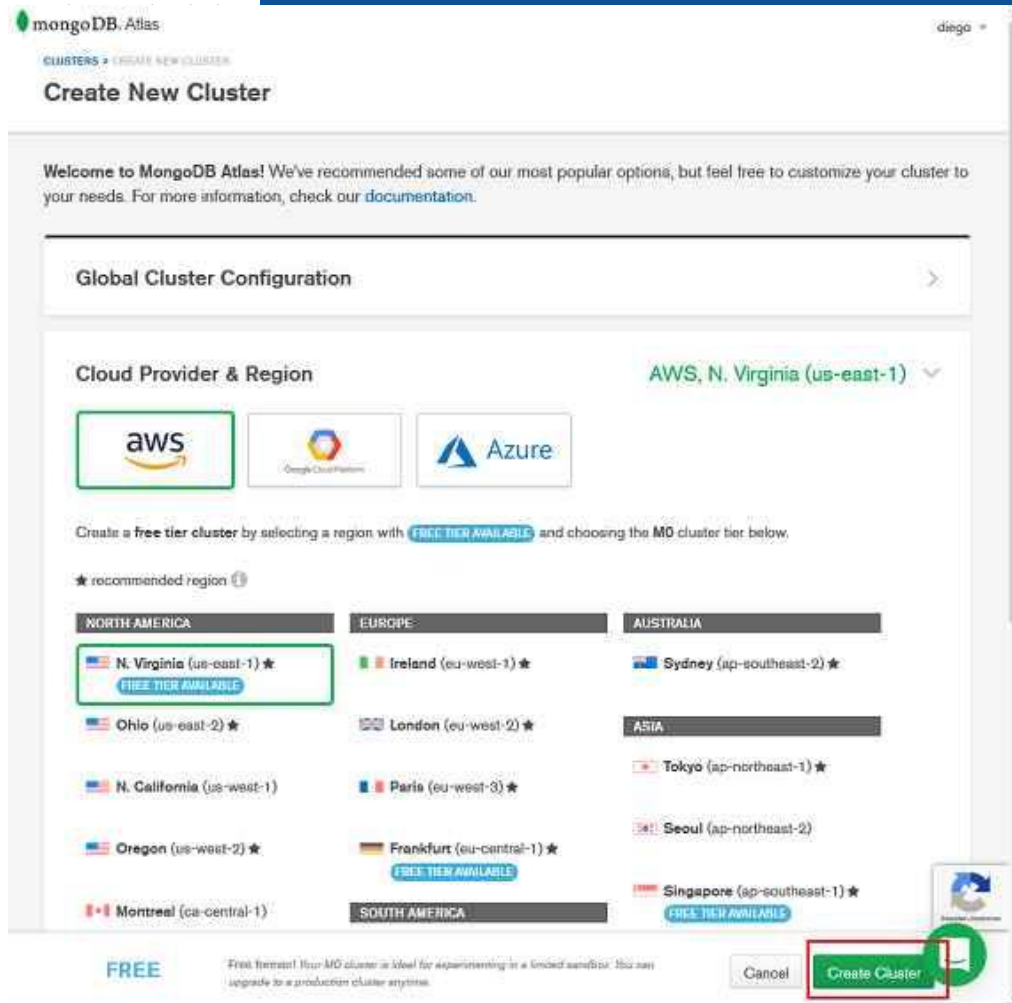
Hasta ahora hemos trabajado con nuestro servidor instalado en forma local. Veremos ahora los pasos que debemos dar para conectarnos a un servidor remoto desde MongoDB shell.

Crearemos un servidor en forma gratuita, un servicio que presta la empresa que desarrolla MongoDB.

Primero debemos obtener una cuenta, para ello debemos registrarnos en [MongoDB Atlas](https://www.mongodb.com/es/cloud/atlas).



Debemos crear como primer paso un "Cluster":



Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#).

