Tema 4. Recuperar documentos con Mongo

Tabla de contenidos

En este tema estudiamos en detalle como hacer las queries en MongoDB, con las que básicamente podemos:

- Realizar queries para obtener documentos mediante las funciones *find* y *findOne*.
- Realizar quieries por rango, inclusión en arrays, distinción, y un amplio abanico de posibilidades gracias a los *$-conditionals*.
- Realizar metaoperaciones sobre los cursores (objeto devuelto por cada query) para limitar el número de resultados, ordenarlos o fijar un *offset*.

Empecemos.

4.1 Introducción a *find*

El método *find* es el que se utiliza para hacer queries en Mongo. Es el equivalente al comando *select* en el modelo relacional. Al consultar una colección, Mongo nos devolverá un subconjunto de documentos, que variará desde el conjunto vació hasta la colección completa.

Find tiene varios parámetros de entrada, el primero de ellos especifica los documentos que queremos recuperar, esto es, el criterio de búsqueda. Este primer parámetro tiene el formato de documento JSON, al igual que hemos visto por ejemplo para la función *update*.

```
db.coleccion1.find({ clave1:valor1 })
```

La función anterior nos devolvería documentos de la colección *coleccion1* cuyo campo *clave1* tenga un valor igual a *valor1*.

El valor por defecto para este primer parámetro es {}, que significa "cualquier documento". Por tanto una query como *db.coleccion1.find({ })* devolverá todos los documentos de la colección *coleccion1*. Como es el valor por defecto para el criterio de la consulta es equivalente hacer *find({ })* y *find( ).*

Debemos ser conscientes que el objeto que utilizamos como criterio de búsqueda es sensible a los tipos de datos con los que trabaja Mongo. De esta forma, si por ejemplo queremos recuperar los alumnos con *edad* igual a 17, que es un valor número, haríamos:

```
> db.alumno.find({edad:17})
```

En el caso de consultar por un campo *string*, solo es necesario especificarlo correctamente entrecomillado.

```
> db.alumno.find({name:"Antonio"})
```

Igual que en SQL podemos especificar los campos que queremos recuperar (select campo1, campo2, campo3 from tabla …), podemos hacerlo también en Mongo. Para ello solo tenemos que pasar un segundo parámetro al método *find*, en el que decimos los que queremos.

db.coleccion1.find({ },**{ clave3:1, clave4:1 }**)

Los campos que queremos que sean devueltos les pondremos el valor 1. El campo _id siempre se devuelve.

También podemos especificar los campos que no queremos que sean devueltos, en este caso como valor para cada una de esos campos, pondremos cero. El resultado será que se devolverán el resto de documentos.

db.coleccion1.find({ },**{ clave3:0, clave4:0 }**)

Ahora vamos a ver unos pequeños ejemplos directamente en Mongo, para ello arranque tanto su servidor como su cliente *shell* de Mongo y ejecute los comandos de guardado que hay en el fichero *4.1.libros_ejemplo.json* de los recursos del tema. Puede bien ir documento a documento o seleccionar todo el texto del fichero y pegarlo directamente en el shell de Mongo.

Estos ejemplos son libros de un supuesto catálogo que podríamos tener para una tienda online. Vea previamente los datos que estamos guardando para una mejor comprensión de los ejemplos.

Cada libro tiene un campo "enstock", que guarda un valor *boolean* donde *true* significa que tenemos copias del libro para servir, y *false* que no. Lo primero que haremos es recuperar todos los libros que tenemos disponibles. Para ello solo tenemos que ejecutar la función *find* sobre la colección *libro* pidiendo el valor *true* para el campo *enstock.* Además solo vamos a recuperar los campos titulo, editorial y precio, esto lo especificamos con el segundo parámetro.

```
> db.libro.find({enstock:true},{titulo:1,editorial:1,precio:1})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "editorial" :
"Planeta", "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "editorial" :
"Plaza & Janes", "precio" : 21.75 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y
fuego 1", "editorial" : "Gigamesh", "precio" : 9.5 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo", "editorial" :
"Embolsillo", "precio" : 11 }
```

Si le resulta incómodo leer los resultados en este formato de salida, pruebe el *formatter pretty()* al finalizar el comando de búsqueda.

```
> db.libro.find({enstock:true},{titulo:1,editorial:1,precio:1}).pretty()
{
      "_id" : "9788408117117",
      "titulo" : "Circo Máximo",
      "editorial" : "Planeta",
      "precio" : 21.75
}
{
      "_id" : "9788401342158",
      "titulo" : "El juego de Ripper",
      "editorial" : "Plaza & Janes",
      "precio" : 21.75
}
{
      "_id" : "9788496208919",
      "titulo" : "Juego de tronos: Canción de hielo y fuego 1",
```

```
        "editorial" : "Gigamesh",
        "precio" : 9.5
}
{
        "_id" : "9788415140054",
        "titulo" : "La princesa de hielo",
        "editorial" : "Embolsillo",
        "precio" : 11

}
```

Como hemos dicho, podemos especificar más criterios de búsqueda, que se interpretarán como condiciones AND, y que por tanto los documentos que se recuperen deberán cumplir todas las condiciones.

Con la siguiente query, además de los libros en stock, exigiremos que su precio sea de 21.75.

```
>
db.libro.find({enstock:true,precio:21.75},{titulo:1,editorial:1,preci
o:1}).pretty()
{
        "_id" : "9788408117117",
        "titulo" : "Circo Máximo",
        "editorial" : "Planeta",
        "precio" : 21.75
}
{
        "_id" : "9788401342158",
        "titulo" : "El juego de Ripper",
        "editorial" : "Plaza & Janes",
        "precio" : 21.75

}
```

4.1.1 Operadores de comparación

Hasta ahora hemos visto como recuperar documentos cuando el valor de una o varias claves es igual a una constante especificada, pero MongoDB permite utilizar los siguientes operadores de comparación:

- $lt (less than), se corresponde con la condición < (menor estricto).
- $lte (less than or equal), se corresponde con la condición <= (menor o igual).
- $gt (greather than), se corresponde con la condición > (mayor estricto).
- $gte (greather than or equal), se corresponde con la condición >= (mayor o igual).

La sintaxis es:

clave: { $operador: valor }

Así si por ejemplo queremos recuperar los libros con un precio mayor que 10 lo haríamos utilizando el operador $gt sobre el campo *precio*:

```
> db.libro.find({"precio":{$gt:10}},{titulo:1,precio:1})
```

```
{"_id" : "9788408117117", "titulo" : "Circo Máximo", "precio" : 21.75 }
{"_id" : "9788401342158", "titulo" : "El juego de Ripper", "precio" : 21.75}
{"_id" : "9788415140054", "titulo" : "La princesa de hielo", "precio" : 11}
{"_id" : "9788408113331","titulo":"Las carreras de Escorpio","precio":17.23}
{"_id" : "9788468738895", "titulo" : "Las reglas del juego","precio" : 15.9}
```

Si ahora queremos recuperar los libros con un precio menor que 20, optamos por el operador $lt.

```
> db.libro.find({"precio":{$lt:20}},{titulo:1,precio:1})
{"_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de hielo y
fuego 1", "precio" : 9.5}
{"_id" : "9788415140054", "titulo" : "La princesa de hielo", "precio" : 11}
{"_id" : "9788408113331","titulo":"Las carreras de Escorpio","precio":17.23}
{"_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio" :15.9}
{"_id" : "9788499088075", "titulo" : "La ladrona de libros", "precio" :9.45}
```

Y para completar si queremos buscar el rango de documentos cuyo precio es mayor que 10 y menor que 20, lo haremos pendiendo ambas condiciones separadas por comas, que como hemos visto implica la condición AND.

```
> db.libro.find({"precio":{$gt:10,$lt:20}},{titulo:1,precio:1})
{"_id" : "9788415140054", "titulo" : "La princesa de hielo", "precio" : 11}
{"_id":"9788408113331", "titulo":"Las carreras de Escorpio","precio":17.23}
{"_id" : "9788468738895", "titulo" : "Las reglas del juego", "precio":15.9}
```

La búsqueda mediante operadores de comparación es especialmente útil para las fechas, ya que en la mayoría de las ocasiones buscar por igualdad de una fecha no suele tener mucho sentido. El funcionamiento es exactamente igual que para cualquier otro tipo de datos, solamente hay que tener un poco de cuidado de crear la fecha con el formato adecuado. Con el siguiente *find* recuperamos los libros posteriores al 1 de Enero de 2013.

```
> var d = new ISODate("2013-01-01T00:00:00Z")
> db.libro.find({fecha:{$gte:d}},{fecha:1})
{ "_id" : "9788408117117", "fecha" : ISODate("2013-08-29T00:00:00Z") }
{ "_id" : "9788401342158", "fecha" : ISODate("2014-03-01T00:00:00Z") }
{ "_id" : "9788408113331", "fecha" : ISODate("2013-06-04T00:00:00Z") }
{ "_id" : "9788468738895", "fecha" : ISODate("2014-02-06T00:00:00Z") }
```

4.1.2 Operador lógico OR

Para hacer queries donde el operador lógico que se quiere es OR, tenemos que distinguir entre:

- OR entre varios valores posibles de un mismo campo.
- OR entre varios campos del documento.

