MongoDB

BASE DE DATOS NOSQL

Contenidos

- Operaciones sobre Bases de datos y Colecciones
- Tipos de Datos
- CRUD’s
- Proyecciones

Operaciones sobre bd

- Comando '**use DATABASE_NAME**'
  - Crea la bd si no existe. Si existe devuelve la base de datos existente.

```
use mydb
```

- Para saber la bd en la que estamos: **db**

```
>db
mydb
```

- Listar todas las bbdd: **show dbs**

```
>show dbs
local     0.78125GB
test      0.23012GB
```

  - *La bd creada no aparece hasta que se inserta un documento*
  - *En MongoDB la bd por defecto es test. Si no se crea ninguna bd las colecciones se guardan en test*

- Para eliminar una bd: **db.dropDatabase()**

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
```

Operaciones sobre colecciones

- En mongodb no hace falta crear colecciones, se crean al insertar un documento

```
>use test

switched to db test

>db.miColeccion.insert({"name" : “lobo”})
```

- Se puede utilizar el comando db.createCollection(nombre, opciones)

Parámetro	Tipo	Descripción
Nombre	String	Nombre de la colección
Opciones	Document	(Opcional) Especifica datos sobre tamaño de memoria e indexación

- Eliminar una colección (devuelve true si se elimina correctamente): db.nombre_coleccion.drop()
- Mostrar colecciones: show collections

Tipos de Datos

Tipo	Descripción	Ejemplo
String	Es el tipo más común de almacenar datos. En MongoDB tiene formato UTF-8	{ "cadena" : "Mi cadena“}
Integer	Almacena valores numéricos. Según el servidor puede ser 32 bit o 64 bit	{"entero" : NumberLong(123456)}
Boolean	Booleano (true/ false)	{"boleano" : true}
Double	Valores con decimales	{"doble" : 50.5 }
Min/ Max keys	Se usa para comparar un valpr con el elemento BSON más bajo o más alto	{x:minKey}{x:maxKey}
Arrays	Almacena arrays, listas o valores multiples	{"miArray":[1,2,3]}
Timestamp	Útil para operaciones de creación y modificación	{timestamp: Date.now()}
Object	Tipo para documentos embebidos	{miObjeto:{tipo:”Object”, nombre:”Objeto”}}
Null	Valores Null	{prop:null}
Symbol	= String para lenguajes con tipos específicos de símbolos	{value:Symbol1}
Date	Almacena fecha o hora en formato UNIX. Se puede crear un objeto Date time propia a partir de Date()	{"fecha" : ISODate("2018-11-05T00:00:00Z")}
Object ID	Almacena el ID del document	{"_id" : ObjectId("5bce32e8203c473bac4a153e")}
Binary data	Para datos binaries	{"binario" : BinData(0,"1234") }
Code	Almacena Código JavaScript	{funcion: Date.now}
Regular expression	Expresiones regulares	{expresion1:new RegExp("%search")}

Insertar Documentos

- **db.miColeccion.insert({})**
  - Si no se especifica MongoDB asigna un ObjectId único con la siguiente configuración:

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id,
   3 bytes incrementer)
```

- Para inserta documentos múltiples en una única query se puede pasar un array de documentos: **db.miColeccion.insertMany([documento1,documento2,...])**
- También puede utilizarse **db.post.save(document)**. Si se pasa un _id que ya existe entonces reemplaza el documento.

Consultar Documentos

- Consultar documentos
  - **<db>.<colección>.find(<documento_filtro>)**
  - Ante estos documentos insertados:

```
{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
{ item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
```

  - `db.inventory.find({size: {h:14, w:21, uom:"cm"}})`
    - búsqueda exacta
  - `db.inventory.find({"size.uom": "in"})`
    - búsqueda exacta por campo JSON
  - `db.inventory.find({"size.h": { $lt:15 }})`
    - búsqueda por operador de comparación: documentos con size.h menor que 15

Consulta de Documentos con filtros de comparación

- Obtener todos los documentos de la colección: **db.COLLECTION_NAME.find()**

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":“Jared"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50
In	{ field: { $in: [<value1>, <value2> ... <valueN>]} }	db.inventory.find({ qty: { $in: [5, 15] } })	where qty in (5,15)
Not In	{ field: { $nin: [<value1>, <value2> ... <valueN>]} }	db.inventory.find({ qty: { $nin: [5, 15] } })	where qty not in(5,15)

Operadores lógicos

- **$and:** Pasando varias condiciones separadas por comas o utilizar:

```
>db.mycol.find(
   {
      $and: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

- **$or**:

```
>db.mycol.find(

   {

      $or: [

         {key1: value1}, {key2:value2}

      ]

   }

).pretty()
```

- **$not: { field: { $not: { <operator-expression> } } }**
- **$nor: db.inventory.find({ $nor: [{ price: 1.99 }, { sale: true }] })**

Elementos

- **$exists:** Exista una propiedad. Devuelve true|false

Ejemplo que exista la propiedad y no sea igual ni a 5 ni a 15:

```
db.inventory.find( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
```

- **$type:** Consulta por tipo de dato:

{ field: { $type: <BSON type> } }

```
db.addressBook.find( { "zipCode" : { $type : 2 } } );
db.addressBook.find( { "zipCode" : { $type : "string" } } );
```

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Deprecated.
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated.
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated.
JavaScript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Decimal128	19	“decimal”	New in version 3.4.
Min key	-1	“minKey”	
Max key	127	“maxKey”	

Array

- **$all**
  - Busca los arrays que contienen todos los elementos especificados en la query:

```
{ tags: { $all: [ "ssl" , "security" ] } }
```

- **$elemMatch**
  - Selecciona Documentos cuyos elementos del array cumplen todas las condiciones:

```
db.scores.find(
   { results: { $elemMatch: { $gte: 80, $lt: 85 } } }
)
```

- **$size**
  - Selecciona documentos con arrays de un tamaño concreto

Proyecciones

- Un **documento de proyección** especifica o restringe los campos que devuelve una consulta.

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

≈

```
SELECT _id, item, status from inventory WHERE status = "A"
```

**el _id del documento se obtiene siempre. Para eliminarlo de la consulta hay que explícitamente desactivarlo:*

```
db.inventory.find( { status: "A" }, { item: 1, status: 1, _id:0 } )
```

Proyecciones

- Un **documento de proyección** especifica o restringe los campos que devuelve una consulta.

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

≈

```
SELECT _id, item, status from inventory WHERE status = "A"
```

**el _id del documento se obtiene siempre. Para eliminarlo de la consulta hay que explícitamente desactivarlo:*

```
db.inventory.find( { status: "A" }, { item: 1, status: 1, _id:0 } )
```

Proyecciones

- Un **documento de proyección** especifica o restringe los campos que devuelve una consulta.

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

≈

```
SELECT _id, item, status from inventory WHERE status = "A"
```

**el _id del documento se obtiene siempre. Para eliminarlo de la consulta hay que explícitamente desactivarlo:*

```
db.inventory.find( { status: "A" }, { item: 1, status: 1, _id:0 } )
```

Métodos de cursor

- La consulta db.colección.find() devuelve un cursor que puede ser modificado por una serie de métodos:

Nombre	Descripción	Ejemplo
cursor.count()	Modifica el cursor para devolver el número de documentos en lugar de los documentos mismos	db.collection.find({ a: { $gt: 5 } }).count()
cursor.forEach()	Aplica una función JavaScript para todos los documentos de un cursor	db.users.find().forEach(function(myDoc) { print("user: " + myDoc.name); });
cursor.hasNext()	Devuelve true si el cursor tiene documentos sobre los que iterar	
cursor.limit()	Restringe el número de documentos que se obtienen de la consulta	db.user.find().limit(3)
cursor.map()	Devuelve los resultados en un array	{x:minKey}{x:maxKey}
cursor.next()	Devuelve el siguiente document	{"miArray":[1,2,3]}
cursor.pretty()	Configura un formato de fácil lectura	db.user.find().pretty()
cursor.size()	Devuelve el número de documentos despúes de aplicar los métodos skip() y limit()	db.user.find().skip(2).size()
cursor.skip()	Devuelve un cursor que comienza a devolver resultados después de pasar un número de documentos	db.suer.find().skip(3)
cursor.sort()	Devuelve los resultados ordenados (-1: desc, 1: asc)	db.user.find().sort({ age : -1, posts: 1 })

Agregaciones

- Se procesan los registros antes de devolver una salida. Equivalente a cláusulas GROUP BY o COUNT(*) en SQL:
  `db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)`
- Ej: carga desde terminal windows con la instrucción **mongoimport** el siguiente fichero media.mongodb.org/zips.json
  - Hacer una consulta para devolver los estados con una población de más de 10.000.000 hab.

