UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

LABORATORIO DE COMPUTACIÓN II

Unidad Temática N° 4:
Bases de Datos NoSQL

Teórico
1° Año – 2° Cuatrimestre

Índice

INTRODUCCIÓN

Hasta ahora se venía hablando sobre las Base de Datos relacionales, pero también existen aquellas que no lo son. Reciben el nombre de NoSQL (Not only SQL) y su término comenzó a utilizarse en el año 1998, para denominar una base de datos relacional que no utilizaba el lenguaje SQL para funcionar.

A medida que las necesidades fueron cambiando el concepto de NoSQL se hizo más fuerte, ya que las bases de datos no podían quedarse estancadas en un modelo que no respondía a problemas que se estaban presentando.

Por lo tanto, se puede decir que el término NoSQL hace referencia al modelo de bases de datos que busca alternativas al sistema de bases de datos relacionales, el cual, se destaca por priorizar el acceso rápido a la información por sobre la integridad de los datos. Las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad–relación, además, en general, no utilizan SQL para la manipulación de datos.

¿Por qué aparecen las Bases de Datos NoSQL?

Los volúmenes de información están creciendo a niveles exponenciales, y la manera en la que se los administra se hace cada vez más compleja. Tanto los usuarios como las empresas desean sacarle el máximo provecho, obligando a que se considere un nuevo tipo de arquitectura que sea capaz de soportar los nuevos requerimientos que se originan en base a los siguientes aspectos:

- El gran aumento del tamaño de los archivos, junto a la cantidad de los mismos.
- La velocidad de respuesta ante las consultas simultánea de millones de usuarios
- Sistemas descentralizados

Ventajas

Las principales ventajas que encontramos a la hora de trabajar con Base de Datos NoSQL son las siguientes:

- **Evitar la complejidad innecesaria**: Con el objetivo de brindar una correcta consistencia de los datos, las Bases de datos relacionales implementan una gran cantidad de restricciones innecesarias.

- **Alto rendimiento:** Muchas bases de datos NoSQL proporcionan un rendimiento superior a los que brindan los RDBS.
- **Escalabilidad horizontal y hardware básico:** A diferencia de las RDBS, la mayoría de las bases de datos no relacionales se diseñaron para escalar horizontalmente y no depender de hardware costoso. Se pueden agregar o eliminar máquinas de forma sencilla sin tener que hacer un gran esfuerzo operativo.
- **No es necesario llevar a cabo el proceso de mapeado:** La mayoría de las bases de datos NoSQL están diseñadas para almacenar estructuras de datos que son simples o más similares a las de los lenguajes de programación orientados a objetos en comparación con las estructuras de datos relacionales.
- **Comprometer la fiabilidad para lograr un mejor desempeño:** En algunos momentos se puede hacer necesario conseguir un mejor desempeño a cambio de que se deje de lado la fiabilidad.
- **Facilidad para descentralizar la base de datos**: Los modelos de datos originalmente diseñados con una sola base de datos en mente (los modelos de datos centralizados) a menudo no se pueden dividir y distribuir fácilmente entre servidores de bases de datos.
- **Consultas simples**: las consultas requieren menos operaciones y son más naturales, por lo tanto, se gana en simplicidad y eficiencia.

Principales diferencias con las BD Relacionales

Uno de los conceptos fundamentales en los que se basan las Bases de Datos NoSQL es el Teorema de CAP,

El teorema de CAP dice que, en un sistema de base de datos distribuida, puede tener como máximo sólo dos de las tres características, en donde se puede tener consistencia, disponibilidad (Availability) y tolerancia a particiones (Partition Tolerance).

Consistencia: *"La consistencia significa que cada usuario de la base de datos tiene una vista idéntica de los datos en un instante dado." (Harrison, 2015)*

Disponibilidad (Availability): "*Disponibilidad significa que, en caso de falla, la base de datos permanece operativa." (Harrison, 2015)*

Tolerancia a particiones (Partition Tolerance): *"La tolerancia a la partición significa que la base de datos puede mantener las operaciones en caso de que la red falle entre dos segmentos del sistema distribuido." (Harrison, 2015)*