En el primer caso lo más recomendable es utilizar ***$in***, ya que acabamos teniendo una consulta más clara y de mejor rendimiento. La sintaxis es la siguiente:

clave: { $in: [valor1, valor2, valorN] }

Y puede leerse como que queremos documentos cuyo campo *clave* esté entre alguno de los valores especificados en el array, esto es, [valor1, valor2, valorN].

De esta forma, para recuperar por ejemplo los libros donde la editorial sea *Debolsillo, Planeta* ***o*** *Gigamesh,* lo hacemos con el operador *$in* y un array de Strings formado por los tres valores que queremos filtrar.

```
> db.libro.find(
{editorial:{$in:["Debolsillo","Planeta","Gigamesh"]}},
{titulo:1,editorial:1}).pretty()
{
     "_id" : "9788408117117",
     "titulo" : "Circo Máximo",
     "editorial" : "Planeta"
}
{
     "_id" : "9788496208919",
     "titulo" : "Juego de tronos: Canción de hielo y fuego 1",
     "editorial" : "Gigamesh"
}
{
     "_id" : "9788499088075",
     "titulo" : "La ladrona de libros",
     "editorial" : "Debolsillo"
}
{
     "_id" : "9788408113331",
     "titulo" : "Las carreras de Escorpio",
     "editorial" : "Planeta"
}
```

El operador opuesto a *$in* es ***$nin****,* que se lee como *not in.* Este devolverá el conjunto de documentos disjunto a $in, o lo que es lo mismo, los documentos cuyo campo clave **no** esté entre alguno de los valores especificados en el array. Su sintaxis es la siguiente:

clave: { $nin: [valor1, valor2, valorN] }

Podemos comprobar su funcionamiento cambiando la query anterior para devolvernos los libros del resto de editoriales.

```
> db.libro.find(
{editorial:{$nin:["Debolsillo","Planeta","Gigamesh"]}},
{titulo:1,editorial:1}).pretty()
{
     "_id" : "9788401342158",
     "titulo" : "El juego de Ripper",
     "editorial" : "Plaza & Janes"
}
{
     "_id" : "9788415140054",
     "titulo" : "La princesa de hielo",
     "editorial" : "Embolsillo"
}
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
```

Fíjese que también ha devuelto el título *"Las reglas del juego"*, que realmente no

tiene informado el campo editorial. Esto es porque el valor null obviamente se considera diferente a cualquiera de los especificados en el array.

El segundo caso para el operador OR es cuando queremos aplicarlo entre diferentes campos del documento. En tal circunstancia tendremos que utilizar el condicional ***$or***, esta es su sintaxis.

$or: [{clave1:valor1}, {clave2:valor2}, {claveN:valorN}]

Con el siguiente *find*, vamos a recuperar aquellos documentos que no estén en stock (*enstock:false*) o que no tengan editorial (*editorial:null*).

```
> db.libro.find({$or:[{enstock:false},{editorial:null}]},
{titulo:1,editorial:1,enstock:1}).pretty()
{
     "_id" : "9788499088075",
     "titulo" : "La ladrona de libros",
     "editorial" : "Debolsillo",
     "enstock" : false
}
{
     "_id" : "9788408113331",
     "titulo" : "Las carreras de Escorpio",
     "editorial" : "Planeta",
     "enstock" : false
}
{
     "_id" : "9788468738895",
     "titulo" : "Las reglas del juego",
     "enstock" : false
}
```

Como es de esperar, en cada una de las condiciones del *$or* podemos utilizar cualquiera de los operadores vistos hasta el momento (y los que veremos más adelante), consiguiendo de esta forma las queries tan complejas como necesitemos. Algunos ejemplos serían.

Documentos sin stock o cuya editorial no sea ni Planeta, ni Debolsillo.

```
> db.libro.find(
{$or:[{enstock:false},{editorial:{$nin:['Planeta','Debolsillo']}}]},
{titulo:1,editorial:1,enstock:1})
```

Documentos sin stock o con fecha anterior al 1 de Enero de 2011.

```
> db.libro.find(
{$or:[{enstock:false},
{fecha:{$lt:new ISODate("2011-01-01T00:00:00Z")}}]}
,{titulo:1,editorial:1,enstock:1})
```

4.1.3 $not

$not es lo que se conoce como un metacondicional, que son opeadores que pueden aplicarse por encima de cualquier otro criterio. Su sintaxis es sencilla, solo hay que ponerlo "fuera" de aquello que queremos negar.

$not: { criterio }

Y así directamente podemos recuperar por ejemplo, documentos que no tengan 480 paginas.

```
> db.libro.find( { paginas: {$not: { $eq:480 } } }
,{titulo:1,paginas:1} ).pretty()
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "paginas" :
1100 }
{
     "_id" : "9788496208919",
     "titulo" : "Juego de tronos: Canción de hielo y fuego 1",
     "paginas" : 793
}
{
     "_id" : "9788499088075",
     "titulo" : "La ladrona de libros",
     "paginas" : 544
}
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
{
     "_id" : "9788408113331",
     "titulo" : "Las carreras de Escorpio",
     "paginas" : 290
}
{
     "_id" : "9788468738895",
     "titulo" : "Las reglas del juego",
     "paginas" : 360
}
```

4.1.4 $exists

El operador *$exists* se utiliza para comprobar los documentos que tienen informado o no un campo determinado. Su sintaxis es la siguiente:

clave: { $exists: boolean}

Donde boolean puede ser *true* o *false*. En el caso de *true* devuelve documentos que tienen informado un campo, aunque su valor sea *null.* Este operador es útil porque debido a la ausencia de definición de esquema de datos de mongo, cada documento de una misma colección puede tener los campos que requiera, y al hacer consultas puede que no nos devuelva ciertos documentos bien porque no entra dentro de los valores exigidos, bien porque el campo no está definido.

Es importante no confundir *$exists* con el operador exists de SQL.

Veamos un ejemplo con dos comandos *find* donde utilizamos el operador *$exists* con ambos valores.

```
> db.libro.find({paginas:{$exists:true}},{paginas:1})
{ "_id" : "9788408117117", "paginas" : 1100 }
{ "_id" : "9788401342158", "paginas" : 480 }
{ "_id" : "9788496208919", "paginas" : 793 }
{ "_id" : "9788499088075", "paginas" : 544 }
{ "_id" : "9788408113331", "paginas" : 290 }
{ "_id" : "9788468738895", "paginas" : 360 }
{ "_id" : "9788466637688", "paginas" : null }
> db.libro.find({paginas:{$exists:false}},{paginas:1})
{ "_id" : "9788415140054" }
```

4.2 Queries específicas por tipo de dato

Como hemos visto en el tema anterior, MongoDB tiene un amplio rango de tipos de dato. Algunos de ellos tienen un comportamiento especial en las queries.

4.2.1 null

null tiene un comportamiento un poco extraño, ya que hace *matching* en los siguientes casos:

- Con valores que almacenan null, por ejemplo "y" : null
- Con campos que no existen en un documento. Si por ejemplo el documentoA no tiene informado el campo x, una búsqueda por { x:null } sí que se encontraría.

Veamos un ejemplo que cubre ambos casos. Para ello inserte un nuevo libro, "Las legiones malditas", que encontrará en los recursos del curso como 4.2.1.null.json. Este nuevo documento tiene *null* en su campo *paginas*, de forma que si buscamos libros con ese criterio, Mongo nos lo devolverá. Pero también nos va a devolver otro libro que no tiene definido el campo páginas ("La princesa de hielo").

```
> db.libro.find({paginas:null},{titulo:1,autor:1,paginas:1}).pretty()
{
     "_id" : "9788415140054",
     "titulo" : "La princesa de hielo",
     "autor" : "Camilla Lackberg"
}
{
     "_id" : "9788466637688",
     "titulo" : "Las legiones malditas",
     "autor" : "Santiago Posteguillo",
     "paginas" : null
}
```

4.2.2 Expresiones regulares

MongoDB incluye el trabajo con expresiones regulares de forma nativa.

