



Manual MongoDB

Proyecto Red Social

Francisco Morales Tapia

BERNAT EL FERRER

DAW 1R CURSO

Tabla de contenido

Introducción a MongoDB	2
¿Qué es MongoDB?	2
Que es una base de datos NoSQL.....	2
Tipos de bases de datos NoSQL	3
Ventajas y desventajas frente a una base de datos relacional	4
Operaciones CRUD	5
¿Qué es CRUD?	5
Operación de creación (Create)	5
Inserción de un solo dato	5
Inserción de múltiples datos	5
Operación de Lectura (Read)	6
Consulta básica	6
Consulta con criterios simples.....	6
Operación de Actualización (Update).....	8
Actualización de Documentos de un valor	8
Actualización de documentos de muchos valores	8
Operación de Eliminar (Delete)	9
Eliminación de Documentos de un valor.....	9
Eliminación de documentos de muchos valores	9

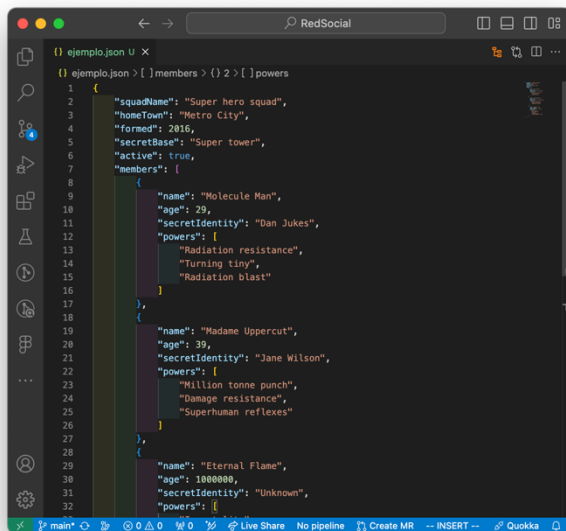
Introducción a MongoDB

¿Qué es MongoDB?

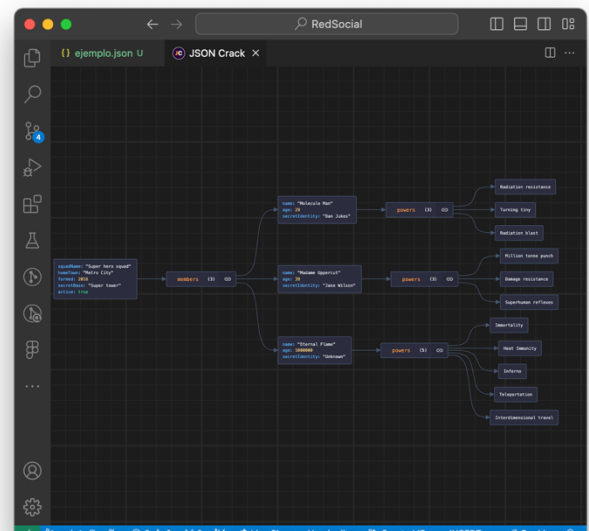
MongoDB es un gestor de base de datos NoSQL, que almacena los datos en un formato parecido al de JSON. Es un gestor de alto rendimiento, que permite la gestión de grandes cantidades de datos. Se suele utilizar en proyectos de desarrollo web, big data, análisis de datos entre otros tipos de ámbitos de uso.

Que es una base de datos NoSQL

Una de datos NoSQL es una base de datos que no es necesaria una estructura de datos que se basa en tablas columnas y filas, sino que este se diferencia por el tipo de ordenación y forma de documentos. Estas bases de datos se asemejan a la estructura de un archivo JSON



```
1 {} ejemplo.json U X
2 {} ejemplo.json > [ ] members > [ ] 2 > [ ] powers
3 {
4   "squadName": "Super hero squad",
5   "homeTown": "Metro City",
6   "formed": 2016,
7   "secretBase": "Super tower",
8   "active": true,
9   "members": [
10    {
11      "name": "Molecule Man",
12      "age": 29,
13      "secretIdentity": "Dan Jukes",
14      "powers": [
15        "Radiation resistance",
16        "Turning tiny",
17        "Radiation blast"
18      ]
19    },
20    {
21      "name": "Madame Uppercut",
22      "age": 39,
23      "secretIdentity": "Jane Wilson",
24      "powers": [
25        "Million tonne punch",
26        "Damage resistance",
27        "Superhuman reflexes"
28      ]
29    },
30    {
31      "name": "Eternal Flame",
32      "age": 1000000,
33      "secretIdentity": "Unknown",
34      "powers": [
35        "Heat vision",
36        "Invisibility",
37        "Firebreath"
38      ]
39    }
40  ]
41 }
```



Tipos de bases de datos NoSQL

Existen varios tipos de bases de datos NoSQL, aunque las más populares utilizan un gestor de archivos tipo JSON, aquí explicaremos un poco sobre los ejemplos de NoSQL:

Bases de datos de documentos

Las bases de datos de documentos almacenan datos en colecciones, siendo los elementos de cada una de esas colecciones documentos que contienen pares clave/valor.