### Global Cluster Configuration

Cloud Provider & Region: **AWS, N. Virginia (us-east-1)**

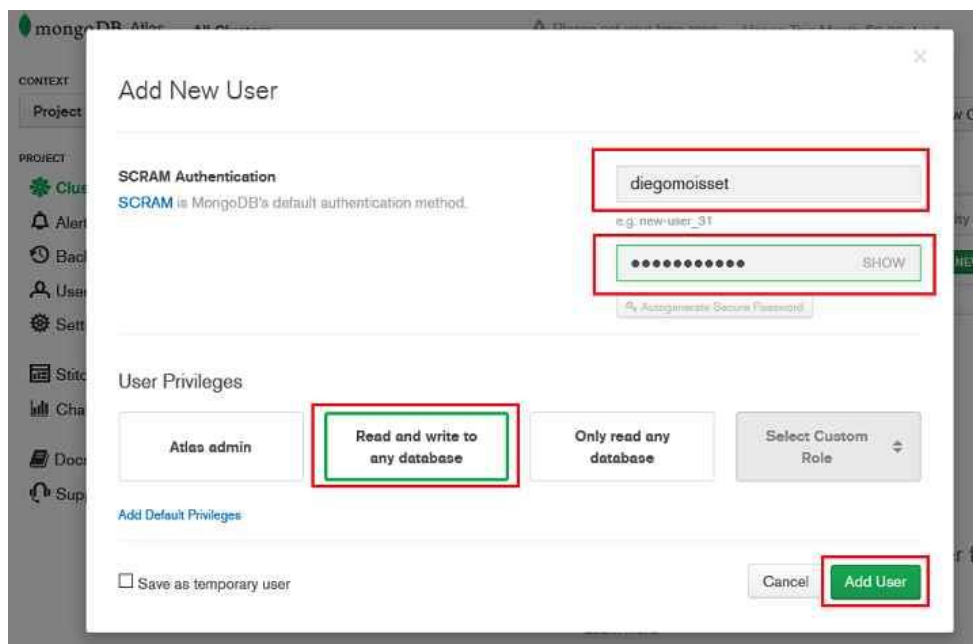
Create a **free tier** cluster by selecting a region with **FREE TIER AVAILABLE** and choosing the **M0** cluster tier below.

★ recommended region ⓘ

NORTH AMERICA	EUROPE	AUSTRALIA
<b>N. Virginia (us-east-1) ★</b> <b>FREE TIER AVAILABLE</b>	Ireland (eu-west-1) ★	Sydney (ap-southeast-2) ★
Ohio (us-east-2) ★	London (eu-west-2) ★	<b>ASIA</b>
N. California (us-west-1)	Paris (eu-west-3) ★	Tokyo (ap-northeast-1) ★
Oregon (us-west-2) ★	Frankfurt (eu-central-1) ★ <b>FREE TIER AVAILABLE</b>	Seoul (ap-northeast-2)
Montreal (ca-central-1)	<b>SOUTH AMERICA</b>	Singapore (ap-southeast-1) ★ <b>FREE TIER AVAILABLE</b>

**FREE** Free tiered! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Seguidamente debemos crear un usuario ingresando su nombre y clave:



### Add New User

SCRAM Authentication  
SCRAM is MongoDB's default authentication method.