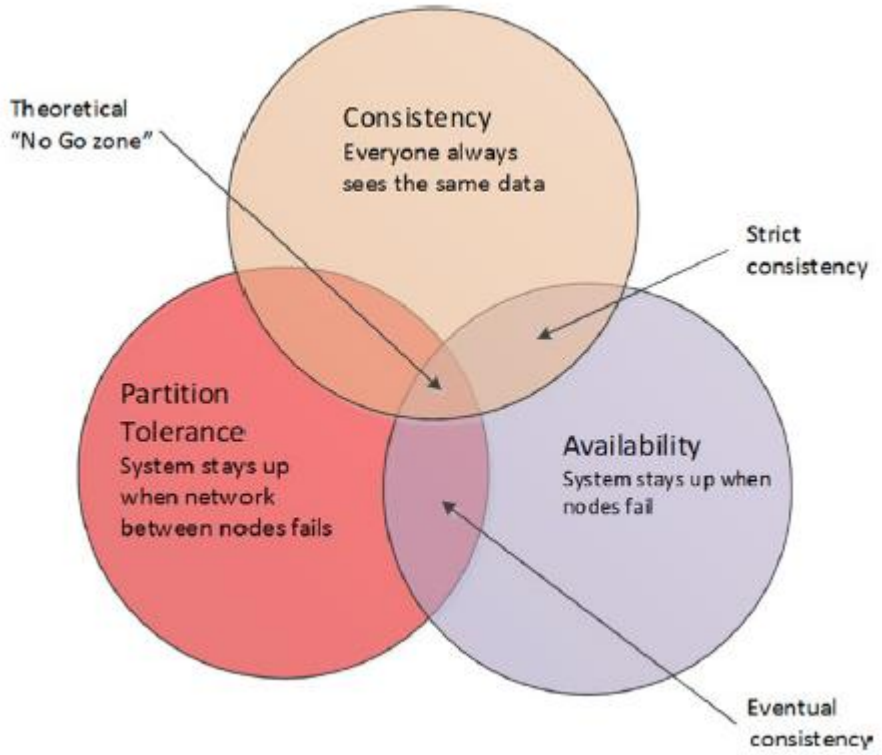

Ilustración 1: Recuperado de Harrison, 2015

- Sistemas coherentes y disponibles (CA), pero con dificultades para funcionar en caso de que haya muchas particiones.
- Sistemas coherentes y tolerantes a particiones (CP), pero con ciertas carencias en temas de disponibilidad.
- Sistemas disponibles y tolerantes a particiones (AP), pero no estrictamente coherente.

Las transacciones en SQL están totalmente ligadas con las propiedades ACID. Es decir, que las transacciones garantizan la Atomicidad, la Consistencia, el aislamiento (Isolation) y la Durabilidad. Sin embargo, en NoSQL con el objetivo de soportar los nuevos requerimientos, algunos aspectos tuvieron que ser sacrificados o modificados.

- Atomicidad (Atomicity): "La transacción es indivisible, o todas las declaraciones de la transacción se aplican a la base de datos o no se aplica

ninguna, por lo tanto, la transacción no puede quedar a medias.” (Harrison, 2015)

- Consistencia (Consistency): “La base de datos permanece en un estado consistente antes y después de la ejecución de la transacción.” (Harrison, 2015)
- Aislado (Isolation): “Si bien uno o más usuarios pueden ejecutar múltiples transacciones simultáneamente, una transacción no debería ver los efectos de otras transacciones en curso.” (Harrison, 2015)
- Durabilidad (Durability): “Una vez que una transacción se guarda en la base de datos, se espera que sus cambios persistan incluso si hay una falla en el sistema operativo o hardware.” (Guy Harrison, 2015)

Por ejemplo, en el caso de la consistencia, en las bases de datos relaciones mantener la consistencia total del sistema hace a estos más lentos, ya que se implementan un conjunto de restricciones para que, por ejemplo, dos usuarios no puedan modificar un archivo al mismo tiempo; en los sistemas NoSQL se maneja una consistencia eventual, que permite a cada sistema hacer cambios a los datos.

La estructura y el orden para el almacenamiento de los datos que brinda una base de datos relacional se dejan de lado, haciendo uso de otros tipos de modelos de almacenamiento de información que luego se verá.

Ya que se maneja un gran volumen de datos, no se suelen permitir operaciones JOIN, ya que las mismas implican una sobrecarga.

Tipos de BD NoSQL

Aunque existen muchas tecnologías en bases de datos NoSQL, se van a destacar las cuatro principales:

- **Base de datos clave - valor:** Este tipo de base de datos posee la estructura de datos más sencilla, la información queda estructurada emparejando claves con valores.