Combinadas con la búsqueda, las expresiones regulares son un mecanismo muy potente y flexible para el *matching* de Strings.

Las expresiones regulares en Mongo siguen la misma sintaxis que el lenguaje de programación Perl, que su vez se ajusta, salvo detalles, al resto de lenguajes de programación. Veamos ejemplos para mostrar su potencia.

Recuperamos los libros del autor Posteguillo, sin preocuparnos por mayúsculas y minúsculas, haríamos esta consulta.

```
> db.libro.find({autor:/posteguillo/i},{titulo:1,autor:1}).pretty()
{
     "_id" : "9788408117117",
     "titulo" : "Circo Máximo",
     "autor" : "Santiago Posteguillo"
}
{
     "_id" : "9788466637688",
     "titulo" : "Las legiones malditas",
     "autor" : "Santiago Posteguillo"
}
```

Recuperamos documentos con la expresión "juego de" en el título.

```
> db.libro.find({titulo:/juego de/i},{titulo:1,autor:1}).pretty()
{
     "_id" : "9788401342158",
     "titulo" : "El juego de Ripper",
     "autor" : "Isabel Allende"
}
{
     "_id" : "9788496208919",
     "titulo" : "Juego de tronos: Canción de hielo y fuego 1",
     "autor" : "George R.R. Martin"
}
```

Recuperamos documentos que empiecen por la expresión "juego de".

```
> db.libro.find({titulo:/^juego de/i},{titulo:1,autor:1}).pretty()
{
     "_id" : "9788496208919",
     "titulo" : "Juego de tronos: Canción de hielo y fuego 1",
     "autor" : "George R.R. Martin"
}
```

4.2.3 Arrays

Hacer queries en arrays está diseñado en MongoDB para que resulte muy sencillo. Para ilustrarlo vamos insertar datos en una nueva colección que nos servirá para hacer unos ejemplos más claros.

Inserte primero unas frutas en el array correspondiente.

```
> db.comida.insert({fruta:["manzana","platano","pera"]})
> db.comida.findOne()
```

```
{
     "_id" : ObjectId("539c7c0533d9395ec25a6919"),
     "fruta" : [
          "manzana",
          "platano",
          "pera"
     ]
}
```

Ahora, para buscar documentos que tengan un elemento concreto dentro del array, es tan simple como poner el nombre del array y el valor que queremos encontrar.

```
> db.comida.findOne({fruta:"pera"})
{
     "_id" : ObjectId("539c7c0533d9395ec25a6919"),
     "fruta" : [
          "manzana",
          "platano",
          "pera"
     ]
}
```

Podemos utilizar cualquiera de los operadores aprendidos hasta el momento, como por ejemplo *$in,* para recuperar documentos que tengan alguno de los valores especificados.

```
> db.comida.findOne({fruta: { $in:["pera","platano","melocoton"] }})
{
     "_id" : ObjectId("539c7c0533d9395ec25a6919"),
     "fruta" : [
          "manzana",
          "platano",
          "pera"
     ]
}
```

E incluso el recién aprendido matching por expresiones regulares.

```
> db.comida.findOne({fruta:/plat/})
{
     "_id" : ObjectId("539c7c0533d9395ec25a6919"),
     "fruta" : [
          "manzana",
          "platano",
          "pera"
     ]
}
```

$all

Si necesita recuperar arrays que contengan más de un elemento, puede utilizar el operador *$all*, que permite especificar una lista de elementos.

Por ejemplo, supongamos que tenemos las siguientes listas.

```
> db.comida.drop()
true

> db.comida.insert({_id:1, fruta:["manzana","platano","melocoton"]})
> db.comida.insert({_id:2, fruta:["manzana","kiwi","naranja"]})
> db.comida.insert({_id:3, fruta:["cerezas","platano","manzana"]})

> db.comida.find().pretty()
{ "_id" : 1, "fruta" : [ "manzana", "platano", "melocoton" ] }
{ "_id" : 2, "fruta" : [ "manzana", "kiwi", "naranja" ] }
{ "_id" : 3, "fruta" : [ "cerezas", "platano", "manzana" ] }
```

Si ahora queremos documentos que tengan los elementos *platano* y *manzana*, lo haremos con *$all* de esta forma.

```
> db.comida.find({fruta: { $all:["platano","manzana"]}})
{ "_id" : 1, "fruta" : [ "manzana", "platano", "melocoton" ] }
{ "_id" : 3, "fruta" : [ "cerezas", "platano", "manzana" ] }
```

El orden de los elementos, tanto en los parámetros del find, como en el que estén almacenados, no importa, se recuperarán los documentos de igual forma.

Para recuperar documentos que tengan un valor concreto en cierto índice del array, se consigue poniendo dicho índice (empezando desde cero) tras un punto del array. Es decir:

- fruta.0 es el primer elemento del array.
- fruta.1 es el segundo elemento del array.
- fruta.2 es el tercer elemento del array.
- etc

Con esto, para encontrar los documentos cuyo primer elemento del array "fruta" es "cerezas" lanzaremos el siguiente comando.

```
> db.comida.find({"fruta.0":"cerezas"})
{ "_id" : 3, "fruta" : [ "cerezas", "platano", "manzana" ] }
```

Es importante poner "fruta.0" entrecomillado, en caso contrario fallará.

$size

Un condicional útil para las consultas contra arrays es *$size*, que permite recuperar arrays que tienen un determinado tamaño.

Primero insertemos un elemento más con un tamaño de array fruta diferente.

```
> db.comida.insert({_id:4,
fruta:["cerezas","platano","manzana","melocoton"]})

> db.comida.find()
{ "_id" : 1, "fruta" : [ "manzana", "platano", "melocoton" ] }
```

```
{ "_id" : 2, "fruta" : [ "manzana", "kiwi", "naranja" ] }
{ "_id" : 3, "fruta" : [ "cerezas", "platano", "manzana" ] }
{ "_id" : 4, "fruta" : [ "cerezas", "platano", "manzana", "melocoton"
] }
```

Ahora, consultando por tamaño de array obtenemos diferentes resultados.

```
> db.comida.find({fruta:{$size:4}})
{ "_id" : 4, "fruta":["cerezas", "platano", "manzana", "melocoton" ]}
> db.comida.find({fruta:{$size:3}})
{ "_id" : 1, "fruta" : [ "manzana", "platano", "melocoton" ] }
{ "_id" : 2, "fruta" : [ "manzana", "kiwi", "naranja" ] }
{ "_id" : 3, "fruta" : [ "cerezas", "platano", "manzana" ] }
```

$slice

El operador $slice se utiliza para devolver un subconjunto de los elementos del array. No influye en el criterio de búsqueda, solo es para especificar a mongo lo que queremos que nos devuelva. Su sintaxis es la siguiente.

clave: { $slice: x}

Donde según x:

- x es un entero positivo (p.ej 10), devuelve los primeros x elementos del array clave.
- x es un entero negativo (p.ej -10), devuelve los últimos x elementos del array clave.
- x es un array (p.ej [2,4], devuelve los elementos desde el 2 al 4, ambos incluidos.

