Mongo DB - Querying

Para todos -> db.getCollection(“ “).find({ })

- FIND
  - ❖ Encontrar todos los documentos donde el valor de la edad es 27

    find({“edad”: 27})

  - ❖ Todos los usuarios con 27 años que se llamen Isabel

    find({“nombre”: “Isabel”, “edad”: 27})

A veces no es necesario recuperar todos los campos del documento, por lo que podemos pasar un segundo argumento que indica cuales queremos recuperar:

- ❖ Recuperar solo el nombre de usuario y email

  find({ }, {“nombre”: 1, “email”: 1})

También se puede especificar cuáles NO queremos recuperar

- ❖ Recuperar todo menos el teléfono y el _id

  find({ }, {“telefono”: 0, “_id”: 0})

FindOne permite recuperar un único documento que cumpla las condiciones especificadas:

findOne({ })

- # OPERADORES CONDICIONALES

    - Edad entre 18 y 30 años

      find({"edad": *{"$gte": 18, "$lte": 30}* })

    - Personas que se registraron antes del 1 de Enero de 2007

      fecha = new Date("01/01/2007")
      find({"registrados": *{"$lt": fecha}* })

    - Usuarios que NO tienen por nombre "Pablo"

      find({"username": *{"$ne": "joe"}* })

$ne *se usa cuando queremos consultar documentos en los que el valor de una clave es distinta (No Equal) a cierto valor.*

- # CONSULTAS DE TIPO OR

Existen dos posibilidades para realizar una consulta "OR":

1. *El operador **"$in"** que puede ser usado para consultar sobre **una variedad de valores para <u>una clave dada</u>**.*
2. *El operador **"$or"** que puede ser usado para consultar sobre **un conjunto de valores dados sobre <u>múltiples claves dadas</u>.***

    - Personas que tienen un dni autorizado (725, 542 y 390):

      find({"dni": *{"$in": [725, 542, 390] }* })

    - Si el usuario puede ser un número o una persona se puede poner lo siguiente:

      find({"user_id": *{"$in": [12345, "Pablo"] }* })

*El operador **"$nin"** es el opuesto de "$in", por lo cual, devuelve los documentos que no coinciden con ninguno de los criterios dados en el array de valores.*

"$or" toma un array con posibles criterios de coincidencia.

- Persona que tienen 30 años ó que no son fumadores

  find("$or": *[{"edad": 30}, {"fumador": false}]* })

- Personas que tienen 30, 34 y 35 años que no son fumadores

  find({"$or": [*{"edad":* {"in": [30, 34, 35] } *}, {"fumador": false}*] })

***"$or"** siempre funciona, pero es mejor usar **"$in"** ya que es más eficiente.*

***"$not"** puede ser aplicado sobre otros criterios*

- Recuperar documentos para el id_num que tiene un valor de 1 modulo 5 (es decir: 1, 6, 11, …)

  find({"id_num": *{"$mod": [5, 1] }* })

- Recuperar elementos con id: 2, 3, 4, 5, 7, 8, 9, 10, 12…

  find({"id_num": *{"$not":* {"$mod": [5, 1] } *}* })

*El **null** en las consultas funciona de una manera peculiar. No solo devuelve aquellos documentos en los que coincide, sino que también devuelve aquellos en los que no existe la clave usada.*

Así por ejemplo para la consulta: **find({"z" : null})**

Devolvería:
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523621"), "z" : null }
{ "_id" : ObjectId("4ba0f0dfd22aa494fd523622"), "y" : 1 }
{ "_id" : ObjectId("4ba0f148d22aa494fd523623"), "y" : 2 }

- CONSULTAS SOBRE ARRAYS

Si se inserta un documento con: ***"fruta": ["manzana", "plátano", "melocotón"]***
y se realiza la siguiente consulta:

find({"fruta": "plátano"})

entonces tendrá éxito sobre el documento insertado.
La consulta es equivalente a si se tuviera un documento de la forma:

{"fruta": "manzana", "fruta": "plátano", "fruta": melocotón"}

Si se tuviera la siguiente colección de documentos:

{ "_id" : 1, "fruta" : ["manzana", "platano", "melocoton"] }
{ "_id" : 2, "fruta" : ["manzana", "pera", "naranja"] }
{ "_id" : 3, "fruta" : ["cereza", "platano", "manzana"] }

Se podría buscar todos los documentos que tienen ***a la vez*** *"manzana" y "platano" de la siguiente manera:*

find({"fruta": {$all: ["manzana", "platano"] } })

No importa el orden en la consulta al poner el $all, pero si lo quitas sí importa.

Esta consulta, se diferencia de: ***find ({ "fruta": "manzana", "fruta": "platano" })***
en que esta última solo devolvería un documento así:

{ "_id" : 1, "fruta" : ["manzana", "platano"] }

- Documentos que tengan en segunda posición del array un melocoton

  find({"fruta.1": "melocoton"})

- Arrays con tamaño 3

  find({"fruta": *{"$size": 3}* })

- EL OPERADOR “$slice”

Si se quieren recuperar el último amigo de un documento como el siguiente:

```
"amigos" : [
      {
         "nombre" : "Sergio",
         "ocupacion" : "Estudiante"
      },
      {
         "nombre" : "Claudia",
         "ocupacion" : "Panadera"
      }
   ]
```

Se podría hacer de la siguiente manera:

findOne({ }, {“amigos”: {“$slice”: -1} })

Además:

findOne({ }, {“amigos”: {“$slice”: 1} })
Devolvería el primero

findOne({ }, {“amigos”: {“$slice”: 2} })
Devolvería los dos primeros

findOne({ }, {“amigos”: {“$slice”: [23, 10]} })
Se salta los 23 primeros y devuelve del 24 al 33

- ARRAYS Y RANGOS

Si se quiere encontrar los documentos donde x está entre 10 y 20, y existe alguna x que es un array entre 5 y 25, este será retornado en la busqueda ya que 25 es más grande que 10 y 5 es más pequeño que 20.

*Si se quiere evitar esto se puede usar los métodos **min()** y **max()** para limitar el rango de índices considerado en la consulta para los valores de los operadores “$gt” y “$lt”*

Ejemplo:
find({“x”: *{“$gt”: 10, “$lt”: 20}*).min({“x”: 10}).max({“x”: 20})

- CONSULTAS SOBRE DOCUMENTOS EMBEBIDOS

Teniendo:

```
 "direccion" : {
       "calle" : "Mayor, 3",
       "ciudad" : "Madrid"
}
```

Si se quiere recuperar un documento en el que la dirección tome el valor: “Mayor, 3. Madrid” se haría de la siguiente manera:

find({“direccion”: *{“calle”: “Mayor, 3”, “ciudad”: “Madrid”}* })

Sin embargo, esta consulta encaja con la estructura del subdocumento. No sería muy eficiente utilizarla si se añade un campo más, o la estructura cambia.

Por ello utilizaremos esta sintaxis para la consulta, en la que no influye la estructura:

find({“direccion.calle”: “Mayor, 3”, “direccion.ciudad”: “Madrid” })

- LIMITAR, SALTOS Y ORDENACIONES

*Para **limitar** los resultados a 3, se podría encadenar la función **limit()** sobre la llamada a find()*

db.getCollection(“ “).find(.....).limit(3)

Si existen menos, se retornan los que hayan.

*Para **saltar** los n primeros documentos que encajen se usa: **skip()***

db.getCollection(“ “).find(....).skip(3)

Si existen menos, no retornará nada.

*Para **ordenar,** la función **sort()** toma las claves por las que quiere ordenar, y los valores que indican un sentido de la ordenación: 1 (ascendente) ó -1 (descendente). Cuando existe más de una clave los resultados son ordenados en ese orden*

find.( ).sort({“nombre”: 1, “edad”: -1 })