TEORÍA

• .find()

El find() se utiliza para hacer búsquedas en JavaScript para las bases de datos que estamos tratando. Dentro del find() se deberá colocar unos corchetes "{}", y dentro de ellos todas las búsquedas que queramos hacer. Si lo dejamos en blanco nos devolverá todos los documentos de la base de datos.

Se utiliza de la siguiente forma:

db.coleccion.find({<query>})

Donde **db** especifica que se trata de una base de datos (que debemos seleccionar previamente en el terminal) y **colección** es donde irá la colección de nuestra base de datos en la que trabajaremos.

➢ $eq

La cláusula $eq se utiliza para igualar un campo a un valor. Dentro del find se utiliza de esta manera:

<campo>: {$eq: <valor>}

➢ $gt, $gte, $lt, $lte

Las cláusulas $gt, $gte, $lt y $lte sirven para hacer búsquedas en cuanto a cantidades, mayores, menores o iguales. Los significados de cada una son:

- **$gt:** mayor que (greater than)
- **$gte:** mayor o igual que (greater than equal)
- **$lt:** menor que (less than)
- **$lte:** menor o igual que (greater than equal)

Se utilizan los cuatro de la siguienta manera:

<campo>: {$gt: <valor>}

➢ $in, $nin

La cláusula $in se utiliza para incluir varios valores en una misma búsqueda. De esta forma, todo lo que vaya dentro de un array posterior al $in irá incluido en el campo:

<campo>: {$in: [<valor1>, <valor2>,…]}

La cláusula $nin es la opuesta a $in. Se escribe igual que $in pero sustituyendo $in por $nin, usándose igualmente un array. La búsqueda seleccionará todos los documentos que no contengan los valores del array.

➢ $ne

La cláusula $ne es contraria a $eq. Se buscarán todos los documentos cuyo valor del campo no sea igual al escrito tras el $ne. Se utiliza similar al $eq:

<campo>: {$ne: <valor>}

➢ $not

La cláusula $not se utiliza para seleccionar los documentos que no tengan el valor escrito en el campo seleccionado. Se escribe de la siguiente manera:

<campo>: {$not: <valor>}

Dicho <valor> puede contener varias cláusulas dentro, no tiene porque ser únicamente un valor.

➢ $regex

La cláusula $regex es un tanto diferente a las vistas anteriormente. Se utiliza para incluir una secuencia de caracteres de uno o más valores. Se utiliza de la siguiente forma:

<campo>: {$regex: /<secuencia>/}

De esta forma, si se sustituye la <secuencia> por una letra como la “m”, buscará todas las palabras que contengan una m. Se pueden añadir secuencias más largas, como por ejemplo “mar”, buscando entonces todas los documentos que tengan esa secuencia.

➢ $and

La cláusula $and se utiliza antes de colocar cada campo del que haremos la selección. Usando el $and especificaremos que pondremos varias búsquedas dentro y que todas deben cumplirse.
Si no se utilizase, con poner una coma entre cada cláusula funcionaría como un $and, pero debido a ciertos problemas de incongruencia al usar distintas cláusulas para un mismo campo, es necesario usar el $and para especificarlo.

Se utiliza de la siguiente manera:

.find({$and: [{<query1>}, {<query2>}] })

➢ $or, $nor

La cláusula $or se utiliza igual que la cláusula $and, pero en vez de poner una serie de búsquedas que deben cumplirse, pone varias búsquedas, pero con que se cumpla una se mostrará ese documento. Funciona de forma semejante a un "o" (**o** una cosa **o** la otra **o** las dos).
Se escribe de forma similar:

.find({$or: [{<query1>}, {<query2>}] })

➢ .pretty(), .count()

El pretty() y el count() se utilizan de forma semejante al find. Mientras que en el find() se escriben todas las cláusulas de la búsqueda que vamos a hacer, pretty y count especifican como queremos ver esos datos.

Con pretty() podremos ver los datos de forma ordenado, ya que de predeterminadamente se nos muestran todos en una misma fila. Con pretty() separará cada documento y cada valor.
Con count() contará el número de documentos que nos devuelvo. No nos mostrará cada documento, sino un número tan solo.