Ahora, solo queda probar algunos ejemplos. Fíjese que en los siguientes *find* no estamos especificando ningún criterio, por lo que se recuperan todos los documentos de la colección.

```
> db.comida.find({},{fruta:{$slice:2}})
{ "_id" : 1, "fruta" : [ "manzana", "platano" ] }
{ "_id" : 2, "fruta" : [ "manzana", "kiwi" ] }
{ "_id" : 3, "fruta" : [ "cerezas", "platano" ] }
{ "_id" : 4, "fruta" : [ "cerezas", "platano" ] }
> db.comida.find({},{fruta:{$slice:-2}})
{ "_id" : 1, "fruta" : [ "platano", "melocoton" ] }
{ "_id" : 2, "fruta" : [ "kiwi", "naranja" ] }
{ "_id" : 3, "fruta" : [ "platano", "manzana" ] }
{ "_id" : 4, "fruta" : [ "manzana", "melocoton" ] }
> db.comida.find({},{fruta:{$slice:[1,3]}})
{ "_id" : 1, "fruta" : [ "platano", "melocoton" ] }
{ "_id" : 2, "fruta" : [ "kiwi", "naranja" ] }
{ "_id" : 3, "fruta" : [ "platano", "manzana" ] }
{ "_id" : 4, "fruta" : [ "platano", "manzana", "melocoton" ] }
```

4.2.4 Documentos embebidos

Para hacer queries contra campos de documentos embebidos dentro de otros documentos, solamente hay que poner la "ruta" completa de claves separada por puntos. Es decir, si tenemos por ejemplo una estructura como esta para los libros.

```
db.libro.save({
     "_id":"9788408117117",
     "titulo":"Circo Máximo",
     "autor": {
          nombre:"Santiago",
          apellidos:"Posteguillo Gómez",
          nacimiento: {
               anyo:1967,
               ciudad:"Valencia"
          }
     },
     "editorial":"Planeta",
     "enstock":true,
     "paginas":1100,
     "precio":21.75
})
```

Podemos lanzar *queries* directamente contra los documentos embebidos, autor y nacimiento de esta forma. Para re

```
> db.libro.findOne({"autor.nombre":"Santiago"})
```

El *find* anterior recuperará el primer documento donde el campo nombre, del campo autor sea "Santiago".

```
> db.libro.findOne({"autor.nacimiento.anyo":{$gt:1965}})
```

El *find* anterior recuperará el primer documento donde el campo anyo, del campo nacimiento, del campo autor sea mayor que 1965.

4.3 Cursores

La base de datos devuelve resultados para *find* utilizando cursores, que realmente es un puntero a los resultados de una query, no los resultados en si mismos. Los clientes integrados con Mongo, iteran sobre los cursores para recuperar los resultados, y ofrecen un conjunto de funcionalidades como limitar el número de resultados, saltar para no tener que recuperar obligatoriamente los primeros, ordenarlos, etc.

Para probar los cursores, vamos a insertar algunos datos de prueba para poder consultarlos. Recuerdo que el shell de Mongo es un shell completo en JavaScript, por lo que podemos utilizar este lenguaje de programación para hacer un bucle desde cero hasta 99 e insertar cada número en una colección de prueba.

```
for(i=0; i<100; i++) {
     db.testing.insert({x : i});
}
```

Si ahora ejecuta en el shell un *find* para recuperar todos los documentos, verá que no obtiene directamente los 100 documentos, sino que el shell solo le devuelve los primeros 20, y al final un mensaje que le dice que escriba "it" para obtener más.

```
> db.testing.find()
{ "_id" : ObjectId("539c93db33d9395ec25a691a"), "x" : 0 }
{ "_id" : ObjectId("539c93db33d9395ec25a691b"), "x" : 1 }
{ "_id" : ObjectId("539c93db33d9395ec25a691c"), "x" : 2 }
{ "_id" : ObjectId("539c93db33d9395ec25a691d"), "x" : 3 }
{ "_id" : ObjectId("539c93db33d9395ec25a691e"), "x" : 4 }
{ "_id" : ObjectId("539c93db33d9395ec25a691f"), "x" : 5 }
{ "_id" : ObjectId("539c93db33d9395ec25a6920"), "x" : 6 }
{ "_id" : ObjectId("539c93db33d9395ec25a6921"), "x" : 7 }
{ "_id" : ObjectId("539c93db33d9395ec25a6922"), "x" : 8 }
{ "_id" : ObjectId("539c93db33d9395ec25a6923"), "x" : 9 }
{ "_id" : ObjectId("539c93db33d9395ec25a6924"), "x" : 10 }
{ "_id" : ObjectId("539c93db33d9395ec25a6925"), "x" : 11 }
{ "_id" : ObjectId("539c93db33d9395ec25a6926"), "x" : 12 }
{ "_id" : ObjectId("539c93db33d9395ec25a6927"), "x" : 13 }
{ "_id" : ObjectId("539c93db33d9395ec25a6928"), "x" : 14 }
{ "_id" : ObjectId("539c93db33d9395ec25a6929"), "x" : 15 }
{ "_id" : ObjectId("539c93db33d9395ec25a692a"), "x" : 16 }
{ "_id" : ObjectId("539c93db33d9395ec25a692b"), "x" : 17 }
{ "_id" : ObjectId("539c93db33d9395ec25a692c"), "x" : 18 }
{ "_id" : ObjectId("539c93db33d9395ec25a692d"), "x" : 19 }
Type "it" for more
```

Si escribe "it" y pulsa enter, obtendrá los 20 siguientes, y seguirá diciéndole que escriba "it" para más mientras no haya llegado a los 100 elementos devueltos.

```
> it
{ "_id" : ObjectId("539c93db33d9395ec25a692e"), "x" : 20 }
{ "_id" : ObjectId("539c93db33d9395ec25a692f"), "x" : 21 }
{ "_id" : ObjectId("539c93db33d9395ec25a6930"), "x" : 22 }
{ "_id" : ObjectId("539c93db33d9395ec25a6931"), "x" : 23 }
{ "_id" : ObjectId("539c93db33d9395ec25a6932"), "x" : 24 }
{ "_id" : ObjectId("539c93db33d9395ec25a6933"), "x" : 25 }
{ "_id" : ObjectId("539c93db33d9395ec25a6934"), "x" : 26 }
{ "_id" : ObjectId("539c93db33d9395ec25a6935"), "x" : 27 }
{ "_id" : ObjectId("539c93db33d9395ec25a6936"), "x" : 28 }
{ "_id" : ObjectId("539c93db33d9395ec25a6937"), "x" : 29 }
{ "_id" : ObjectId("539c93db33d9395ec25a6938"), "x" : 30 }
{ "_id" : ObjectId("539c93db33d9395ec25a6939"), "x" : 31 }
{ "_id" : ObjectId("539c93db33d9395ec25a693a"), "x" : 32 }
{ "_id" : ObjectId("539c93db33d9395ec25a693b"), "x" : 33 }
{ "_id" : ObjectId("539c93db33d9395ec25a693c"), "x" : 34 }
{ "_id" : ObjectId("539c93db33d9395ec25a693d"), "x" : 35 }
{ "_id" : ObjectId("539c93db33d9395ec25a693e"), "x" : 36 }
{ "_id" : ObjectId("539c93db33d9395ec25a693f"), "x" : 37 }
{ "_id" : ObjectId("539c93db33d9395ec25a6940"), "x" : 38 }
{ "_id" : ObjectId("539c93db33d9395ec25a6941"), "x" : 39 }
Type "it" for more
```

4.3.1 Limits, Skips y Sorts

Las opciones de query más comunes son limitar el número de resultados, saltar un número de resultados, y ordenarlos. Todas estas opciones deben especificarse en la propia query al sistema.

Para poder comparar con las siguientes consultas, vea como devuelve Mongo por defecto todos los libros que tenemos.

```
> db.libro.find({},{titulo:1})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo" }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de
hielo y fuego 1" }
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros" }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
{ "_id" : "9788466637688", "titulo" : "Las legiones malditas" }
```

Para fijar un límite, debe concatenar la función *limit* tras la función *find*. Por ejemplo, para solo recuperar los primeros tres libros almacenados, use esto:

```
> db.libro.find({},{titulo:1}).limit(3)
{ "_id" : "9788408117117", "titulo" : "Circo Máximo" }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper" }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de
hielo y fuego 1" }
```

Si hubiera menos documentos que cumplieran el criterio de búsqueda que el número del limit, solo se devolvería ese número de documentos menor que hicieron *matching*. Por ejemplo, si buscamos libros de la editorial "Planeta", solo nos devuelve los dos documentos que lo cumplen.

```
> db.libro.find({editorial:"Planeta"},{titulo:1}).limit(3)
{ "_id" : "9788408117117", "titulo" : "Circo Máximo" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio" }
```

La función *skip* para "saltar" algunos resultados, funciona de forma similar a *limit*, concatenándola tras find. Con el siguiente skip(3) saltamos los primeros tres títulos, y el cursor empezará a devolver a partir del cuarto.

```
> db.libro.find({},{titulo:1}).skip(3)
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros" }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo" }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio" }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego" }
{ "_id" : "9788466637688", "titulo" : "Las legiones malditas" }
```

Si hubiera menos documentos que los "saltados", entonces no se devolvería ningún resultado.

La función *sort* sirve para ordenar los resultados que se devuelven. Recibe un objeto json con las claves y un valor 1 (para ordenación ascendente) o -1 (para ordenación descendente). Si se pasan varias claves, el orden se irá aplicando "de izquierda a derecha", como en el order by de una sentencia *select*.