Es como un sistema que nos permite almacenar elementos especificados por el lenguaje JSON. Cada documento puede tener una estructura diferente, como hemos señalado antes, lo que facilita la adaptación a cambios en el esquema de datos incluso entre documentos que pertenecen a una misma colección.

Bases de datos tabulares

Las bases de datos tabulares son más conocidas en el sistema de bases de datos del modelo relacional, pero también las encontramos en el modelo NoSQL, aunque en menor medida. Este modelo permite el almacenamiento en tablas, aunque admiten mayor versatilidad, ya que permite variar el formato de los registros en ellas.

Bases de datos orientadas a grafo

Este modelo de bases de datos NoSQL está compuesto por documentos que se relacionan entre sí y que permiten representar relaciones complejas entre los datos. Son ideales para aplicaciones que requieren un alto nivel de conectividad como redes sociales, bases de conocimiento o inteligencia artificial.

Bases de datos de clave-valor

Las bases de datos NoSQL denominadas clave-valor son las más sencillas de todas y almacenan datos en una organización de pares de clave y valor. Son las que permiten más altas velocidades en la lectura y escritura de datos y se usan principalmente en sistemas de almacenamiento en caché.

Bases de datos multivalor

Las bases de datos multivalor permiten trabajar también con pares clave/valor, solo que permiten almacenar múltiples valores asociados a una clave. Esto puede hacerlas útiles para aplicaciones que gestionan datos complejos y variables.

Ventajas y desventajas frente a una base de datos relacional

Ventajas

- **Flexibilidad:** Al ser una base de datos que no está conectada entre sí, es más fácil la adaptabilidad de datos según sus cambios constantes.
- **Alta gestión de BigData:** esta nos permite tener una. Mayor optimización y grandes consultas de una gran cantidad de datos, a pesar de que otros aplicativos no estén adaptadas.
- **Rendimiento:** Las bases de datos noSql están optimizadas para tareas específicas por lo que puede ofrecer un mayor. Rendimiento en cuanto a consultas.
- **Alta adaptabilidad:** Estas bases de datos pueden ser de gran recomendación para tecnologías que se ejecutan en tiempo real, IoT (Internet of Things), entre otras.

Desventajas:

- **Madurez:** Al no ser un sistema con años en el mercado, no se han realizado exhaustivos recursos de aprendizaje y soporte.
- **Sintaxis de consultas:** Para los que manejas base de datos SQL al realizar el cambio a NoSQL puede ser complejo de entender al principio.
- **Integración de datos:** Al no tener una relación directa colecciones, estas pueden tener una inconsistencia de datos.

Operaciones CRUD

¿Qué es CRUD?

Las operaciones CRUD son las operaciones básicas que nos permitirá poder manejar la base de datos, ya sea relacional como no relacional. Las operaciones CRUD se dividen en 4 siglas que son:

- Create (Creación)
- Read (Lectura)
- Update (Actualización)
- Delete (Eliminación)

Con estas 4 operaciones nos permitirá la gestión de una base de datos.

Operación de creación (Create)

Inserción de un solo dato

Para poder insertar un solo dato, tan solo deberemos de usar la sentencia:

- `db.collection.insertOne({})`

De esta forma podremos insertar un solo dato dentro de nuestra colección, creada y seleccionada previamente. En el momento de insertar un dato dentro de una colección, si esa colección existiera, insertaría el dato dentro en cambio si no existiera, esta crearía una nueva colección con el nombre que le hayamos introducido.

Ejemplo de inserción de datos a una colección de usuario:

- `db.usuarios.insertOne({nombre: "Sara", apellido: "Prado", edad: 24, mail: "sara_p@gmail.com"})`

Inserción de múltiples datos

Para la inserción de múltiples datos deberemos de realizar una sentencia similar a la de 'insertOne', la sintaxis es similar, varía en la inserción que cambia a 'insertMany([])', donde dentro del paréntesis introduciremos todos los datos.

- `db.usuarios.insertMany([
 {nombre: "Sara", apellido: "Prado", edad: 24, mail: "sara_p@gmail.com"},
 {nombre: "Jake", apellido: "Sully", edad: 24, mail: "sara_p@gmail.com"}])`

Operación de Lectura (Read)

Consulta básica

Para realizar una consulta básica deberemos de tener seleccionada la base de datos, para poder seleccionar una base de datos deberemos de escribir:

- `use db` → 'db' se refiere a la base de datos

Una vez seleccionada ya podremos realizar las consultas. Para realizar una consulta en una colección podemos usar:

- `db.collection.find()` → De esta forma podemos realizar una consulta que nos devolverá todos los datos de la colección.