  Ejemplos de este tipo de base de datos NoSQL de clave-valor son Cassandra, Redis, DynamoDB.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/03/2021
K5	25,MMM,2544

Tabla 1: Elaboración propia

- **Base de datos documentales**: Las bases de datos documentales son otro tipo de base de datos NoSQL con un grado de complejidad y flexibilidad superior a las bases de datos clave – valor.

  Un documento contiene un campo de identificación único y valores que pueden ser de una variedad de tipos. Los documentos pueden contener estructuras anidadas.

  Ejemplos de este tipo son MongoDB, ArangoDB, CouchDB.

- **Base de datos en grafo:** La información viene representada por los nodos, y las relaciones entre los datos por las aristas.

  Un grafo es un conjunto de nodos que mantienen una serie de relaciones entre sí. Las relaciones tienen nombre y sentido, y siempre tienen un nodo de inicio y uno de fin.

  Un ejemplo de este tipo de BD es Neo4j

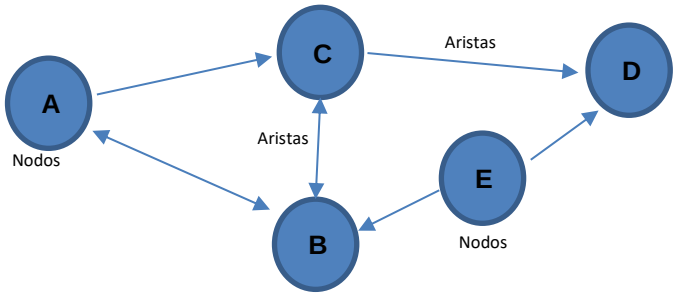

Gráfico 1: Elaboración propia

- **Base de datos Orientada a Objetos**: En las bases de datos orientadas a objetos la información se representa como objetos. Esta forma de representar la información es similar a la que se utiliza en la POO.
- **Bases de datos en columnas**: Una base de datos orientada a columnas almacena su información en columnas, en lugar de filas.

Un ejemplo es HBase.

¿Qué es MongoDB?

"MongoDB es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado."

"MongoDB almacena datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse con el tiempo"

Un registro en MongoDB es un documento, ¿pero esto qué significa?

"Un documento es una estructura de datos compuesta de pares de campos y valores. Los documentos en MongoDB son muy similares a lo que serían objetos en notación JSON, solo que aquí es un esquema dinámico denominado BSON. Los valores de los campos pueden incluir otros documentos, matrices (arrays) y matrices de documentos."

Las ventajas de utilizar documentos son:

- Los documentos corresponden a tipos de datos nativos en muchos lenguajes de programación.
- Los documentos y matrices embebidos reducen la necesidad de realizar joins, los cuales son muy pesados.
- Los esquemas dinámicos admiten polimorfismo con total fluidez."

MongoDB almacena documentos en colecciones. Las colecciones son análogas a las tablas en las bases de datos relacionales.