Por ejemplo, en la siguiente consulta vamos a recuperar los libros ordenados, primero por precio ascendentemente (precio:1), y en caso de empate por número de páginas descendentemente (paginas:-1).

```
> db.libro.find({},{titulo:1,precio:1,paginas:1}).sort({precio:1,
paginas:-1})
{ "_id" : "9788499088075", "titulo" : "La ladrona de libros",
"paginas" : 544, "precio" : 9.45 }
{ "_id" : "9788496208919", "titulo" : "Juego de tronos: Canción de
hielo y fuego 1", "paginas" : 793, "precio" : 9.5 }
{ "_id" : "9788415140054", "titulo" : "La princesa de hielo",
"precio" : 11 }
{ "_id" : "9788468738895", "titulo" : "Las reglas del juego",
"paginas" : 360, "precio" : 15.9 }
{ "_id" : "9788408113331", "titulo" : "Las carreras de Escorpio",
"paginas" : 290, "precio" : 17.23 }
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "paginas" :
1100, "precio" : 21.75 }
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "paginas"
: 480, "precio" : 21.75 }
{ "_id" : "9788466637688", "titulo" : "Las legiones malditas",
"paginas" : null, "precio" : 22 }
```

Estas tres funciones, *limit, skip* y *sort*, pueden combinarse como se quiera para recuperar exactamente lo que queramos. De esta forma, en la siguiente consulta recuperamos tres documentos (limit(3)) desde el tercer resultado (skip(2)) tras ordenar los documentos por precio ascendentemente y paginas descendentemente.

```
>
db.libro.find({},{titulo:1,precio:1,paginas:1}).limit(3).skip(2).sort
({precio:1, paginas:-1}).pretty()
{
     "_id" : "9788415140054",
     "titulo" : "La princesa de hielo",
     "precio" : 11
}
{
     "_id" : "9788468738895",
     "titulo" : "Las reglas del juego",
     "paginas" : 360,
     "precio" : 15.9
}
{
     "_id" : "9788408113331",
     "titulo" : "Las carreras de Escorpio",
     "paginas" : 290,
     "precio" : 17.23
}
```

4.4 Agregación.

Una vez tenemos datos almacenados en MongoDB, queremos hacer más que solamente recuperarlos, queremos analizarlos de diversas formas interesantes. Este tema introduce las herramientas de agregación que Mongo ofrece.

El entorno de agregación nos permite transformar y combinar documentos en una colección. Básicamente, se construye un *pipeline* (secuencia de comandos) que procesan un conjunto de documentos a través de varios "bloques": filtrado, proyecciones, agrupaciones, ordenación, limitación y *skipping* (saltar resultados).

Por ejemplo, si de la colección de libros de ejemplo quisiéramos saber los tres autores con más libros escritos, crearíamos un *pipeline* con los siguientes pasos:

- Proyectaríamos el autor de cada libro.
- Agruparíamos los autores por nombre, contando el número de ocurrencias.
- Ordenaríamos los autores por dicho número de ocurrencias, descendentemente.
- Limitaríamos el resultado a los primeros tres.

Cada uno de estos pasos se corresponde con un operador de agregación.

1. {"$project" : { "autor" : 1 }}

   Esto "proyecta/extrae" el autor de cada documento. La sintaxis es similar al selector de campos utilizado con *find* (el segundo parámetro), especificando "nombrecampo": 1 para incluir o "nombrecampo":0 para excluir.

   Después de esta operación, cada documento de los resultados es algo así: {"_id":*id*, "autor":*"nombre de autor"*}. Estos resultados solo existen en memoria, no se escribirán nunca a disco.

2. {"$group" : { "_id":"$autor", "count": {"$sum":1}}}

   Eso agrupa por autores e incrementa "count" para cada documento en los que el autor aparece.

   Primero especificamos el campo por el que queremos agrupar, que es "autor". Esto lo indicamos con el "_id":"$autor". Podríamos entenderlo como: tras el *Group* habrá un documento resultado para cada autor, por lo que "autor" se convertirá en el identificador único ("_id").

   El segundo campo significa añadir 1 al campo "count" para cada documento en el grupo. Fíjese que los documentos de entrada (libro) no tienen el campo "count", es un nuevo campo creado por el "$group".

Al final de este paso, cada documento resultado es algo como: {"_id":"*nombre de autor*", "count": *numeroDeLibros*}.

3. {"$sort" : {"count" : -1}}

   Esto reordena los resultados por el valor del campo "count" descendentemente (-1). En caso de ascendente sería "count":1.

4. {"$limit" : 3}

   Limita el número de resultados a los primeros tres.

Para hacer la primera prueba, vamos a rellenar de nuevo la colección libro. Primero elimine la colección y guarde los libros que tiene en los recursos del curso, en el fichero *5.1.libros_ejemplo.json.*

```
> db.libro.drop()
```

Cargue 5.1.libros_ejemplo.json y compruebe que se han insertado los 10 libros de ejemplo.

```
> db.libro.count()
```

```
10
```

Finalmente, solo nos queda ejecutar el comando *agregate* pasando cada uno de estos pasos como parámetro.

```
> db.libro.aggregate({"$project" : {"autor" : 1}}, {"$group" : {"_id"
: "$autor", "count" : {"$sum" : 1}}}, {"$sort" : {"count" : -1}},
{"$limit" : 3})
{ "_id" : "Santiago Posteguillo", "count" : 3 }
{ "_id" : "George R.R. Martin", "count" : 2 }
{ "_id" : "Anna Casanovas", "count" : 1 }
```

Para depurar los diferentes pasos del agregate, puede ir invocándolo poco a poco, es decir, empezar por ejemplo solamente con el $project e ir añadiendo los demás en vista de los resultados.

```
> db.libro.aggregate({"$project" : {"autor" : 1}})
{ "_id" : "9788408117117", "autor" : "Santiago Posteguillo" }
{ "_id" : "9788401342158", "autor" : "Isabel Allende" }
{ "_id" : "9788496208919", "autor" : "George R.R. Martin" }
{ "_id" : "9788499088075", "autor" : "Markus Zusak" }
{ "_id" : "9788415140054", "autor" : "Camilla Lackberg" }
{ "_id" : "9788408113331", "autor" : "Maggie Stiefvater" }
{ "_id" : "9788468738895", "autor" : "Anna Casanovas" }
{ "_id" : "9788408118329", "autor" : "Santiago Posteguillo" }
{ "_id" : "9788466637688", "autor" : "Santiago Posteguillo" }

{ "_id" : "9788496208681", "autor" : "George R.R. Martin" }
```

Ahora, añadimos el $project y el $group.

```
> db.libro.aggregate({"$project" : {"autor" : 1}}, {"$group" : {"_id"
: "$autor", "count" : {"$sum" : 1}}})
{ "_id" : "Anna Casanovas", "count" : 1 }
{ "_id" : "Maggie Stiefvater", "count" : 1 }
{ "_id" : "Camilla Lackberg", "count" : 1 }
{ "_id" : "George R.R. Martin", "count" : 2 }
{ "_id" : "Markus Zusak", "count" : 1 }
{ "_id" : "Isabel Allende", "count" : 1 }
{ "_id" : "Santiago Posteguillo", "count" : 3 }
```

Y así podría continuar depurando hasta el paso que requiera.

Operaciones *Pipeline*

Cada operador recibe un conjunto de documentos, hace algún tipo de transformación sobre ellos, y después pasa el resultado de la transformación. Esto es lo que se conoce como *pipeline*, el paso de datos de un operador a otro donde el output de uno es el input de otro. El último operador del *pipeline* acabará dando los resultados al cliente, conformando el resultado final de toda la secuencia.

Los operadores pueden combinarse en cualquier orden y tantas veces como se quiera.

4.4.1 Operador $match

El operador $match filtra documentos de forma que podamos ejecutar una agregación sobre un subconjunto de documentos.. Por ejemplo, si quisiéramos calcular estadísticas (agregación) para libros solo de una editorial, deberíamos añadir una expresión "$match" como {$match: {"editorial":"Planeta"}}.

$match puede utilizar todos los operadores para query estudiados (*$gt, $lt, $in,* etc).

Una buena práctica es poner las expresiones *$match* tan pronto como se pueda en el *pipeline*. Los beneficios son dos: primero filtramos documentos que no necesitamos pronto, evitando procesamientos innecesarios en el *pipeline*, y segundo que *$match* utilizará índices contra las colecciones, si existen, cosa que no hará en medio del *pipeline* ya que los documentos están en memoria, y por tanto sin índices.