Username: **diegomoiisset**  
e.g. new-user\_31

Password: **.....**   
Automatically Secure Password

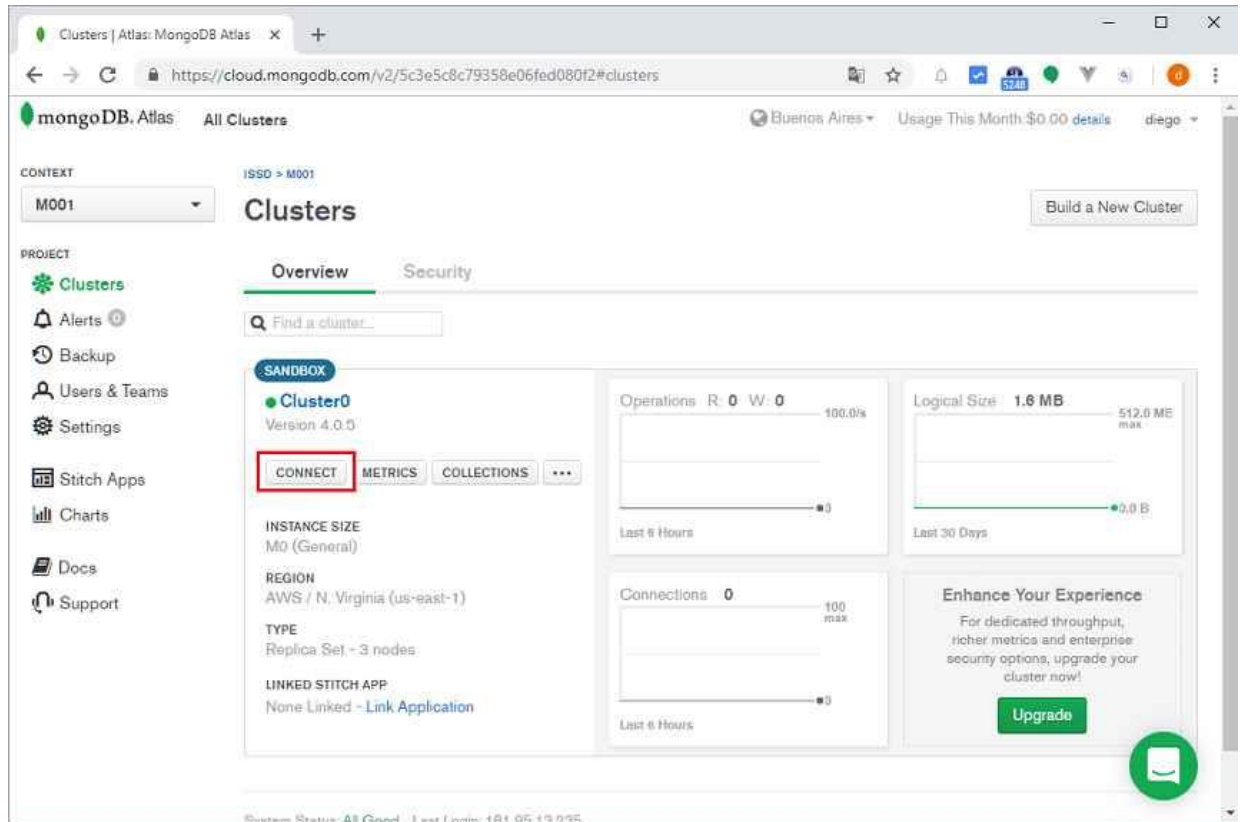
User Privileges:

[Add Default Privileges](#)

