

Manual de mongoDb

Evaluación M02



21 de abril de 2024

Bernat el ferrer – asix1b

Santiago Lozada Benitez

índice

Contenido

[1. Introducción 2](#_Toc165049350)

[1.1 ¿Qué es MongoDB? 2](#_Toc165049351)

[1.2 Definición de bases de datos NO-SQL 2](#_Toc165049352)

[1.3 Tipos de datos NO-SQL 3](#_Toc165049353)

[1.4 Ventajas y desventajas frente a las bases de datos relacionales 4](#_Toc165049354)

[1.5 Conclusión y utilidad 5](#_Toc165049355)

[2. Creación de la estructura de datos en MongoDB 6](#_Toc165049356)

[2.1 Estructura de datos 6](#_Toc165049357)

[2.2 Creación de colecciones 6](#_Toc165049358)

[2.3 Creación de documentos 8](#_Toc165049359)

[2.4 Validación de la estructura de datos 8](#_Toc165049360)

[2.5 Operaciones CRUD 12](#_Toc165049361)

[2.5.1 Inserción de datos 12](#_Toc165049362)

[2.5.2 Buscador de datos (agregar expresiones regulares) 12](#_Toc165049363)

[2.5.3 Modificación de datos 13](#_Toc165049364)

[2.5.4 Eliminación de datos 13](#_Toc165049365)

[3. Comparaciones con otras bases de datos NO-SQL 14](#_Toc165049366)

[3.1 MongoDB vs Hbase 14](#_Toc165049367)

[3.2 MongoDB vs Neo4j 16](#_Toc165049368)

[3.3 MongoDB vs Zope 17](#_Toc165049369)

# Introducción

## 1.1 ¿Qué es MongoDB?

MongoDB es una base de datos No-SQL en la que almacena información desestructurada, es decir, no se organiza en tablas como en las bases de datos relacionales si no en ficheros con estructura BSON. Proporciona un modelo de datos flexibles, donde no se relacionan los datos de los ficheros entre sí. Además de ser una base de datos distribuida en su núcleo, por lo que la alta disponibilidad, la escalabilidad horizontal y la distribución geográfica están integradas.

## 1.2 Definición de bases de datos NO-SQL

Las Bases de datos NoSQL permiten almacenar y consultar datos fuera de las estructuras tradicionales, aunque puede guardar datos de bases de datos relacionales, aunque los almacenas maneras diferentes.

Es un tipo de base donde los datos están distribuidos, significa que la información se copia y almacena en los distintos servidores, pueden ser locales o remotos. Esto garantiza la disponibilidad y la fiabilidad de los datos.

En comparación con la estructura SQL si queremos gestionar diferentes datos de clientes, las tablas se unirían a través de una clave común “ID” que es muy práctico para almacenar y recuperar daos rápidamente, pero el inconveniente es que requiere una gran cantidad de memoria. Al agregar más memoria, la base SQL solo escala verticalmente no horizontalmente, por lo que significa que su memoria está muy limitada por el hardware que tenga. En cambio, con la estructura No-SQL no es necesario conectar las tablas y el uso de compartido facilitan mucho la escalabilidad horizontal.

## 1.3 Tipos de datos NO-SQL

Distintos tipos de bases de datos No-SQL:

* **Bases de datos de documentales:**

Son las más comunes, almacenan los datos en colecciones, siendo los elementos de cada una de esas colecciones documentos que contienen pares claves / valor, realiza consultas más avanzadas sobre el contenido del documento.

* **Bases de datos tabulares:**

Son conocidas en los sistemas de SQL, pero también se conocen como No-SQL. El modelo permite almacenar tablas, y admiten una mayor versatilidad ya que permite variar el formateo de los registros.

* **Bases de datos orientadas a grafo:**

La información se representa como nodos de un grafo y sus relaciones con las aristas de este. Este tipo de bases de datos ofrece una navegación más eficiente entre relaciones que en un modelo relacional.

* **Bases de datos de clave-valor:**

Este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de una manera más rápida.

* **Bases de datos multivalor:**

Permiten trabajar con pares de clave / valor, aunque almacenan múltiples valores asociados a una clave.

* **Bases de datos de tiempo real:**

Están especializados en trabajar con datos en tiempo real, ofrecen actualizaciones multimedia, y son perfectos para gestionar imágenes o contenido multimedia.

