Upgrade hub

Sentencias II

- Algunos métodos de find
- Ordenaciones
- Operadores
- Filtros
- Ejemplos en todo y Ejercicios

Métodos de .find( )

.distinct( )
.limit( )
.skip( )
.sort( )

db.collection.distinct()

Supongamos:
{ "_id": 1, "dept": "A", "item": { "sku": "111", "color": "red" }, "sizes": ["S", "M"] }
{ "_id": 2, "dept": "A", "item": { "sku": "111", "color": "blue" }, "sizes": ["M", "L"] }
{ "_id": 3, "dept": "B", "item": { "sku": "222", "color": "blue" }, "sizes": "S" }
{ "_id": 4, "dept": "A", "item": { "sku": "333", "color": "black" }, "sizes": ["S"] }

devolviendo valores de un campo: `db.inventario.distinct( "dept" )`

devolviendo valores de un campo array: `db.inventario.distinct( "sizes" )`

devolviendo valores de un campo embebido: `db.inventario.distinct( "item.sku" )`

con condición: `db.inventario.distinct( "item.sku", { dept: "A" } )`

.limit(n) y .skip(n)

1. Consulta de datos

```
db.amigos.find({},{Nombre:1})
```

Personas mayores de 25 años

```
db.amigos.find({"Edad": {$gt: 25}})
```

2. Mostrar proyección

Mostrar nombre y apellidos de las 3 primeras Marisas encontradas

```
db.amigos.find({Nombre: "Marisa" },{Nombre:1, Apellidos:1}).limit(3)
```

Mostrar nombre y apellidos de las Marisas encontradas, empezando por el cuarto documento

```
db.amigos.find({Nombre: "Marisa" },{Nombre:1, Apellidos:1}).skip(3)
```

Ordenaciones

A la hora de realizar una consulta podemos encontrarnos la necesidad de realizar una ordenación de los datos. Ya que si no decimos nada las consultas en MongoDB devolverán los datos tal cual se encuentren almacenados en la base de datos, es decir, con su orden de inserción.

Así si queremos realizar ordenaciones en MongoDB deberemos de utilizar el método **.sort()**. La sintaxis del método MongoDB .sort() es la siguiente:

```
db.coleccion.find(filtro).sort(ordenacion)
```

Indicaremos el campo o los campos por los cuales queremos ordenar y el tipo de ordenación que queremos: ordenación ascendente u ordenación ascendente.

Ordenaciones

La estructura de la ordenación es la siguiente:

.sort({campo1:tipoOrdencion1, campo2:tipoOrdenacion2,..., campoN:tipoOrdenacionN});

Dónde los tipos de ordenación y sus valores son:

- **Ascendente:** 1
- **Descendente:** -1

```
db.alumnos.find().sort(apellido:1)
```

Operadores y Filtros

- Operadores Relacionales,
- Operadores de pertenencia,
- Filtros REGEX, EXISTS,TYPE
- Operadores Lógicos

Operadores Relacionales

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

Operadores Relacionales

“ Igual que ”: Función **$eq**, se usa de la siguiente manera:

```
db.puntuaciones.find( { puntos : { $eq : 95 } } )
```

“ Distinto de”: Función **$ne**:

```
db.puntuaciones.find( {puntos: { $ne : 95 } } ).limit(10)
```

Operadores Relacionales

“ Mayor que ”: Función **$gt**, se usa de la siguiente manera:

```
db.puntuaciones.find( { puntos : { $gt : 95 } } )
```

“ Mayor o igual que ”: Función **$gte**:

```
db.puntuaciones.find( { puntos : { $gte : 95 } } )
```

Operadores Relacionales

“ Menor que ”: Función **$lt**:

```
db.puntuaciones.find( { puntos : { $lt : 95 } } )
```

“ Menor o igual que ”: Función **$lte**

```
db.puntuaciones.find( { puntos : { $lte : 95 } } )
```

Operadores Relacionales.- Ejemplos

```
db.puntuaciones.find( {puntos : { $gte : 90, $lt : 95} } )
```

¿Qué devuelve?

