CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UNIDAD 8

BASES DE DATOS NOSQL: MONGODB

BASES DE DATOS 22/23
CFGS DAW

FORMULARIO RESUMEN

Autores:
Abelardo Martínez y Pau Miñana

UD8. MONGODB: FORMULARIO RESUMEN

Este formulario obvia algunos parámetros de las funciones que no vamos a usar, por tanto no define la nomenclatura completa de algunas de ellas. Podéis encontrar la nomenclatura completa de todas las funciones, con ejemplos de uso, en la documentación de MongoDB

https://www.mongodb.com/docs/manual/reference/method/

1. DOCUMENTOS JSON

```
{
"NombreCampo1":"valor_texto",
"NombreCampo2":valor_numérico[.decimales],
"NombreCampoDoc":{"NC1":"valor","NC2":"valor",…},
"NombreCampoArray":[
                        "Valor1_Texto",
                        Valor2_numérico,
                        {"NombreCampo1":"valor","NombreCampo2":"valor",…}
                    ]
}
```

- Se pueden combinar/incluir los campos como se desee en Documentos o Arrays.
- La estructura de los documentos en colecciones no tienen por qué ser iguales en absoluto
- Se recomienda siempre el uso de comillas en los nombres de los campos.
- Los valores numéricos no llevan comillas.
- Para definir **fechas** usar un documento `{"$date":"año4cifras-mes-dia"}` en documentos JSON a importar e `ISODate("año4cifras-mes-dia")` en los INSERTs.

2. DATA DEFINITION LANGUAGE (DDL)

show databases/show dbs : mostrar todas las BD.

use NombreBD : seleccionar la BD NombreBD para usarla, si no existe, la crea.

db : Ver BD activa en este momento.

db.dropDatabase() : Borrar BD en uso.

show collections : Mostrar colecciones de la BD en uso.

db.createCollection(NombreColeccion): Crear una colección en la BD.

db.NombreColeccion.drop() : Borrar la colección NombreColeccion.

3. DATA MANIPULATION LANGUAGE (DML)

Cuando hablamos de documentos, pueden definirse en ese momento o estar predefinidos anteriormente usando una variable: var doc1 = {"NombreCampo":"Valor",...}

db.NombreCol.insertOne({documento}) : inserta un documento en la colección NombreCol,

db.NombreCol.insertMany([{documento1},...]) : inserta varios documentos usando un Array.

db.NombreCol.deleteOne({filtro}) : Borra el primer documento que cumple el filtro.

db.NombreCol.deleteMany({filtro}) : Borra todos los documentos que cumplen el filtro.

db.NombreCol.updateOne({filtro},{acciones},{upsert:true}): Actualiza el primer documento que cumple el filtro, con las acciones especificadas. El tercer valor, opcional, hace que si no encuentra ningún documento según el filtro, añada uno nuevo con lo especificado en las acciones. Acciones:

- *$set:{doc}* documento con campos y valores a actualizar. *$set:{"campo1":"T", campo2:5}*
- *$unset:{doc}* documento con los campos a eliminar, se pone un valor al campo (o "", por ejemplo) para que el documento sea válido pero estos valores no afectan a la orden. *$unset:{"campo1":""}*
- *$inc:{doc}* documento con los campos y cuánto hay que incrementarlos *{"campo1":5}*
- *$rename:{doc}* documento con los nombres a cambiar *{"campo1":"NuevoNombre"}*

Ten en cuenta que las acciones no se pueden repetir, si se usan varios *$set*, solo se aplica el último. Para afectar varios campos se ponen todos en el mismo documento .

db.NombreCol.updateMany({filtro},{acciones},{upsert:true}): Equivalente al anterior pero los cambios afectan a todos los documentos que cumplan el filtro.

4. DATA QUERY LANGUAGE (DQL)

***db.NombreCol.find({filtro},{proyeccion}).operaciones* :** Realiza una consulta. Todos los documentos y operaciones de esta función son opcionales.

- Proyección: Equivalente al SELECT en SQL, documento con los campos a mostrar. Si se incluye solo se muestran los campos puestos a 1. La _id se muestra por defecto excepto que se ponga a 0. Si se ponen campos a 0 muestra todo menos esos. Ejemplo: *{"Nombre":1, _id:0}*
- Operaciones: A destacar las siguientes
  - *.limit()* limita el número de documentos a mostrar. (*.limit(3)* mostrar 3 resultados)
  - *.sort()* Equivalente al ORDER BY, recibe un objeto con los parámetros de ordenación. Valor 1 ordenar de menor a mayor, -1 de mayor a menor. (*.sort({"Nombre":1})* Ordenar alfabéticamente por Nombre)
  - *.count()* Sin parámetros. Sólo se devuelve el número de objetos de la respuesta.
- Filtro: Equivalente al WHERE en SQL, documento con los campos y valores para filtrar. Se

pueden usar distintos operadores en el mismo.