```
db.zipcode.aggregate(
[
{$group:{_id:"$state",totalpop:{$sum:"$pop"}}},
{$match:{totalpop:{$gte:10*000*000}}}
]
)
```

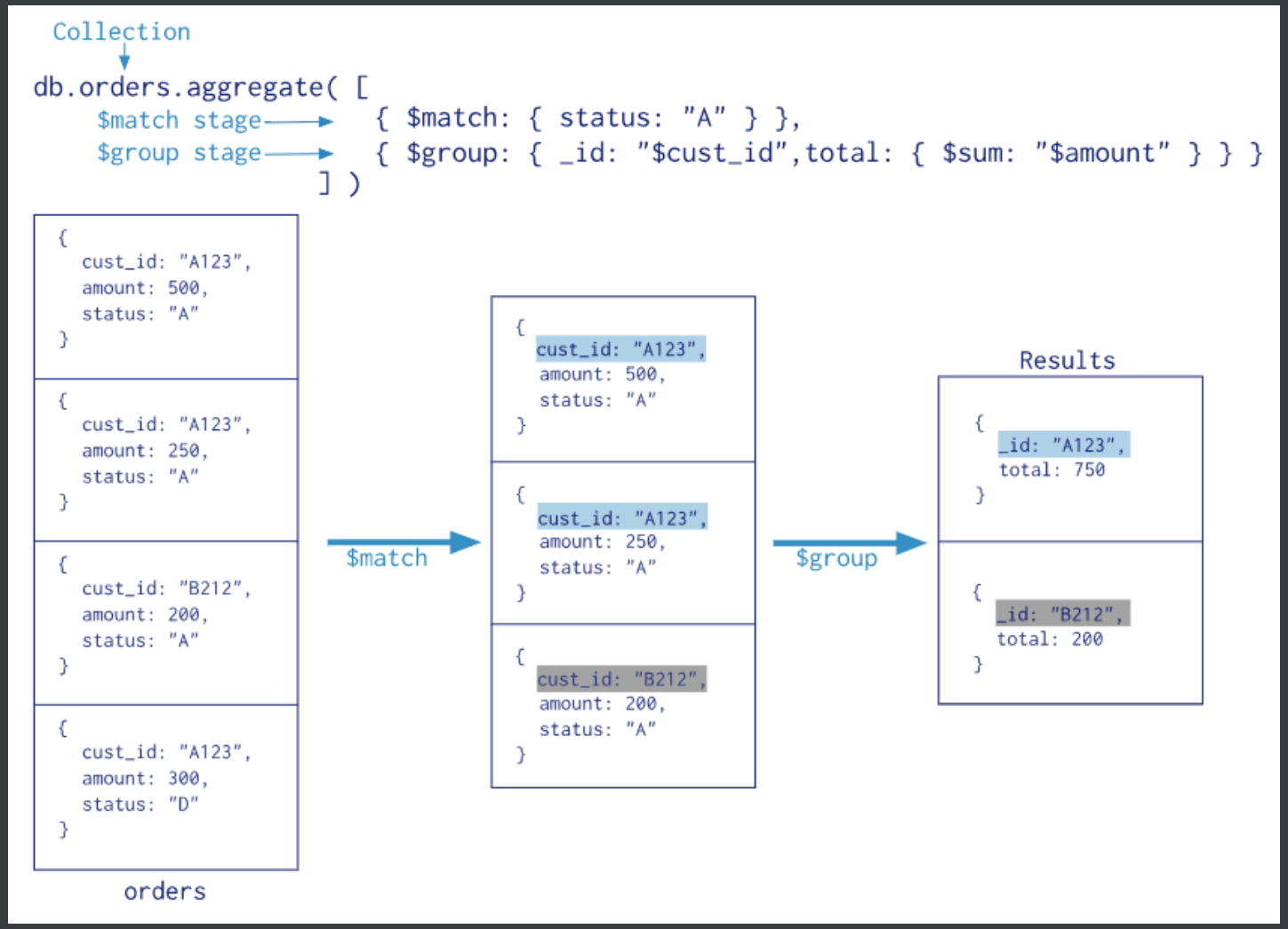
Pipeline - Aggregation

- MongoDB soporta el concepto de pipeline de la Shell de UNIX al utilizar el framework de agregación.
- Existen un conjunto de etapas posibles y cada una de ellas toma un conjunto de documentos como input y produce como resultado otro grupo de documentos (o un documento JSON al final del pipeline), que son el input de la etapa siguiente.
- Las posibles etapas en el framework aggregation son:
  - **$project** − Se utiliza para seleccionar campos específicos de una colección
  - **$match** − Operación de filtrado para reducir el número de documentos para la siguiente etapa
  - **$group** − Realiza la agregación como tal
  - **$sort** − Ordena los documentos
  - **$skip** − Salta n elementos
  - **$limit** − Limita la cantidad de documentos a partir de las posiciones actuales
  - **$unwind** − Descompone los documentos que utilizan arrays en tantos documentos como elementos tenga el array (normaliza el array)

Pipeline - Aggregation

- **Limitaciones de Aggregation framework**
- Existen algunas características que hay que tener en consideración.
  - **Hay un límite de 100mb de tamaño de pipeline para cada etapa** durante el procesado, a menos que establezcamos el atributo allowDiskUse en la ejecución, lo que lo hará más lento.
  - **Hay 16mb de límite de tamaño para la salida de cada pipeline**, ya que se maneja como un único documento. Se puede evitar obteniendo un cursor y recorriendo el cursor elemento a elemento.
  - **Sharding**: cuando se usa group o sort, todos los datos se reunen en un solo shard (el principal) antes de continuar procesando los pipes.

Agregaciones



Operadores de Agregación

$abs	$dayOfWeek	$last	$not	$stdDevSamp	$trim
$add	$dayOfYear	$let	$objectToArray	$strcasecmp	$trunc
$addToSet	$divide	$literal	$or	$strLenBytes	$type
$allElementsTrue	$eq	$ln	$pow	$strLenCP	$week
$and	$exp	$log	$push	$substr	$year
$anyElementTrue	$filter	$log10	$range	$substrBytes	$zip
$arrayElemAt	$first	$lt	$reduce	$substrCP	
$arrayToObject	$floor	$lte	$reverseArray	$subtract	
$avg	$gt	$ltrim	$rtrim	$sum	
$ceil	$gte	$map	$second	$switch	
$cmp	$hour	$max	$setDifference	$toBool	
$concat	$ifNull	$mergeObjects	$setEquals	$toDate	