Devolverá todos los documentos que tengan puntajes entre 90 y 94 (incluyendo el número 90 y excluyendo el número 95)

Operadores Relacionales

Éstas funciones también aplican para letras, por ejemplo si se quieren los nombres de personas que empiecen por a, b y c es posible escribiendo el comando:

```
db.directores.find( { nombre: { $lte : "C" } } )
```

Operadores pertenencia

$in, y $nin : devuelven el conjunto de documentos (registros) que pertenecen (o no) al array impuesto:

Operadores de pertenencia

Función **$in**: selecciona los documentos donde el valor de un campo es igual a cualquier valor en la matriz especificada

```
db.tienda.find( { qty: { $in: [ 5, 15 ] } } )
```
selecciona aquellos que tengan cantidad 5 o 15

Función **$ni:** selecciona documentos en los que el valor del campo, no esté especificado en el array o documentos donde el campo no existe

```
db.tienda.find( { qty: { $nin: [ 5, 15 ] } } )
```
selecciona aquellos que no tengan ni 5 ni 15 unidades

Operadores pertenencia

```
db.medicamentos.find( { cantidad: { $in: [ 0, 5 ] } })
```

```
db.medicamentos.update(
                    { tags: { $in: ["analgesia", "resfriado"] } },
                    { $set: { venta:true } }
                  )
```

Operadores - Ejemplo en JS

en JavaScript...

```
1.  conn = new MongoClient();
2.  db = conn.getDB("demografia");
3.  cursor = db.ciudades.find({habitantes:{$gt:1000000}});
4.  while (cursor.hasNext()) {
        printjson(cursor.next());
    }
5.  conn.close()
```

Operadores Relacionales - Ejemplo

```
db.ciudades.find({ciudad:{$ne: “Madrid”}})
```

¿Qué devuelve?

Operadores Relacionales - Ejercicio T^co

Vamos a buscar ciudades que tengan más de un millón de habitantes.

1.- Así el documento que compongamos será:

{$gt:1000000}

2.- Que se anidará al documento con el campo sobre habitantes:

{habitantes:{$gt:1000000}}

3.- Sólo nos quedará poner las consultas mayor que en MongoDB dentro del método .find(). db.ciudades.find({habitantes:{$gt:1000000}}, {NombCiudad:1,_id:0})

Filtros

Una de las maneras más importantes para filtrar información son con los comandos:

- **$exists**
- **$type**
- **$regex**

Filtros - Exists

Si tuviéramos una colección donde no todos los registros tengan determinado campo, podríamos filtrar la información de dos maneras, traer los datos que NO tengan "teléfono", o traer los que SI tengan, ambas opciones se harían de la siguiente manera:

```
db.personas.find( { "telefono" : { $exists : true } } )
```

```
db.personas.find( { "telefono" : { $exists : false } } )
```

Filtros - Type

Suponiendo que tenemos un campo que puede ser tanto como numérico, como tipo string, podemos filtrar la información con el comando **$type** y se usa de la misma manera que el anterior filtro, con la diferencia de que en vez de poner valores true o false se pone el tipo de la variable que se requiere.

```
db.personas.find( { telefono : { $type : “string” } } )
```

Ejemplo:

```
db.notas.find( { "mediaClase" : { $type : [ "string" , "double" ] } },{nombre:1, mediaClase:1, _id:0} )
```

y devolverá tanto los registros que tengan el campo media: 9.33333333333 o “5.5”, pero no devolverá las notas con formato INT, como: 6

Filtros

Tipos:

- double,
- string,
- object,
- array,
- objectid,
- bool,
- null,
- date,
- int,
- decimal,
- number (double, 32-bit integer, 64-bit integer, decimal)
- ...

Filtros - Regex

Por último **$regex** nos ayuda a filtrar información que termine o contenga algún carácter en específico:

```
db.alumnos.find( { nombre: { $regex : "e" } } )
```

Retornará todas las personas que tengan un nombre que contenga la letra "e"

```
db.alumnos.find( { apellido : { $regex : "^R" } } )
```

devolverá aquellos que su apellido empiece con “R”

```
db.alumnos.find( { apellido : { $regex : "z$" } } )
```

Retornará todas las personas que tengan un apellido que termine con la letra "z"

Filtros - Regex

```
db.productos.find( { codigo: { $regex: /s/ } } )
```

Devolverá aquellos documentos que contengan una “s” formando parte de su campo “código”

```
db.productos.find( { descripcion: { $regex: /^S/}})
```

y también funcionará: `db.productos.find({descripcion:/^M/})`

Devolverá aquellos documentos en los que el campo descripción empiece con la letra S

```
db.productos.find( { codigo: { $regex: /s3$/ } } )
```