- ***{"Campo1":Valor1}*** filtra los documentos en que campo 1 valga "Valor1".
- ***{"NombreCampoDoc.NC1":"valor"}*** filtra usando campos dentro de documentos
- ***{"NombreCampoArray.X":valor}*** filtra usando un elemento del array (X es el índice)
- ***{"NombreCampoArray.3.NombreCampo1":"valor"}*** filtra usando NombreCampo1 en el 3r elemento del Array (tendría más sentido si hubiese más objetos como el 3º). *NOTA: En los ejemplos anteriores, para que el punto no dé errores el nombre completo debe estar siempre entre comillas. Por otro lado, al filtrar por valores en el Array se selecciona el documento entero, el Array no se separa, así que en el resultado se verán asimismo TODOS los elementos del Array*
- *{Campo1:**{$exists:true/false}**}* filtra los documentos que tienen/no tienen Campo1.
- *{Campo1:**{$in/$nin:[v1,v2,..]}**}* Filtrar si Campo1 está/no está en una lista de valores.
- *{Campo1:**{$gt/$gte:v1}**}* Filtrar si el Campo1 es mayor/mayor o igual a v1.
- *{Campo1:**{$lt/$lte:v1}**}* Filtrar si el Campo1 es menor/menor o igual a v1.
- *{Campo1:**{$ne:v1}**}* Filtrar si el Campo1 es distinto a v1.
- ***{$and:[{doc},{doc},...]}*** Para combinar un array de documentos dentro de un mismo filtro en una operación AND. En realidad es equivalente a separar por comas todas esas condiciones dentro de un mismo documento, útil para poner varias condiciones sobre un mismo campo o si hay varias variables definidas con cada una de las condiciones, ya que cada una es un documento distinto:

  var a1 ={Campo1:"Valor"}

  var a2 = {Campo2:"Valor2"}

  ..find({$and:[a1,a2]}) es equivalente a *..find({Campo1="Valor", Campo2="Valor2"})*,

  sin $and no se pueden usar esas variables pues son 2 documentos.
- ***{$or:[{doc},{doc},...]}*** Operación lógica OR entre los documentos incluidos.
- ***{$nor:[{doc},{doc},...]}*** Filtrar cuando alguno de los documentos no se cumpla. Debe ser un Array aunque solo se use un documento para una operación NOT simple.

  ..find({$nor:[{Campo1:4}]}) filtrar los documentos con Campo1 NO igual a 4.

db.NombreCol.aggregate([{docStage1},{docStage2},...]). Stages:

- *$match:{doc}* Similar al filtro de la función find o al WHERE de SQL.
  `db.NombreCol.aggregate([{$match:{"NombreCampo1":"valor"}}])`
- *$project:{doc}* Parecido a la proyección del find o al SELECT de SQL. Los campos no incluidos en el project no pasan a la siguiente fase.
  - `db.NombreCol.aggregate([{$project:{"NombreCampo2":1}}])`
  - `..aggregate([{$project:{"NombreCampo1":1}},{$match:{"NombreCampo1":"valor"}}])`
  - `..aggregate([{$match:{"NombreCampo2":5}},{$project:{"NombreCampo1":1}}])`
  - `..aggregate([{$project:{"NombreCampo1":1}},{$match:{"NombreCampo2":5}}])`
    `NombreCampo2 se pierde al aplicar el project antes.`

- *$sort:{doc}* Similar a la operación sort o al ORDER BY de SQL.
- *$limit: X* Similar a la operación limit del find, devuelve solo los primeros X documentos.
- *$count:"Nombre"* Similar a la operación count del find, pero devuelve un documento con el campo Nombre que tiene el numero de elementos como valor.
- *$unwind:"$NombreArray"* Separar los documentos según elementos del array NombreArray
- *$lookup:{doc}* Hacer JOINs, el documento necesita 4 parámetros:
  - *from:* nombre de la colección a incluir como objeto embebido
  - *localField:* nombre del campo en la colección del aggregate a unir con foreignField.
  - *foreignField:* nombre del campo en la colección del from a unir con localField.
  - *as:* nombre que daremos al campo con los documentos de la colección del from.

  ```
  Supongamos otra colección "col2" con un campo "ajena" que funciona como clave ajena de "NombreCampo2" en nuestra colección:
  db.NombreCol.aggregate([{$lookup:{from:"col2",localField:"NombreCampo2",
                                    foreignField:"ajena",as:"DocsCol2"}}])
  ```

- *$group:{doc}* Similar al GROUP BY de SQL y al uso de funciones agregadas. Para el doc:
  - *_id* es necesario e indica el criterio de agrupación (GROUP BY), con **$**. Vacío si no se quiere agrupar y sólo se quiere calcular la agrupada de todos los documentos.
  - *Nombre:{acumulador}* el acumulador es la función agregada a usar, se le pasa el campo sobre el que actúa como dato, con **$**. Más comunes: *$max*, *$min*, *$count:{}*, *$avg*, *$sum*

  ```
  Supongamos que tenemos varios documentos como los del apartado 1 y queremos agruparlos por "NombreCampo1" y ver la media de "NombreCampo2"
  db.NombreCol.aggregate([{$group:{_id:"$NombreCampo1",
                                   Media:{$avg:"$NombreCampo2"}}}])
  Contar los documentos en la colección
  db.NombreCol.aggregate([{$group:{_id:"", Cantidad:{$count:{} }}}])
  ```

- Para aplicar funciones agregadas a los elementos de un array dentro de un mismo documento se puede poner un acumulador en el project y lo calcula directamente; el Array no se agrupa realmente, así que sus elementos se pueden ver igualmente.

  ```
  Supongamos ahora que en "NombreCampoArray" los elementos son números, para ver el máximo de los números dentro de "NombreCampoArray" en cada documento
  db.a.aggregate([{$project:{Maximo:{$max:"$NombreCampoArray"}}}])
  ```