CRUDS

Para create ya vimos insertOne e insertMany -Puedo crear mis propios id's

```
    >db.personas.insertMany( [ {_id:"yoga", nombre:"Natasha"}, {_id:"running",
nombre: "Carlos"} ] )
```

Yo puedo reescribir los id's, pero no repetirlos en dos valores distintos

- El comportamiento de Mongo por defecto es que cuando encuentra un error deja de actuar
- Si sobreescribí una clase y luego tuve un error, la primera acción se realizará pero las siguientes no

Con el parámetro ordered: false se obvia este comportamiento y si encuentra un fallo sigue con las otras operaciones

```
    >db.personas.insertMany( [ {_id:"yoga", nombre:"Natasha"}, {_id:"running",
nombre: "Carlos"} ], {ordered: false} )
```

Importar un archivo JSON con mongoimport( )

Hay que salir de la shell de mongo con ctrl+C

- Me situo en el directorio de la carpeta.
- El documento es una matriz. Lo especifico con --jsonArray
- Uso el --drop para borrarla por si ya existe

  ```
    C:> mongoimport nombre_del_archivo.json - d dbnombre_de_la_base_de_datos -
  c nombre_de_la_colección --jsonArray --drop
  ```

- inicio la shell de mongo con mongo

  ```
    > show dbs
    > use dbnombre_de_la_base_de_datos
    > show collections
  ```

Resumen:

```
- mongoimport
- -d para el nombre de la database
- -c para el nombre de la colección
- --jsonArray para especificar que es un arreglo
- --drop para borrar si ya hubiera una existente
```

Operadores en MongoDB

> En la documentación oficial, en Reference/Operators estan los query and projection operators, logical, evaluation, array...

- Para encontrar a los usuarios por el apellido

```
> db.usuarios.find( {apellido: "Santos"} ).pretty()
```

- Se puede usar el operador $eq para encontrar valores iguales

```
> db.usuarios.find( { "Salario Anual": {$eq:"49000"} } )
```

Operadores de comparación

- Si uso findOne, solo me mostrará el primero. Si uso find me mostrará todos
  - **$ne** not equal: traerá aquel o aquellos documentos que no sean igual a esta condición o filtro

```
> db.usuarios.find( { "Salario Anual": {$ne:"49000"} } )
```

  - **$gt** greater than, mayor que
  - **$gte** greater than or equal, mayor o igual que
  - **$lt** less than, menor que
  - **$in** si los elementos estan en la lista

```
> db.usuarios.find( {"Salario Anual": { $in:[ "48000","51000",
"54000" ] } } )
```

  - **$nin** not in, incluirá los que no tengan los elementos de la lista

Operadores de lógica

- **and** y
- **or** o
- **not** ninguno
- **nor** cualquiera que no sean estas dos
- Ejemplos
    - Tráeme los que tengan el apellido Pérez o Gutierrez

```
  > db.usuarios.find( {$or: {apellido:"Pérez"}, {apellido:"Gutiérrez"}
} )
```

    - Tráeme los que tengan los dos apellidos

```
  > db.usuarios.find( {$and: {apellido:"Pérez"}, {apellido:"Gutiérrez"}
} )
```

  Puedo usar operadores dentro de las consultas

```
  > db.usuarios.find( $and: {"Salario": {$gt: "430000"}, "Salario":
{$lt:54000} } ).pretty()
```

Si pongo .count() al final me dice cuántos matches en número

Puedo usar el not para decir no igual a simulando el **ne**

```
> db.usuarios.find("Salario anual": {$not: {$eq:"49000"}}).count()
```

Operadores de elemento

- **$exixts** busca si existe ese campo en los documentos
    - Esto mostrará aquellos que no tengan Salario Anual

```
> db.usuarios.find( {"Salario Anual": {$exists:false} } ).pretty()
```

- **type** selecciona el documento si especificas el tipo de dato que tiene ese documento: number, null, boolean, string

```
      > db.usuarios.find( {"Salario Anual": {$type:"string"} } ).pretty()
      > db.usuarios.find( {"Salario Anual": {$type:"number", $gt:30000} }
).pretty()
```

Operadores de evaluación

- **regex** encuentra cualquier texto en cualquier expresión

```
  > db.usuarios.find( {"Categoría": {$regex: /Jefe/}} ).pretty()
```

- **expr** compara expresiones, como por ejemplo ventas y objetivos. Tiene una sintaxis específica
    - uso greater than

```
    > db.ventas.find( {$expr: { $gt["$ventas", "$objetivos"] } } )
```

- Las ventas superen el objetivo pero si las ventas son menor que 100 traer solo las ventas si superaron por 20 las del objetivo.
- Primero mayor que con el objetivo -usaré then (después). $cond para la condición con if
- Ventas less than es la condición, luego uso el then
- Substract para restar

```
  >db.ventas( { $expr: {$gt: [{$cond: {if:{$lt:["$ventas",100] }, then:
{$substract:["$ventas", 20]} else: "$ventas"}}, "$objetivo"  ] } } )
```

Arrays

- tengo este objeto en usuarios: { "_id": "1234", "nombre": "Pepito", "direccion": { "ciudad": "Barcelona" } }

Para buscar por dirección Barcelona es

```
   > db.usuarios.find( { direccion:{ciudad:"Barcelona"} } ).pretty()
   > db.usuarios.insertOne({nombre:"David", direccion:{ciudad: "Barcelona",
calle:"Còrsega"}})
```