Con el siguiente ejemplo, filtraríamos solo aquellos libros con un precio mayor que 20, reduciendo el conjunto inicial de documentos y formando la entrada de los siguientes pasos del *pipeline.*

```
> db.libro.aggregate({$match:{precio:{$gt:20}}})
{ "_id" : "9788408117117", "titulo" : "Circo Máximo", "autor" :
"Santiago Posteguillo", "editorial" : "Planeta", "enstock" : true,
"paginas" : 1100, "precio" : 21.75, "fecha" : ISODate("2013-08-
29T00:00:00Z") }
```

```
{ "_id" : "9788401342158", "titulo" : "El juego de Ripper", "autor" :
"Isabel Allende", "editorial" : "Plaza & Janes", "enstock" : true,
"paginas" : 480, "precio" : 21.75, "fecha" : ISODate("2014-03-
01T00:00:00Z") }
{ "_id" : "9788466637688", "titulo" : "Las legiones malditas",
"autor" : "Santiago Posteguillo", "editorial" : "Planeta", "enstock"
: true, "paginas" : 860, "precio" : 22, "fecha" : ISODate("2008-11-
19T00:00:00Z") }
```

Como hemos comentado, cualquiera de las condiciones estudiadas del find podemos utilizarlas con match. Como por ejemplo, los libros en stock.

```
> db.libro.aggregate({$match:{enstock:true}})
```

O utilizando expresiones regulares y recuperando los libros con la palabra juego o la palabra ladron en el titulo.

```
> db.libro.aggregate({$match:{titulo:/(juego|ladron)/i}})
```

4.4.2 Operador $project

La "proyección" es mucho más potente en el *pipeline* que en el lenguaje "normal" de la query. $project nos permite extraer campos de subdocumentos, renombrar campos, y realizar operaciones interesantes sobre ellos.

La operación más sencilla de $project realiza una selección simple de unos documentos de entrada, es decir, de los documentos de entrada con campos *a, b, c* y *d*, con $project decimos que nos quedamos únicamente con *a* y *c.*

Para incluir o excluir un campo, se utiliza la misma sintaxis que para el segundo argumento de una query, es decir "nombrecampo": 1 para incluir o "nombrecampo":0 para excluir.

Por defecto, el campo "_id" siempre se devuelve si existe en los documentos de entrada. Lo podemos eliminar por ejemplo utilizando $project de la siguiente forma, en la que devolveríamos todos los campos excepto "_id".

{"$project" : { "_id" : 0 }}

Con $project también podemos renombrar el campo proyectado. Por ejemplo si queremos devolver el campo "_id" de los libros como "isbn" haríamos esto.

```
> db.libro.aggregate({"$project" : {"isbn" : "$_id"}})
{ "_id" : "9788408117117", "isbn" : "9788408117117" }
{ "_id" : "9788401342158", "isbn" : "9788401342158" }
{ "_id" : "9788496208919", "isbn" : "9788496208919" }
{ "_id" : "9788499088075", "isbn" : "9788499088075" }
{ "_id" : "9788415140054", "isbn" : "9788415140054" }
{ "_id" : "9788408113331", "isbn" : "9788408113331" }
{ "_id" : "9788468738895", "isbn" : "9788468738895" }
{ "_id" : "9788408118329", "isbn" : "9788408118329" }
{ "_id" : "9788466637688", "isbn" : "9788466637688" }
{ "_id" : "9788496208681", "isbn" : "9788496208681" }
```

La clave está en $_id, cuando ponemos el símbolo del dólar ($) más el nombre de un campo, nos estamos refiriendo al valor que tomará dicho campo para cada documento en el entorno de agregación. Por tanto con *$project: {"isbn":"$_id"}* estamos diciendo que proyectamos un nuevo campo isbn como el valor que tendrá el campo _id en cada documento de entrada. Igualmente podríamos haber renombrado cualquier otro campo, por ejemplo el campo "enstock" como "disponible".

```
> db.libro.aggregate({"$project" : {"isbn" : "$_id",
"disponible":"$enstock"}})
{ "_id" : "9788408117117", "isbn" : "9788408117117", "disponible" : true }
{ "_id" : "9788401342158", "isbn" : "9788401342158", "disponible" : true }
{ "_id" : "9788496208919", "isbn" : "9788496208919", "disponible" : true }
{ "_id" : "9788499088075", "isbn" : "9788499088075", "disponible" : false }
{ "_id" : "9788415140054", "isbn" : "9788415140054", "disponible" : true }
{ "_id" : "9788408113331", "isbn" : "9788408113331", "disponible" : false }
{ "_id" : "9788468738895", "isbn" : "9788468738895", "disponible" : false }
{ "_id" : "9788408118329", "isbn" : "9788408118329", "disponible" : true }
{ "_id" : "9788466637688", "isbn" : "9788466637688", "disponible" : true }
{ "_id" : "9788496208681", "isbn" : "9788496208681", "disponible" : true }
```

Fíjese que en el ejemplo anterior, el campo _id se sigue devolviendo igualmente, eso es porque como hemos dicho, por defecto, siempre se devuelve _id. Para eliminarlo solo tenemos que añadir un elemento más a $project, diciendo que queremos excluirlo.

```
> db.libro.aggregate({"$project" : {"isbn" : "$_id", "disponible":
"$enstock", "_id":0}})
```

Es importante que comprenda que $project recibe un conjunto de "operaciones" a aplicar, separadas por coma, y que podemos especificar todas las que queramos.

```
> db.libro.aggregate({"$project" : {"isbn" : "$_id", "disponible":
"$enstock", "_id":0, "autor":1}})
```

4.4.2.1 Expresiones en el Pipeline

La operaciones más simples en $project son inclusión, exclusión y nombres de campo (*"$nombrecampo"*). Aunque hay otras más, mucho más potentes. Podemos utilizar expresiones, que nos permiten combinar múltiples literales y variables en un valor único.

Expresiones matemáticas. Las expresiones matemáticas nos permiten manipular valores numéricos contra los que operar, y que normalmente se especifican en un array.

Por ejemplo, supongamos que tenemos una colección de *empleados* con dos campos, *salario* y *bonus*. Si queremos calcular la suma de ambos en una variable *total_a_pagar* tenemos que utilizar la operación de suma *$add*, que recibe dos parámetros con los campos o constantes a sumar. Recuerde que cuando ponemos *"$nombredecampo"* se substituirá en la operación por el valor del campo *"nombredecampo".*

```
> db.empleados.insert({name:"Pedro", salario:24000, bonus:2500})
> db.empleados.insert({name:"Laura", salario:27000, bonus:300})
> db.empleados.aggregate({
"$project": { "total_a_pagar":{ "$add":["$salario","$bonus"]}}})

{ "_id" : ObjectId("53a7289a63fe582e01dc37ab"), "total_a_pagar" : 26500 }
{ "_id" : ObjectId("53a728a863fe582e01dc37ac"), "total_a_pagar" : 27300 }
```

Las operaciones se pueden anidar como queramos y conseguir las operaciones tan complejas como necesitemos. Supongamos que ahora queremos calcular el salario mensual, para lo queremos dividir la suma anterior por doce.

```
> db.empleados.aggregate({
"$project": { "total_a_pagar_mensual": { "$divide":
     [{ "$add":["$salario","$bonus"]},12]}}})

{ "_id" : ObjectId("53a7289a63fe582e01dc37ab"), "total_a_pagar_mensual" :
2208.3333333333335 }
{ "_id" : ObjectId("53a728a863fe582e01dc37ac"), "total_a_pagar_mensual" :
2275 }
```

Esta es la sintaxis para cada operador matemático.

“$add”: [expr1, expr2, … , exprN]. Toma uno o mas expresiones y las suma.

“$substract”: [expr1, expr2]. Resta la expr2 a la expr1.

“$multiply”: [expr1, expr2, ..., expón]. Toma una o más expresiones y las multiplica.

“$divide”: [expr1, expr2]. Divide la expr1 entre la expr2.

“$mod”: [expr1, expr2]. Di vide la expr1 entre la expr2 y devuelve el resto.

Expresiones de fechas. Muchas agregaciones son basadas en el tiempo. ¿Qué ocurrió la semana pasada? ¿Y el mes pasado?. Es por esto que el entorno de agregación tiene un conjunto de expresiones que pueden utilizarse para extraer información sobre fechas de una forma muy usable, son: “$year”, “$month”, “$week”, “$dayOfMonth”, “$dayOfWeek”, “$dayOfYear”, “$hour”, “$minute” y “$second”.