• aggregation()

El método aggregation se utiliza como una extensión del método **find()** ya que nos permite un uso mas especializado y amplio, pudiendo hasta realizar búsquedas entre distintas colecciones. Se utiliza de la misma forma en la que el **find()** a simple vista:

db.coleccion.aggregation([ ])

Dentro de los corchetes podremos añadir distintas búsquedas que denominaremos **Pipeline Stages**, dentro de estas podremos usar los operadores, llamados aquí como **Pipeline Operators**. Muchos de los operadores del find se pueden usar dentro de los stages, pero con una forma de escritura distinta.

Para poner los stages dentro de un aggregation deberán separarse cada uno por corchetes, de esta forma:

db.coleccion.aggregation([
{ stages1 },
{ stages2 }
])

➲ Stages

➢ $match

El operador **$match** es uno de los más usados, se emplea de la siguiente forma:

db.coleccion.aggregation([{$match: <expression>}])

Dentro se colocará una expresión por la que la operación filtrará. Si se coloca que cierto campo debe tener X valor, tomará todos los que cumplan dicha función y no mostrará el restro.

➢ $group

El operador **$group** es otro de los más usados, se emplea de la siguiente forma:

db.coleccion.aggregation([{$group: {<$expression>} }])

Dentro se colocarán distintas expresiones por las que agruparemos los resultados, por ejemplo si hay un valor que se repite y queremos usarlo como id deberemos colocar en la expresión: **{_id: "$field"}**.

➢ $project

El operador project sirve para decidir que campos se mostrarán y cuales no, se utiliza de una forma distinta a los dos anteriores:

db.coleccion.aggregation([{$project: {<expression>:0, <expression>:1,…}])

Se colocarán los campos uno a uno separados por comas, como si se tratase del **insert()** y pondremos un **0** o un **1** dependiendo si queremos que se muestre o no. El cero no se mostrará y el uno sí. No es necesario poner cero en todos, si tan solo ponemos los **1** solo se mostrarán esos.

➢ $count

Automáticamente cuenta los documentos que obtenemos:

db.coleccion.aggregation([{$count: <field_name>}])

Pondremos delante de él el nombre que queramos darle al campo que nos devolverá al contarlo.

➢ $merge

El **$merge** se utiliza para implementar la búsqueda que estamos haciendo en una colección dentro de la base de datos.

db.coleccion.aggregation([{$merge: "nombre de la nueva colección" }])

➢ $unwind

El **$cond** se utiliza para convertir cada valor de un array en un solo campo de forma independiente, duplicando el documento por cada valor del array.

db.coleccion.aggregation([{ $unwind: "$nombreDelCampo" }])

➢ $sort

El **$sort** se utilzia para ordenar los documentos devueltos en un orden determinado (-1, 1).

db.coleccion.aggregation([{ $sort: { "campo": 1/-1} }])

➢ $lookup

El **$lookup** se utiliza para unir una colección a un stage en el que estemos, pudiendo ser otra colección o habiéndola pasado por búsquedas ya.

```
db.coleccion.aggregation([ {
$lookup:
{
from: <colección para unir>,
localField: <campo del anterior stage>,
foreignField: <campo de la colección que vas a unir>,
as: <nombre al que le darás a la nueva colección>
}
} ])
```

➲ Operators

➢ $cond

El **$cond** se utiliza como un if/else en otros lenguajes de programación. Se utiliza una condición, si se cumple se realizará un código que elijamos, sino se realizará otro o directamente nada. Se emplea de la siguiente forma:

db.coleccion.aggregation([{$project:
{$cond:{if:{<condición>},then:<condición>,else:<condicion}}}])

➢ $cond

El **$cond** se utiliza como un if/else en otros lenguajes de programación. Se utiliza una condición, si se cumple se realizará un código que elijamos, sino se realizará otro o directamente nada. Se emplea de la siguiente forma:

db.coleccion.aggregation([{$project:
{$cond:{if:{<condición>},then:<condición>,else:<condicion}}}])

➢ $sum, $substract, $divide, $multiply

Estos operadores son operadores de operaciones. Todos se usan de la misma forma pero con operaciones distintas (suma, resta, división, multiplicación).

db.coleccion.aggregation([{$project:
{$sum: [“$valor1”,”$valor2”]}
}])

➢ $round

El **$round** se utiliza de forma similar a los anteriores, pero en vez de realizar una operación redondea el valor dado por el número de decimales que deseemos.

db.coleccion.aggregation([{$project:
{$round: [“$valor1”,<número de decimales>]}
}])

- **$expr**

El **$expr** se utiliza dentro del match para poder usar operadores como el $eq, $gte….

db.coleccion.aggregation([{$match:
{$expr: {<operación>} }
}])