Consulta con criterios simples

Si queremos realizar una consulta que filtre por la edad deberemos de realizar la consulta 'find()', dentro de la consulta deberemos de introducir los datos como si

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

Operaciones de consulta (comparación, lógica, elementos de un array)

En una consulta podemos filtrar el resultado sus operadores o su lógica de comparación, para la concatenación de filtrado de datos lo separaremos con ',' y sus operadores se dividen en:

COMPARACIÓN

Operador	Definición	Implementación	Expresado en SQL
<code>\$lt</code>	Menor que (<)	edad: { <code>\$lt</code> : 18}	SELECT * FROM PERSONA WHERE EDAD < 18;
<code>\$gt</code>	Mayor que (>)	Edad: { <code>\$gt</code> : 18}	SELECT * FROM PERSONA WHERE EDAD > 18;
<code>\$gte</code> <code>\$lte</code>	<code><=</code> <code>>=</code>	Edad: { <code>\$gte</code> : 18} Edad: { <code>\$lte</code> : 18}	SELECT * FROM PERSONA WHERE EDAD=18 AND EDAD<=18;
<code>\$eq</code>	equals	Edad: { <code>\$eq</code> : 20}	SELECT * FROM PERSONA WHERE EDAD=20;
<code>\$in</code>	in	Edad: { <code>\$in</code> [18, 20, 22]}	SELECT * FROM PERSONA WHERE EDAD IN (18, 20, 22);
<code>\$ne</code>	Not equals	Edad: { <code>\$ne</code> : 20}	SELECT * FROM PERSONA WHERE EDAD != 20;

LOGICA

Operdor	Definicion	Implementacion	Expresado en SQL
\$and	and	\$and: [{edad: {\$eq: 25}}, {\$altura: {\$eq: 1.87}}]	SELECT * FROM PERSONA WHERE EDAD = 25 AND ALTURA = 1.87;
\$not	not	\$not: {edad: {\$lt: 18}}	SELECT * FROM PERSONA WHERE EDAD NOT < 18;
\$nor	Not or	\$nor: [{edad: {\$eq: 25}}, {\$altura: {\$eq: 1.87}}]	SELECT * FROM PERSONA WHERE EDAD = 25 AND ALTURA = 1.87;
\$or	or	\$or: [{nombre: {\$eq: "Juan"}}, {nombre: {\$eq: "Jorje"}}]	SELECT * FROM PERSONA WHERE NOMBRE = "Juan" OR NOMBRE = "Jorje";

ARRAY

Operdor	Definicion	Implementacion
\$all	Matches arrays that contain all elements specified in the query.	db.articles.find({ tags: { \$all: [["ssl", "security"]] } })
\$elemMatch	Selects documents if element in the array field matches all the specified \$elemMatch conditions.	db.scores.find({ results: { \$elemMatch: { \$gte: 80, \$lt: 85 } } })
\$size	Selects documents if the array field is a specified size.	db.collection.find({ field: { \$size: 2 } });

Operación de Actualización (Update)

Para poder realizar un update debemos de introducir una de las tres opciones, según el tipo de update que realicemos actualizaremos un solo campo de la colección o múltiples datos de la colección.

- `db.collection.updateOne()` *New in version 3.2*
- `db.collection.updateMany()` *New in version 3.2*
- `db.collection.replaceOne()` *New in version 3.2*

A continuación mostramos un ejemplo sobre como realizar un update sobre la colección de 'users' en donde se cambiara el 'status' del 'user' con edad menor que 18 hacia "reject".

```
db.users.updateMany(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } }  
)
```



Actualización de Documentos de un valor

Para poder realizar un update de un solo campo, deberemos de utilizar el 'updateOne()', de esta forma podremos modificar los campos de una sola fila. En este caso se actualizarán los datos de la colección 'inventory' que contenga el ítem "papel", este reemplaza los datos por los que les proporcionemos. En caso de que existieran más de un campo con el ítem "papel" este solo actualizara el primer campo que coincida con el filtrado.

```
db.inventory.updateOne(  
  { item: "paper" },  
  {  
    $set: { "size.uom": "cm", status: "P" },  
    $currentDate: { lastModified: true }  
  })
```

Actualización de documentos de muchos valores

De esta forma actualizaremos todos los campos que contengan un "qty" menor que 50

```
db.inventory.updateMany(  
  { "qty": { $lt: 50 } },  
  {  
    $set: { "size.uom": "in", status: "P" },  
    $currentDate: { lastModified: true }  
  })
```


Operación de Eliminar (Delete)

Con delete podemos realizar la operación de eliminar registros filtrando los datos para eliminar los datos necesarios. En MongoDB existen dos formas de eliminar registros:

- `db.collection.deleteOne()` *New in version 3.2*
- `db.collection.deleteMany()` *New in version 3.2*

Para realizar una operación que elimine registros debemos de utilizar una de las opciones anteriores según el caso en que eliminemos los datos, la sintaxis es la siguiente.

```
db.users.deleteMany(  
  { status: "reject" }  
)
```



Eliminación de Documentos de un valor

En el caso de que queramos eliminar un registro que coincida con el filtrado que hemos introducido deberemos de usar la siguiente instrucción:

```
db.inventory.deleteOne( { status: "D" } )
```

Eliminación de documentos de muchos valores

Si deseamos realizar un borrado completo de una colección, debemos de utilizar la siguiente instrucción:

```
db.collection.deleteMany({})
```

En el caso de que deseemos eliminar múltiples filas deberemos de utilizar la consulta anterior extendiendo con el filtrado necesario.

```
db.inventory.deleteMany({ status : "A" })
```