Cuando uso find hago una búsqueda exacta. Sólo va a encontrar lo que coincida exactamente por lo que David no saldría

Si quiero que salga David sería con la notación con punto:

```
> db.usuarios.find( { "direccion.ciudad":"Barcelona"}  ).pretty()
```

- **$size** busca el número de elementos dentro de una matriz - Me devuelve el documento que tiene 2 aficiones

  ```
  > db.usuarios.find({aficiones: {$size:2}})
  ```

- **$all** coge todos los elementos independientemente del orden - si tiene nadar y correr, puede tener otras aficiones también que lo mostrará

  ```
  > db.usuarios.find( {aficiones: {$all:["nadar", "correr"] } } )
  ```

- **$elemMatch** es si coincide con el elemento
- Busca dónde la nota de mates sea mayor a 85 e inglés a 80 pero con and busca que uno de los dos se cumpla porque busca en todo el documento

  ```
  > db.usuarios.find( {$and:[ {"notas.mates":{$gt:85}, {"notas.ingles:
  {$gt:80 } } } ] } ).pretty()
  ```

- Para hacer que busque dentro de ese elemento específico usaré **elemMatch**
- Ahora solo devuelve los que se cumplen ambas condiciones >db.usuarios.find({notas: {$elemMatch {mates:{$gt:85}, {ingles: {$gt:80 } } } }).pretty()

find y el cursor

Puedo guardar en variables una busqueda

```
> const objetoCursor = db.usuarios.find()
```

De esta manera solo saldrá de uno en uno

```
> objetoCursor.next()
```

Con .hasNext() pregunta si se quiere el siguiente

Ordenando con sort()

- **sort** Se puede añadir el orden que se quiere - Si le pongo en 1 lo hace en orden descendiente, al contrario con -1

```
> db.usuarios.find().sort({Empleo_ID:1})
> db.usuarios.find().sort({Oficina:-1, Empleo_ID:1})
```

skip() y limit()

- skip() salta - Me salto los primeros 10

```
> db.usuarios.find().sort({Empleo_ID:1}).skip(10)
```

- limit() limita - Sáltate los 10 primeros, me devuelvas solo los 2 primeros despues de esos 10 que te saltas

```
> db.usuarios.find().sort({Empleo_ID:1}).skip(10).limit(2)
```

Proyección

- Son los datos que vamos a mostrar, proyectar con find
- Proyecta los nombres. Aparecen todos los nombres de la colección

```
> db.usuarios.find( {}, {"Nombre": 1} )
```

Proyección en Matrices

Devuelve todos los documentos con aficiones

```
> db.usuarios.find({aficiones:1}). pretty
```

Si quiero que solo muestre la afición que he escogido previamente uso el .$: 1

```
> db.usuarios.find({aficiones: "correr"},{aficiones.$:1}). pretty
```

$slice

- **$slice** con 1 argumento es cuántos cojo - con dos es cuántos me salto y cuantos corto

```
> db.usuarios.find( {nombre: "David"}, { aficiones: {$slice:[1, 1]} } )
```

Update

```
> db.usuarios.updateOne({"nombre":"Juan"}, {$set: { notas:[ { año:1,
mates:0,ingles:3 } ] } } )
```

- Puedo usar updateMany y el operador $in para incluir dos o más documentos

```
> db.usuarios.updateMany({"nombre":{$in: ["Juan","Alfonso"] } }, {$set: {
notas:[ { año:1, mates:0,ingles:3 } ] } } )
```

Incrementando valores con $inc

- **$inc** incrementaen n, se pueden usar negativos

```
> db.usuarios.updateOne({nombre: "Juan"}, {$inc:{edad:1} } )
```

Actualizando valores con $min, $max, $mul

- **$min** solo lo va a actualizar si es menor que el valor actual, es decir si resta

```
> db.usuarios.updateOne({nombre: "Juan"}, {$min:{edad:20} } )
```

- **$max** lo mismo que min pero al revés

- **$mul** es multiplicar
- Si quiero que incremente la edad el 10% lo multiplico por 1.1

```
> db.usuarios.updateOne({nombre: "Juan"}, {$mul:{edad:1.1} } )
```

Eliminando un campo con $unset

- **$unset** es el opuesto a set

```
> db.usuarios.updateOne( {nombre:"Juan"}, {$set:{dirección: null} } )
```

Si quiero borrar dirección uso $unset y desaparece el campo

```
> db.usuarios.updateOne( {nombre:"Juan"}, {$unset:{dirección:""} } )
```

Renombrando con $rename

- Ahora nombre irá con mayuscula

```
> db.usuarios.updateOne( {nombre: "Juan"}, {$rename:{nombre: "Nombre"} } )
```

Upsert

- Es una mezcla de insert y update. Si exsite actualizalo, si no existe, créalo
- Se usa con un objeto incluyendo upsert

```
> db.usuarios.updateMany({"nombre": "Juan" }, {$set: { notas:[ { año:1,
mates:0,ingles:3 } ] } }, {upsert: true} )
```

Delete

- deleteOne(), deleteMany() . Puedo usar el operador $in para incluir dos o mas en la eliminación con many

```
> db.usuarios.deleteMany( {nombre: {$inc:["Mario", "Juana"] } } )
```

Borrando diferentes niveles en Mongo

- Borrar todos los documentos:

```
> db.usuarios.deleteMany( { } )
```

- Otra opción es usar drop para borrar la colección

```
> db.usuarios.drop()
```