Devolverá aquellos documentos que contengan un “s3” finalizando su campo “código”

Operadores lógicos

OR

AND

NOT

NOR

Operadores lógicos: OR

Se usa el comando **$OR** cuando se quiera filtrar por una u otra razón, pero que sólo es necesario que se cumpla una de las razones para que se imprima ese resultado, por ejemplo:

```
db.inventario.find( { $or: [ { qtty: { $lt: 20 } }, { precio: 10 } ] } )
```

Esto mostrará aquellos documentos que cumplan que , o bien su stock está por debajo de 20 unidades o su precio es 10, o ambas.

Operadores lógicos: AND

El comando $**and** se usa cuando se quieren que ambas condiciones se aplican

```
db . inventario . find ( { $and : [ { precio : { $ ne : 1.99 } }, { precio : { $ exists : true } } ] } )
```

Esta consulta seleccionará todos los documentos en la inventory colección donde:

- el valor del campo “Precio” no es igual 1.99 **y**
- el campo “Precio” **existe**

lo mismo que: `db . inventario . find ( { precio : { $ ne : 1.99 , $exists : true} } )`

Operadores lógicos: NOT

```
db . inventario . find ( { precio : { $ not : { $ gt : 1.99 } } } )
```

Esta consulta seleccionará todos los documentos en la colección donde:

- el valor del campo “Precio” es menor o igual que 1.99 **o**
- el campo “Precio” no existe

Mientras que `db . inventario . find ( { precio : { $lte: 1.99 }})` devolvería sólo aquellos documentos con el precio menor de 1.99

Operadores lógicos: NOR

Realiza la operación lógica en un array de una o más expresiones de consulta y selecciona los documentos que **fallan en** todas las expresiones de consulta en el array

```
db.inventario.find( { $nor: [ { precio: 1.99 }, { oferta: true } ]  } )
```

Devolverá Todos aquellos documentos que:

- contengan el campo “precio” con valor distinto a 1,99 y que contenga el campo “oferta” distinto de TRUE, **ó**
- contengan el campo “precio” con valor distinto a 1,99 PERO no contengan el campo “oferta”, **ó**
- no contengan el campo “precio”, PERO contengan el campo “oferta” distinto de TRUE **ó**
- no contengan ni el campo “precio” ni el campo “oferta”

Ejercicio 1.-

1.- insertar en la colección ciudades

{ "ciudad" : "Madrid", "habitantes" : 3233527, capital: “si”}
{ "ciudad" : "Barcelona", "habitantes" : 1620943 }
{"ciudad" : "Valencia", "habitantes" : 797028 }
{ "ciudad" : "Sevilla", "habitantes" : 702355 }
{"ciudad" : "Zaragoza", "habitantes" : 679624 }

2.- ordenar por ciudad, ascendente, y sólo obtener los cuatro primeros documentos

3.- ordenar por población descendente

4.- obtener las ciudades que empiecen por “M”

5.- ordenar , a la vez, primero por nombre (desc) y luego por número de habitantes (asc)

6.- Obtener aquellas ciudades que sean capital, y mostrarlo.

7.- Mostrar el nombre de las ciudades que superen un millón de habitantes

8.-¿qué devolverá la consulta: db.ciudades.find({ciudad:{$in:['Avila','Zamora','Madrid']}})?
9.- ¿Número de ciudades que componen la colección?

10.- Ciudades que empiecen por “B” o acaben por “z”

db.collection.renameCollection()

```
db.rrecord.renameCollection("record")
```

db.collection.count()

Devolverá el número total de documentos de una colección.

`db.orders.count()` es igual que : `db.orders.find().count()`

```
db.orders.count( { ord_dt: { $gt: 3 } } )
```

igual que:

```
db.orders.find( { ord_dt: { $gt: 3) } } ).count()
```

IMPORTAR un documento JSON

1.- Abrir el servidor y el cliente de mongo

2.- Situarse **en CMD** en la ruta:
Program Files\MongoDB\Server\4.0\bin>

3.- Ejecutar en CMD:
mongoimport.exe --db NombBD --collection NombColec --drop --file "<Ruta_del_archivo_json_a_insertar>/NombArchivo.json”

Ejercicio 2.-