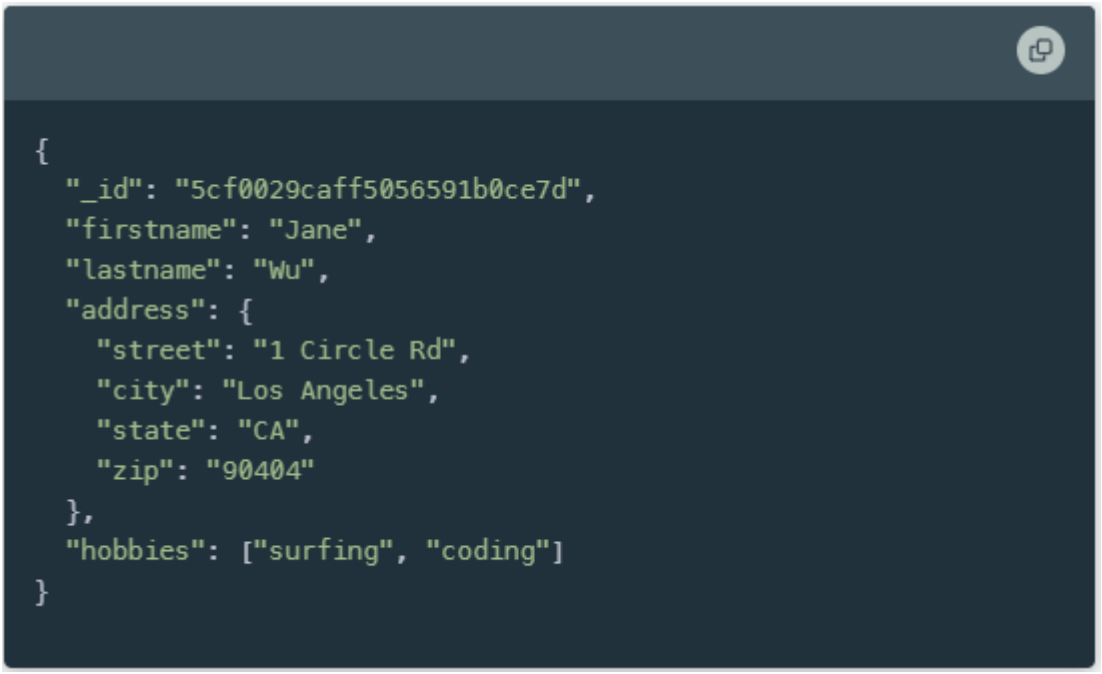

Imagen 1: ***Recuperado de la web de MongoDB***

Las claves son cadenas, pero existen un conjunto de restricciones. Una clave dentro de un documento debe ser única y no puede contener un carácter nulo, además, se deben evitar el uso de puntos y el símbolo dólar($), ya que dentro del motor estos son caracteres especiales.

MongoDB es *case sensitive* (distingue mayúsculas y misnúsculas), y el orden en el que están definidos los pares de valores determinan un documento de otro, es decir que si tenemos el documento {“x” : 1, “y” : 2}, esto va a ser diferente de otro documento {“y” : 2, “x” : 1}.

En cada par nombre-valor, el nombre debe ir rodeado de comillas dobles, aunque la gran mayoría de lenguajes de programación permiten que no se utilicen, excepto que el nombre incluya espacios, puntos o empiece por algo que no sea una letra.

Los tipos de datos que tenemos en MongoDB son:

- Nulo
- Boolean
- Número (donde tenemos números enteros y decimales)
- Cadena de caracteres (string)
- Arrays
- Documentos
- Fechas

- Expresiones regulares
- Identificadores de objetos
- Datos Binarios
- Código JavaScript

Como se había expresado en párrafos presedentes, una colección es un conjunto de documentos, en donde las colecciones son análogas a las tablas en las bases de datos relacionales.

"De forma predeterminada, una colección no requiere que sus documentos tengan el mismo esquema; es decir, los documentos de una sola colección no necesitan tener el mismo conjunto de campos y el tipo de datos de un campo puede diferir entre los documentos de una colección." (MongoDB - https://docs.mongodb.com/manual/core/databases-and-collections/)

A pesar de esto, es una buena práctica que una colección agrupe documentos muy similares, en donde cada uno de ellos contiene cierta relación.

Dichas colecciones se agrupan en una base de datos. Entonces, se podría decir que una base de datos es un grupo de colecciones.

Haciendo una analogía con respecto a las bases de datos relaciones, las colecciones serían las tablas, mientras que las filas los documentos, ya que los documentos forman parte de las colecciones y las colecciones se almacenan en las Bases de Datos.

"En MongoDB, las bases de datos contienen una o más colecciones de documentos. Para seleccionar una base de datos para usar, en el shell mongo, emita la instrucción 'use <db>' "(MongoDB - https://docs.mongodb.com/manual/core/databases-and-collections/)

```
>>> use test
switched to db test
```

Imagen 2: Elaboración propia

Mientras que para visualizar todas las bases de datos, se debe utilizar la instrucción 'show dbs'

```
>>> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Imagen 3: Elaboración propia

Dentro del shell, si se coloca la instrucción ‘db’ se mostrará la base de datos actual.

```
>>> db
test
```

Imagen 4: Elaboración propia

No es necesario crear la base de datos antes de cambiar de BD. MongoDB crea la base de datos cuando almacena datos por primera vez en ella.

Por lo tanto, si se desea utilizar una base de datos llamada laboratorio (que aún no existe), la instrucción sería ‘use laboratorio’, pero esto no genera la base de datos, la generación viene dada una vez que le se crea una colección en ella.

Para insertar una colección y de esta manera crear la bd, la instrucción es ‘db.collection.insert()’. Dentro de los paréntesis se debe enviar como parámetro el documento que se le quiere insertar a esa colección.

Si se quiere crear la colección alumnos, y dentro de la misma insertar un documento, la instrucción sería la siguiente:

db.alumnos.insert({“nombre”:“Matias”, “legajo”:123})

```
>>> db.alumnos.insert({"nombre":"Matias","legajo":123})
WriteResult({ "nInserted" : 1 })
```

Imagen 5: Elaboración propia

Cabe aclarar que MongoDB brinda 2 métodos diferentes de inserción, los cuales son:

- db.collection.insertOne()
- db.collection.insertMany()

Insertar un solo documento

Para insertar un solo documento utilizamos la sentencia "db.collection.insertOne()"

El siguiente ejemplo inserta un nuevo documento en la colección de alumnos. Si el documento no especifica un campo _id, MongoDB agrega el campo _id con un valor ObjectId al nuevo documento.