Como es obvio, solo se pueden utilizar este tipo de expresiones sobre campos guardados con el tipo Date.

Cada una de estas expresiones son básicamente lo mismo, toman la fecha, la expresión y devuelven un número. Con siguiente sentencia de agregación vamos a recuperar tanto la fecha, como el año de dicha fecha, gracias a la expresión “$year”.

```
> db.libro.aggregate({"$project": { "fecha":1, "year":
{"$year":"$fecha" }}})
{"_id":"9788408117117","fecha":ISODate("2013-08-29T00:00:00Z"),"year":2013}
{"_id":"9788401342158","fecha":ISODate("2014-03-01T00:00:00Z"),"year":2014}
{"_id":"9788496208919","fecha":ISODate("2011-11-24T00:00:00Z"),"year":2011}
```

```
{"_id":"9788499088075","fecha":ISODate("2009-01-09T00:00:00Z"),"year":2009}
{"_id":"9788415140054","fecha":ISODate("2012-10-30T00:00:00Z"),"year":2012}
{"_id":"9788408113331","fecha":ISODate("2013-06-04T00:00:00Z"),"year":2013}
{"_id":"9788468738895","fecha":ISODate("2014-02-06T00:00:00Z"),"year":2014}
{"_id":"9788408118329","fecha":ISODate("2012-05-19T00:00:00Z"),"year":2012}
{"_id":"9788466637688","fecha":ISODate("2008-11-19T00:00:00Z"),"year":2008}
{"_id":"9788496208681","fecha":ISODate("2009-01-24T00:00:00Z"),"year":2009}
```

Hemos generado un nuevo campo proyectado "year" aplicando la expresión de "$year" contra el campo *fecha*. De forma muy parecida podemos sacar el mes.

```
> db.libro.aggregate({"$project": { "fecha":1, "mes":
{"$month":"$fecha" }}})
{"_id":"9788408117117","fecha":ISODate("2013-08-29T00:00:00Z"),"mes":8}
{"_id":"9788401342158","fecha":ISODate("2014-03-01T00:00:00Z"),"mes":3}
{"_id":"9788496208919","fecha":ISODate("2011-11-24T00:00:00Z"),"mes":11}
{"_id":"9788499088075","fecha":ISODate("2009-01-09T00:00:00Z"),"mes":1}
{"_id":"9788415140054","fecha":ISODate("2012-10-30T00:00:00Z"),"mes":10}
{"_id":"9788408113331","fecha":ISODate("2013-06-04T00:00:00Z"),"mes":6}
{"_id":"9788468738895","fecha":ISODate("2014-02-06T00:00:00Z"),"mes":2}
{"_id":"9788408118329","fecha":ISODate("2012-05-19T00:00:00Z"),"mes":5}
{"_id":"9788466637688","fecha":ISODate("2008-11-19T00:00:00Z"),"mes":11}
{"_id":"9788496208681","fecha":ISODate("2009-01-24T00:00:00Z"),"mes":1}
```

Expresiones de Strings. Hay una pocas operaciones básicas sobre cadenas de caracteres que también podemos utilizar. Son estas:

"$substr": [expr, start, length]. Devuelve el substring de la expresión *expr*, empezando en la posición *start* de la cadena y devolviendo desde ahí un total de *length* caracteres.

En el siguiente ejemplo proyectamos un campo *inicio* los primeros quince caracteres del campo título.

```
> db.libro.aggregate({"$project": {inicio:{"$substr":
["$titulo",0,15] }}})
{ "_id" : "9788408117117", "inicio" : "Circo Máximo" }
{ "_id" : "9788401342158", "inicio" : "El juego de Rip" }
{ "_id" : "9788496208919", "inicio" : "Juego de tronos" }
{ "_id" : "9788499088075", "inicio" : "La ladrona de l" }
{ "_id" : "9788415140054", "inicio" : "La princesa de " }
{ "_id" : "9788408113331", "inicio" : "Las carreras de" }
{ "_id" : "9788468738895", "inicio" : "Las reglas del " }
{ "_id" : "9788408118329", "inicio" : "Los asesinos de" }
{ "_id" : "9788466637688", "inicio" : "Las legiones ma" }
{ "_id" : "9788496208681", "inicio" : "Choque de reyes"
```

"$concat": [expr1, expr2, ..., exprN]. Devuelve un nuevo string resultado de la concatenación de expr1, expr2 y así hasta exprN.

En el siguiente ejemplo proyectamos un campo *descripción* en el que concatenamos el *titulo*, un guión, y el *autor.*

```
>
```

```
db.libro.aggregate({"$project":{"descripcion":{"$concat":["$titulo","
- ","$autor"]},"_id":0}})
{"descripcion" : "Circo Máximo - Santiago Posteguillo" }
{"descripcion" : "El juego de Ripper - Isabel Allende" }
{"descripcion" : "Juego de tronos: Canción de hielo y fuego 1 - George R.R.
Martin" }
{"descripcion" : "La ladrona de libros - Markus Zusak" }
{"descripcion" : "La princesa de hielo - Camilla Lackberg" }
{"descripcion" : "Las carreras de Escorpio - Maggie Stiefvater" }
{"descripcion" : "Las reglas del juego - Anna Casanovas" }
{"descripcion" : "Los asesinos del emperador - Santiago Posteguillo"}
{"descripcion" : "Las legiones malditas - Santiago Posteguillo" }
{"descripcion" : "Choque de reyes: Canción de hielo y fuego 2 - George R.R.
Martin" }
```

"$toLower": expr. Devuelve un string en minúsculas.

"$toUpper": expr. Devuelve un string en mayúsculas.

Expresiones lógicas. Existen algunas operaciones lógicas que podemos utilizar.

"$cmp": [expr1, expr2]. Compara las dos expresiones y devuelve 0 si son iguales, un número negativo si *expr1* es menor que *expr2*, y un número positivo si *expr1* es mayor que *expr2.*

En el siguiente ejemplo proyectamos un campo barato, que tendrá valor -1 si el precio es mayor que 15, y valor 1 si es menor que 15.

```
>
db.libro.aggregate({"$project":{"barato":{"$cmp":[15,"$precio"]},"pre
cio":1} })
{ "_id" : "9788408117117", "precio" : 21.75, "barato" : -1 }
{ "_id" : "9788401342158", "precio" : 21.75, "barato" : -1 }
{ "_id" : "9788496208919", "precio" : 9.5, "barato" : 1 }
{ "_id" : "9788499088075", "precio" : 9.45, "barato" : 1 }
{ "_id" : "9788415140054", "precio" : 11, "barato" : 1 }
{ "_id" : "9788408113331", "precio" : 17.23, "barato" : -1 }
{ "_id" : "9788468738895", "precio" : 15.9, "barato" : -1 }
{ "_id" : "9788408118329", "precio" : 12.95, "barato" : 1 }
{ "_id" : "9788466637688", "precio" : 22, "barato" : -1 }
{ "_id" : "9788496208681", "precio" : 12, "barato" : 1 }
```

**"$strcasecmp"**: [string1, string2]. Compara dos strings de forma insensible a mayúsculas y minúsculas. El valor devuelto funciona como en el caso de "$cmp".

"$eq"/"$ne"/"$gt"/"$gte"/"$lt"/"$lte" : [expr1, expr2]. Realiza la comparación entre *expr1* y *expr2,* devolviendo *true/false* dependiendo de si se cumple o no.

Replanteamos el ejemplo anterior de la proyección del campo "barato" utilizando uno de estos comparadores.

```
>
db.libro.aggregate({"$project":{"barato":{"$gt":[15,"$precio"]},"prec
io":1} })
```

```
{ "_id" : "9788408117117", "precio" : 21.75, "barato" : false }
{ "_id" : "9788401342158", "precio" : 21.75, "barato" : false }
{ "_id" : "9788496208919", "precio" : 9.5, "barato" : true }
{ "_id" : "9788499088075", "precio" : 9.45, "barato" : true }
{ "_id" : "9788415140054", "precio" : 11, "barato" : true }
{ "_id" : "9788408113331", "precio" : 17.23, "barato" : false }
{ "_id" : "9788468738895", "precio" : 15.9, "barato" : false }
{ "_id" : "9788408118329", "precio" : 12.95, "barato" : true }
{ "_id" : "9788466637688", "precio" : 22, "barato" : false }
{ "_id" : "9788496208681", "precio" : 12, "barato" : true }
```

4.4.3 Operador $group

La agrupación nos permite agrupar documentos basándonos en ciertos campos y combinar sus valores. Quizás lo comprenda mejor con algunos ejemplos.