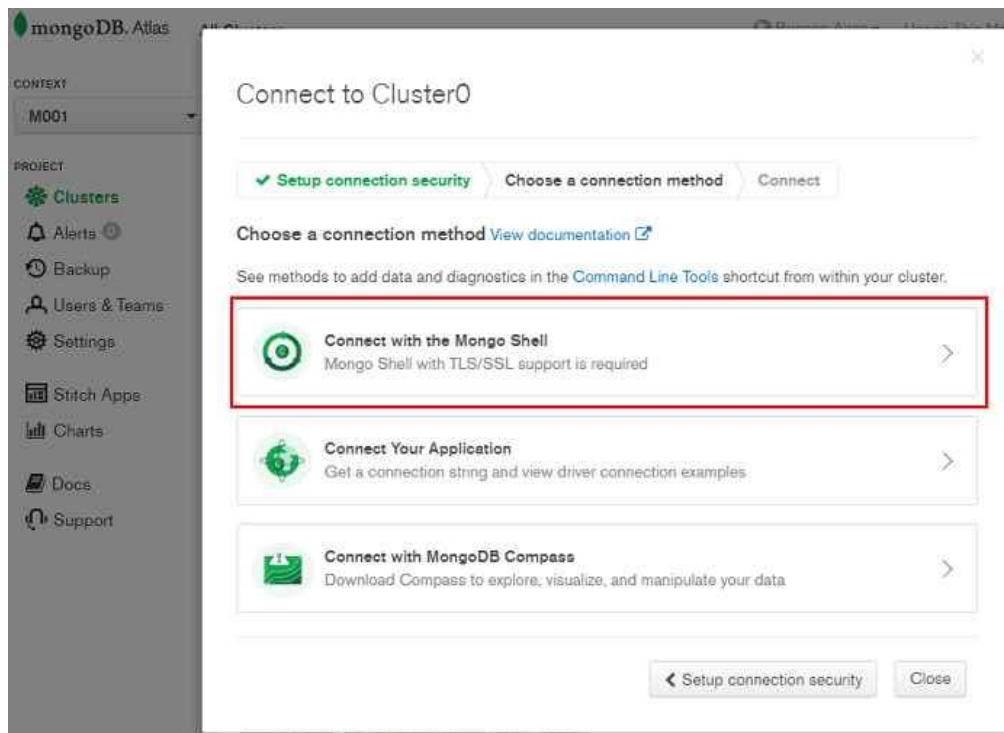
☐ Save as temporary user

Una vez registrado se nos suministran todos los accesos para poder crear bases de datos en nuestro servidor remoto.

Desde su panel de control podemos obtener la cadena de conexión para conectarnos al servidor remoto MongoDB desde nuestro shell:

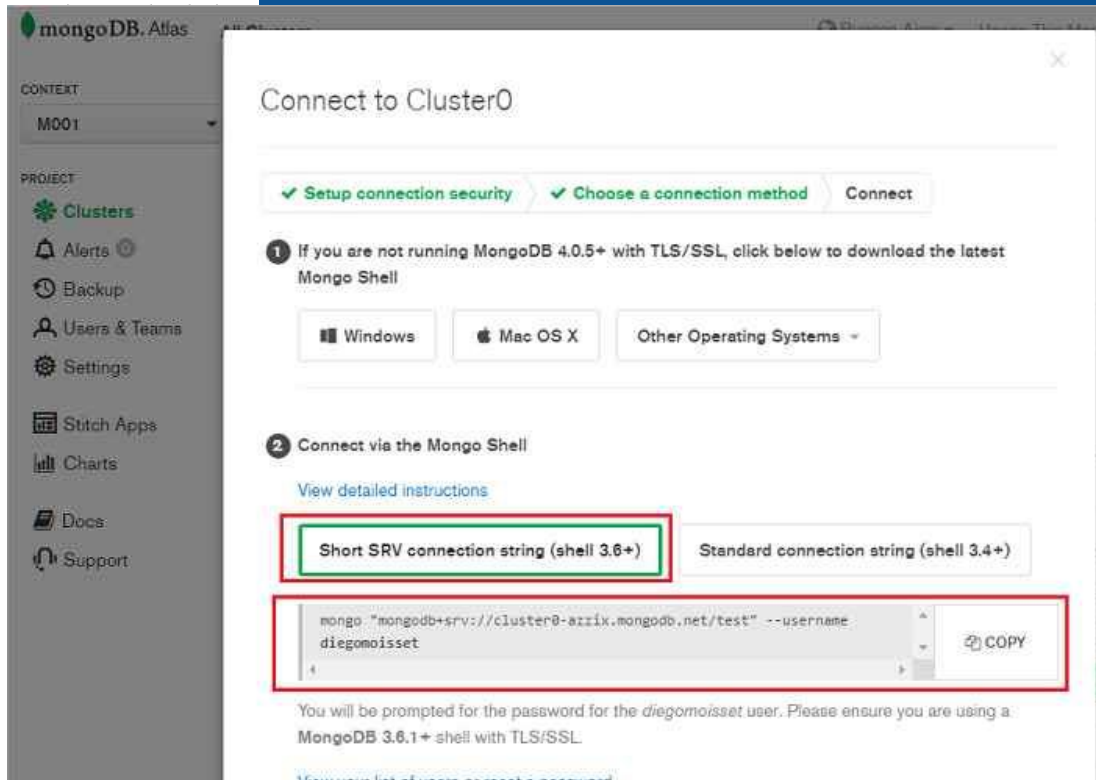


Presionando el botón "connect" aparece un diálogo que nos informa las distintas formas que tenemos para conectarnos al servidor según la aplicación que utilicemos:



Utilizaremos:





Desde una consola de Windows procedemos a ejecutar 'mongosh' con la cadena indicada en el panel de control, se nos solicita la clave definida para el usuario que creamos anteriormente:

```

C:\Users\diego>mongo "mongodb+srv://cluster0-vv9dc.mongodb.net/test" --username diegomoisset
MongoDB shell version v4.0.5
Enter password:
connecting to: mongodb://cluster0-shard-00-01-vv9dc.mongodb.net:27017,cluster0-shard-00-02-vv9dc.mongodb.net:27017,cluster0-shard-00-00-vv9dc.mongodb.net:27017/test?authSource=admin&gssapiServiceName=mongodb&replicaSet=Cluster0-shard-00ssl=true
2019-01-22T08:55:52.781-0300 I NETWORK [js] Starting new replica set monitor for Cluster0-shard-0/cluster0-shard-00-01-vv9dc.mongodb.net:27017,cluster0-shard-00-02-vv9dc.mongodb.net:27017,cluster0-shard-00-00-vv9dc.mongodb.net:27017
2019-01-22T08:55:53.535-0300 I NETWORK [ReplicaSetMonitor-TaskExecutor] Successfully connected to cluster0-shard-00-02-vv9dc.mongodb.net:27017 (1 connections now open to cluster0-shard-00-02-vv9dc.mongodb.net:27017 with a 5 second timeout)
2019-01-22T08:55:53.667-0300 I NETWORK [js] Successfully connected to cluster0-shard-00-00-vv9dc.mongodb.net:27017 (1 connections now open to cluster0-shard-00-00-vv9dc.mongodb.net:27017 with a 5 second timeout)
2019-01-22T08:55:53.883-0300 I NETWORK [js] changing hosts to Cluster0-shard-0/cluster0-shard-00-00-vv9dc.mongodb.net:27017,cluster0-shard-00-02-vv9dc.mongodb.net:27017 from Cluster0-shard-0/cluster0-shard-00-00-vv9dc.mongodb.net:27017,cluster0-shard-00-01-vv9dc.mongodb.net:27017,cluster0-shard-00-02-vv9dc.mongodb.net:27017
2019-01-22T08:55:54.446-0300 I NETWORK [ReplicaSetMonitor-TaskExecutor] Successfully connected to cluster0-shard-00-00-vv9dc.mongodb.net:27017 (1 connections now open to cluster0-shard-00-00-vv9dc.mongodb.net:27017 with a 5 second timeout)
2019-01-22T08:55:54.628-0300 I NETWORK [js] Successfully connected to cluster0-shard-00-01-vv9dc.mongodb.net:27017 (1 connections now open to cluster0-shard-00-01-vv9dc.mongodb.net:27017 with a 5 second timeout)
2019-01-22T08:55:55.347-0300 I NETWORK [ReplicaSetMonitor-TaskExecutor] Successfully connected to cluster0-shard-00-02-vv9dc.mongodb.net:27017 (1 connections now open to cluster0-shard-00-02-vv9dc.mongodb.net:27017 with a 5 second timeout)
Implicit session: session { "id" : UUID("3f4f030a-617e-45f3-82c1-b3ff25b06b62") }
MongoDB server version: 4.0.5
Error while trying to show server startup warnings: user is not allowed to do action [getLog] on [admin]
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
  