```
>>> db.alumnos.insertOne({"nombre":"matias","legajo":123})
{
    "acknowledged" : true,
    "insertedId" : ObjectId("604d2639bbe6c6160939d7e4")
}
```

Imagen 6: Elaboración propia

Como se puede observar insertOne() devuelve un documento que incluye el valor del campo _id del documento recién insertado.

Insertar múltiples documentos

"Db.collection.insertMany()" permite insertar varios documentos en una colección. Se debe pasar una serie de documentos a través del método.

En el siguiente ejemplo, al igual que sucedió en el anterior, si no se especifica un campo _id, MongoDB agrega el campo _id automáticamente.

```
>>> db.alumnos.insertMany([{"alumnos":"Matias", "legajo":123},
{"alumnos":"Nicolas","legajo":1234}])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("604d2a8a759006f14714a54c"),
        ObjectId("604d2a8a759006f14714a54d")
    ]
}
```

Imagen 7: Elaboración propia

Operaciones de lectura

Las operaciones de lectura recuperan documentos de una colección. MongoDB proporciona el método “db.<collection>.find()” para leer los documentos de una colección. Dentro del método se pueden especificar diferentes filtros de consulta.

En el siguiente ejemplo, se utiliza el método “db.alumnos.find({})” para obtener todos los documentos de la colección alumnos.

```
>>> db.alumnos.find({})
{ "_id" : ObjectId("604d2df5b7cf5b788008f227"), "nombre" : "Matias", "legajo" : 123 }
{ "_id" : ObjectId("604d2e2cb7cf5b788008f228"), "nombre" : "Martin", "legajo" : 1234 }
{ "_id" : ObjectId("604d2e2cb7cf5b788008f229"), "nombre" : "Nicolas", "legajo" : 12345
}
```

Imagen 8: Elaboración propia

Si queremos obtener un documento en particular, especificamos dentro del método el campo y el valor del mismo.

```
>>> db.alumnos.find({"legajo":123})
{ "_id" : ObjectId("604d2df5b7cf5b788008f227"), "nombre" : "Matias", "legajo" : 123 }
```

Imagen 9: Elaboración propia

Para modificar el formato del retorno, agregue “.pretty()” a la operación de búsqueda:

```
>>> db.alumnos.find({}).pretty()
{
    "_id" : ObjectId("604d2df5b7cf5b788008f227"),
    "nombre" : "Matias",
    "legajo" : 123
}
{
    "_id" : ObjectId("604d2e2cb7cf5b788008f228"),
    "nombre" : "Martin",
    "legajo" : 1234
}
{
    "_id" : ObjectId("604d2e2cb7cf5b788008f229"),
    "nombre" : "Nicolas",
    "legajo" : 12345
}
```

Imagen 10: Elaboración propia

Operaciones de actualización

Las operaciones de actualización modifican los documentos existentes en una colección. MongoDB proporciona los siguientes métodos para actualizar documentos de una colección

- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

En MongoDB, las operaciones de actualización tienen como objetivo una sola colección. Puede especificar criterios o filtros que identifican los documentos a actualizar. Estos filtros utilizan la misma sintaxis que las operaciones de lectura.

Para actualizar un documento, MongoDB proporciona operadores de actualización, como es el caso de "$set", que se utiliza para modificar los valores de los campos.

Operadores de actualización

Los siguientes modificadores están disponibles para su uso en operaciones de actualización

Campos

Nombre	Descripción
$currentDate	Establece el valor de un campo a la fecha actual, ya sea como Fecha o Marca de tiempo.
$inc	Incrementa el valor del campo en la cantidad especificada.
$min	Solo actualiza el campo si el valor especificado es menor que el valor del campo existente.
$max	Solo actualiza el campo si el valor especificado es mayor que el valor del campo existente.
$mul	Multiplica el valor del campo por la cantidad especificada.
$rename	Cambia el nombre de un campo.
$set	Establece el valor de un campo en un documento.

Nombre	Descripción
$setOnInsert	Establece el valor de un campo si una actualización da como resultado la inserción de un documento. No tiene ningún efecto sobre las operaciones de actualización que modifican documentos existentes.
$unset	Elimina el campo especificado de un documento.

Tabla 2: Elaboración propia

Array

Nombre	Descripción
$	Elimina todos los valores coincidentes de una matriz.
$[]	Actúa como un marcador de posición para actualizar todos los elementos de una matriz para los documentos que coinciden con la condición de la consulta.
$[<identifier>]	Actúa como un marcador de posición para actualizar todos los elementos que coinciden con la condición arrayFilters para los documentos que coinciden con la condición de la consulta.
$addToSet	Agrega elementos a una matriz sólo si aún no existen en el conjunto.
$pop	Elimina el primer o último elemento de una matriz.
$pull	Elimina todos los elementos de la matriz que coinciden con una consulta específica.
$push	Agrega un elemento a una matriz.
$pullAll	Elimina todos los valores coincidentes de una matriz.

Tabla 3: Elaboración propia

Actualizar un solo documento

El siguiente ejemplo usa el método db.collection.updateOne () en la colección alumnos para actualizar el primer documento donde el legajo es igual a 1234:

```
>>> db.alumnos.updateOne({"legajo":1234},{$set:{"nombre":"Lucas"}})
```

Imagen 11: Elaboración propia

Actualizar múltiples documentos

El siguiente ejemplo usa el método db.collection.updateMany () en la colección de users para actualizar todos los documentos donde la edad es menor que 18:

Imagen 12: ***Recuperado de la web de MongoDB***

Reemplazar documentos

Para reemplazar todo el contenido de un documento excepto el campo _id, pase un documento completamente nuevo como segundo argumento a db.collection.replaceOne ().

Al reemplazar un documento, el documento de reemplazo debe constar únicamente de pares de campo / valor; es decir, no se debe incluir las expresiones de los operadores de actualización.

El documento de reemplazo puede tener diferentes campos del documento original. En el documento de reemplazo, puede omitir el campo _id ya que el campo _id es inmutable; sin embargo, si incluye el campo _id, debe tener el mismo valor que el valor actual.

Eliminar documentos

Las operaciones de eliminación eliminan documentos de una colección. MongoDB proporciona los siguientes métodos para eliminar documentos de una colección:

- -db.collection.deleteOne()
- -db.collection.deleteMany()

En MongoDB, las operaciones de eliminación tienen como objetivo una sola colección. Puede especificar criterios o filtros que identifiquen los documentos que se eliminarán. Estos filtros utilizan la misma sintaxis que las operaciones de lectura.

Eliminar todos los documentos

Para eliminar todos los documentos de una colección, pase un documento de filtro vacío {} al método db.collection.deleteMany ().

En el siguiente ejemplo se eliminan todos los documentos de la colección alumnos. El método devuelve un documento con el estado de la operación.

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

Imagen 13: Elaboración propia

Eliminar todos los documento que cumplan cierta condición

Para eliminar todos los documentos que coincidan con un criterio de eliminación, pase un parámetro de filtro al método deleteMany ().

En el siguiente ejemplo se eliminan todos los documentos de la colección alumnos en donde el legajo sea 123.

```
>>> db.alumnos.deleteMany({"legajo":123})
{ "acknowledged" : true, "deletedCount" : 1 }
```

Imagen 14: Elaboración propia

Eliminar un documento que cumpla cierta condición

Para eliminar como máximo un solo documento que coincida con un filtro especificado (aunque varios documentos pueden coincidir con el filtro especificado) utilice el método db.collection.deleteOne ().

BIBLIOGRAFÍA

Chodoro K (2013) MongoDB the definite Guide.

Harrison G. (2015) Next Generation Databases - NoSQL, NewSQL and Big Data.

MongoDB (2021) Recuperado de: https://docs.mongodb.com/

MongoDB (2021) Recuperado de: https://docs.mongodb.com/manual/

MongoDB (2021) Recuperado de: https://docs.mongodb.com/manual/tutorial/getting-started/

MongoDB (2021) Recuperado de: https://docs.mongodb.com/manual/reference/ /operator/update/

Redmon E. & Wilson J. (2012) Seven Database in Seven Weeks- A guide to modern Databases and the NoSQL movement