$concatArrays	$in	$meta	$setIntersection	$toDecimal	
$cond	$indexOfArray	$min	$setIsSubset	$toDouble(aggregation)	
$convert	$indexOfBytes	$millisecond	$setUnion	$toInt	
$dateFromParts	$indexOfCP	$minute	$size	$toLong	
$dateToParts	$isArray	$mod	$slice	$toObjectId	
$dateFromString	$isoDayOfWeek	$month	$split	$toString	
$dateToString	$isoWeek	$multiply	$sqrt	$toLower	
$dayOfMonth	$isoWeekYear	$ne	$stdDevPop	$toUpper	

EJERCICIO 2

- Crear una bd “libreria” y cargar los scripts de github: https://github.com/rglepe/MeanStack/tree/master/Ejercicio2
  - Obtener todos los autores
  - Obtener los autores cuyo apellido sea DATE
  - Obtener los libros editados en 1998 o en 2005
  - Obtener el número de libros de la editorial Addison-Wesley
  - Obtener el libro que ocupa la tercera posición
  - Obtener la lista de autores de cada libro junto con el título
  - Obtener los títulos de libro publicados con posterioridad a 2004.
  - Obtener los libros editados en 1998 o en 2005
  - Obtener los libros editados desde 2001 y precio mayor que 50
  - Obtener los libros publicados por la editorial Addison-Wesley después de 2005.
  - Obtener el título de libro y editorial para aquellos libros que tengan un precio superior a 50€.
  - Obtener los libros publicados en 1998 o a partir de 2005.

Expresiones Regulares

- Se usan para buscar patrones en un string. MongoDB usa PCRE (Perl Compatible Regular Expression) como lenguaje de expresiones regulares
- Sintaxis: **{ <field>: { $regex: /pattern/, $options: '<options>' } }**
  { <field>: { $regex: 'pattern', $options: '<options>' } }
  { <field>: { $regex: /pattern/<options> } }

opciones	descripción
i	ignora mayúsculas y minúsculas
m	busca caracteres ancla en cada línea de un string multilínea (^ Primer carácter, $ último carácter)
x	ignora espacios en blanco
s	usa “.” como comodín de caracteres

EJERCICIO 3

- Obtener la población total de las ciudades cuyo nombre empieza por A o B
- Obtener la media de población de entre todas las ciudades de Nueva York y California cuya población supera los 25000 ciudadanos.
- Obtener los usuarios que han hecho más comentarios y los que han hecho menos de entre todos los posts de un blog. (tip: usar la cláusula unwind para crear un documento por cada comentario de cada post (que inicialmente estarían en un array dentro de el post correspondiente).