```

Todas las actividades que desarrollemos ahora se efectuarán en el servidor remoto que acabamos de crear. Probemos de crear una base de datos, colecciones etc:

```
show dbs

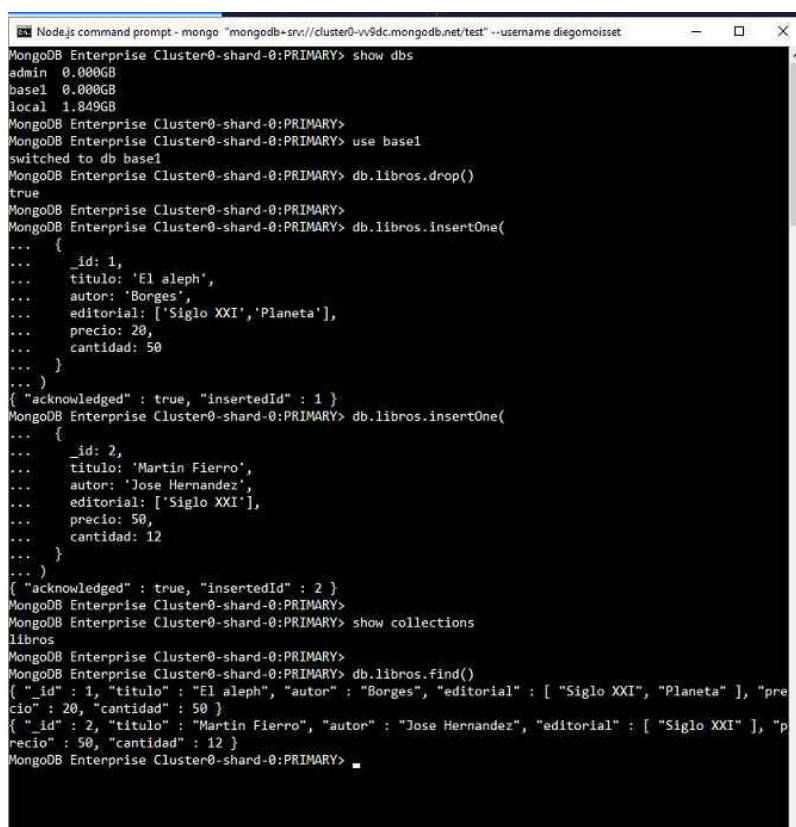
use base1
db.libros.drop()

db.libros.insertOne(
  {
    _id: 1,
    titulo: 'El aleph',
    autor: 'Borges',
    editorial: ['Siglo XXI', 'Planeta'],
    precio: 20,
    cantidad: 50
  }
)
db.libros.insertOne(
  {
    _id: 2,
    titulo: 'Martin Fierro',
    autor: 'Jose Hernandez',
    editorial: ['Siglo XXI'],
    precio: 50,
    cantidad: 12
  }
)

show collections

db.libros.find()
```

La ejecución genera la siguiente salida en nuestro shell de MongoDB conectado al servidor remoto:



```
Node.js command prompt - mongo "mongodb+srv://cluster0-vv9dc.mongodb.net/test?--username=diegomaisset"
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show dbs
admin 0.000GB
base1 0.000GB
local 1.849GB
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
MongoDB Enterprise Cluster0-shard-0:PRIMARY> use base1
switched to db base1
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.libros.drop()
true
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.libros.insertOne(
... {
...   _id: 1,
...   titulo: 'El aleph',
...   autor: 'Borges',
...   editorial: ['Siglo XXI', 'Planeta'],
...   precio: 20,
...   cantidad: 50
... }
... )
{ "acknowledged" : true, "insertedId" : 1 }
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.libros.insertOne(
... {
...   _id: 2,
...   titulo: 'Martin Fierro',
...   autor: 'Jose Hernandez',
...   editorial: ['Siglo XXI'],
...   precio: 50,
...   cantidad: 12
... }
... )
{ "acknowledged" : true, "insertedId" : 2 }
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show collections
libros
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.libros.find()
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
MongoDB Enterprise Cluster0-shard-0:PRIMARY>
```