- Si tenemos medidas de temperatura por minuto, y queremos calcular la media por día, deberíamos agrupar las medidas por el valor de su "dia".
- Si tenemos una colección de estudiantes y queremos organizarlos en grupos basándonos en su nota, debemos agrupar por su campo "nota".
- Si tenemos una colección de usuarios y queremos saber cuantos usuarios tenemos por ciudad, podemos agrupar por los campos "provincia" y "ciudad", creando un grupo por dicho par.

Cuando elegimos uno o varios campos por los que agrupar, se lo pasamos a la función "$group" como el campo "_id" del grupo. De esta forma, para los ejemplos anteriores tendríamos respectivamente.

- {"$group" : { "_id":"$dia" }}
- {"$group" : { "_id":"$nota" }}
- {"$group" : { "_id": {"provincia":"$provincia", "ciudad":"$ciudad" }}}

El resultado de "$group", será un documento por cada grupo. Por ejemplo, en el caso de la agrupación por nota el resultado seria ago como:

```
{ "_id" : "Sobresaliente" }
{ "_id" : "Notable" }
{ "_id" : "Bien" }
{ "_id" : "Suficiente" }
{ "_id" : "Insuficiente" }
```

Pero claro, solamente con los grupos no podemos hacer mucho. Lo que en realidad queremos es poder calcular valores a nivel de grupo, y de los documentos incluidos en cada grupo. Aquí es donde entran en juego los *operadores de agrupación.*

Operadores de agrupación

Estos operadores de agrupación nos permiten computar resultados para cada grupo. Al inicio de este tema, en el punto *5.1. Entorno de agregación* vimos un

ejemplo en el que usábamos "$sum" para añadir 1 al total para cada documento del grupo. A continuación veremos este y otro tipo de agrupadores.

"$sum": value. Suma *value* para cada documento y devuelve el resultado. Fíjese que aunque en el ejemplo inicial usaba un literal (1), pueden ser valores más complejos. Por ejemplo con el siguiente comando sumamos el total de los precios de los libros de cada autor.

```
> db.libro.aggregate({"$group":{"_id":"$autor",
"total_precios":{"$sum":"$precio"}}})
{ "_id" : "Anna Casanovas", "total_precios" : 15.9 }
{ "_id" : "Maggie Stiefvater", "total_precios" : 17.23 }
{ "_id" : "Camilla Lackberg", "total_precios" : 11 }
{ "_id" : "George R.R. Martin", "total_precios" : 21.5 }
{ "_id" : "Markus Zusak", "total_precios" : 9.45 }
{ "_id" : "Isabel Allende", "total_precios" : 21.75 }
{ "_id" : "Santiago Posteguillo", "total_precios" : 56.7 }
```

"$avg": value. Devuelve la media aritmética del conjunto de valores del grupo. El ejemplo anterior en el que calculábamos el total, podemos modificarlo para calcular la media por autor.

```
> db.libro.aggregate({"$group":{"_id":"$autor",
"media":{"$avg":"$precio"}}})
{ "_id" : "Anna Casanovas", "media" : 15.9 }
{ "_id" : "Maggie Stiefvater", "media" : 17.23 }
{ "_id" : "Camilla Lackberg", "media" : 11 }
{ "_id" : "George R.R. Martin", "media" : 10.75 }
{ "_id" : "Markus Zusak", "media" : 9.45 }
{ "_id" : "Isabel Allende", "media" : 21.75 }
{ "_id" : "Santiago Posteguillo", "media" : 18.900000000000002 }
```

"$max": value. Devuelve el valor más alto del grupo. Por ejemplo calculamos el libro más largo, es decir, con más páginas, para cada editorial.

```
> db.libro.aggregate({"$group":{"_id":"$editorial",
"mas_largo":{"$max":"$paginas"}}})
{ "_id" : "Debolsillo", "mas_largo" : 544 }
{ "_id" : null, "mas_largo" : 360 }
{ "_id" : "Gigamesh", "mas_largo" : 928 }
{ "_id" : "Embolsillo", "mas_largo" : null }
{ "_id" : "Plaza & Janes", "mas_largo" : 480 }
{ "_id" : "Planeta", "mas_largo" : 1137 }
```

"$min": value. Devuelve el valor más bajo del grupo.

"$first": expr. Devuelve el primer valor que se observe en el grupo, ignorando los siguientes valores. Este operador solo es útil cuando conocemos el orden en el que los datos van a ser procesados, es decir, después de hacer un *sort.*

**"$last"**: expr. Es el opuesto de "$first", devuelve el último.

4.4.4 Operador $sort

Podemos ordenar por cualquier campo o conjunto de campos utilizando la misma sintaxis que para las *queries* "normales". Si tenemos que ordenar un número de documentos muy elevado, es recomendable desde el punto de vista del rendimiento, que hagamos la ordenación al principio del *pipeline* y además tengamos un índice por el conjunto de campos por los que ordenar. De otra forma, $sort puede ser lento y consumir mucha memoria.

Se pueden utilizar tanto campos existentes como campos proyectados para ordenar. De esta forma podemos por ejemplo completar la agregración en la que calculábamos la media aritmética del precio de los libros por autor, y ordenar los resultados por dicha media calculada ascendentemente.

```
> db.libro.aggregate({"$group":{"_id":"$autor",
"media":{"$avg":"$precio"}}},{"$sort":{media:1}})
{ "_id" : "Markus Zusak", "media" : 9.45 }
{ "_id" : "George R.R. Martin", "media" : 10.75 }
{ "_id" : "Camilla Lackberg", "media" : 11 }
{ "_id" : "Anna Casanovas", "media" : 15.9 }
{ "_id" : "Maggie Stiefvater", "media" : 17.23 }
{ "_id" : "Santiago Posteguillo", "media" : 18.900000000000002 }
{ "_id" : "Isabel Allende", "media" : 21.75 }
```

Para ordenar cada campo es valor 1, para ordenación ascendente y -1 para descendente. En el siguiente ejemplo ordenamos primero por precio descendente y después por páginas ascendente.

```
> db.libro.aggregate({"$project":{precio:1,paginas:1}},
{"$sort":{precio:-1,paginas:1}})
{ "_id" : "9788466637688", "paginas" : 860, "precio" : 22 }
{ "_id" : "9788401342158", "paginas" : 480, "precio" : 21.75 }
{ "_id" : "9788408117117", "paginas" : 1100, "precio" : 21.75 }
{ "_id" : "9788408113331", "paginas" : 290, "precio" : 17.23 }
{ "_id" : "9788468738895", "paginas" : 360, "precio" : 15.9 }
{ "_id" : "9788408118329", "paginas" : 1137, "precio" : 12.95 }
{ "_id" : "9788496208681", "paginas" : 928, "precio" : 12 }
{ "_id" : "9788415140054", "precio" : 11 }
{ "_id" : "9788496208919", "paginas" : 793, "precio" : 9.5 }
{ "_id" : "9788499088075", "paginas" : 544, "precio" : 9.45 }
```

4.4.5 Operador $limit

La función $limit toma un número N y devuelve los primeros N documentos resultantes.

```
> db.libro.aggregate({"$project":{precio:1,paginas:1}},
{"$sort":{precio:-1,paginas:1}},{"$limit":3})
{ "_id" : "9788466637688", "paginas" : 860, "precio" : 22 }
{ "_id" : "9788401342158", "paginas" : 480, "precio" : 21.75 }
{ "_id" : "9788408117117", "paginas" : 1100, "precio" : 21.75 }
```

4.4.6 Operador $skip

La función $skip toma un número N y desecha los primeros N documentos del conjunto de resultados.

```
>
db.libro.aggregate({"$project":{precio:1,paginas:1}},{"$sort":{precio
:-1,paginas:1}},{"$skip":3})
{ "_id" : "9788408113331", "paginas" : 290, "precio" : 17.23 }
{ "_id" : "9788468738895", "paginas" : 360, "precio" : 15.9 }
{ "_id" : "9788408118329", "paginas" : 1137, "precio" : 12.95 }
{ "_id" : "9788496208681", "paginas" : 928, "precio" : 12 }
{ "_id" : "9788415140054", "precio" : 11 }
{ "_id" : "9788496208919", "paginas" : 793, "precio" : 9.5 }
{ "_id" : "9788499088075", "paginas" : 544, "precio" : 9.45 }
```