* **Bases de datos de almacén de objetos:**

Almacena los objetos que son elementos más abstractos que los documentos, pueden servir para almacenar binarios o archivos multimedia.

* **Bases de datos de memoria:**

Almacenan los datos en la memoria RAM del ordenador en lugar de en discos.

* **Bases de datos orientadas a objetos:**

La información se representa mediante objetos de la misma forma que son representaos en los lenguajes de programación orientada a objetos como es el caso en JAVA o C#.

* **Bases de datos distribuidas:**

Se encuentran en distintos sistemas alejados o en distintas redes, estos sistemas se conocen como nodos.

## 1.4 Ventajas y desventajas frente a las bases de datos relacionales

MongoDB destaca por su escalabilidad horizontal, lo que facilita la incorporación de nuevos servidores a la infraestructura de base de datos, ofreciendo una alternativa más flexible que las bases de datos SQL tradicionales. Esta característica es crucial para gestionar grandes volúmenes de datos y tráfico intenso sin sufrir interrupciones. Además de proporcionar una flexibilidad superior en el esquema de datos permitiendo modificaciones ágiles y el manejo de elementos con esquemas variados dentro de una misma estructura.

Otro aspecto importante es su capacidad para un rendimiento elevado en operaciones de lecturas y escrituras, optimizando eficientemente los datos de colecciones de gran tamaño. Asimismo, garantiza una alta disponibilidad y rendimiento en tiempo real ya que la configuración de estos sistemas es muy sencilla.

MongoDB es conocido por sus ventajas, también presenta ciertas desventajas. Por ejemplo, usualmente ofrece rendimiento de alta velocidad con la indexación adecuada, una implementación incorrecta de los índices puede resultar en una velocidad muy baja. Otro de los problemas sería el tamaño de datos y anidamiento limitados, permite un tamaño limitado de sólo 16 MB para un documento.

Otra consideración importante es la gestión del almacenamiento debido a la ausencia de funcionalidad de unión que conducen a la duplicación de datos, incrementando en la redundancia de datos que ocupa espacio innecesario en la memoria.

## 1.5 Conclusión y utilidad

En conclusión, MongoDB destaca por su versatilidad en la gestión de estructura de datos, ofreciendo la escalabilidad y la eficiencia notables, sin abandonar principios fundamentales de las bases de datos relaciones, como la consistencia de datos y la integración con otras herramientas de desarrollo.

Por sus características, MongoDB es una herramienta para el desarrollo de aplicaciones como redes sociales, aplicaciones móviles, entre otras, que demandan una base de datos capaz de proporcionar un alto rendimiento y flexibilidad, manteniendo al mismo tiempo la consistencia y seguridad en los datos.

# Creación de la estructura de datos en MongoDB

Para utilizar la base de datos de MongoDB es necesario tener instalado el Docker para poder trabajar mediante el CMD.

Para comenzar con el comando “Docker exec -it nombreMongoDB mongosh” activaremos la base de datos, a continuación, con el comando “use + nombreBaseDeDatos” creará o seleccionará la base. Si existe la seleccionará, sino la creará.

- **Show dbs**: Mostrará la lista de todas las bases de datos creadas

- **db.dropDatabase():** Eliminará la base de datos seleccionada.

**Texto

Descripción generada automáticamenteEjemplo:**

## 2.1 Estructura de datos

Esta estructura almacena datos en documentos, son similares a objetos JSON lo que permite que sea más flexible. Cada documento puede tener una estructura propia, lo que permite tener campos y matrices anidados eliminado de los esquemas prefinidos.

Además, existen las colecciones, son un grupo de documentos asociados a una base de datos. De igual modo son muy flexibles y no dependen de un esquema, además que los documentos no tienen por qué ser del mismo tipo de dato.

## 2.2 Creación de colecciones

En la creación de una colección no es necesario crearlo por ti mismo, sino que el sistema, las genera automáticamente cuando se crea una base de datos se añaden documentos a la misma.

db.createCollection (nombre\_de\_la\_colección, opciones)

La sintaxis de esta sería:

**db.createCollection:** Asigna una tarea al sistema.

**nombre\_de\_la\_colección:** Nombre de la colección.

**opciones:** {capped, autoIndexId, tamaño, max}.

* **capped:** Decide si la colección es capped(limitada), o no. También hay que especificar tamaño máximo. Si la colección supera su valor especifico, las entradas más antiguas se eliminarán automáticamente.
* **autoIndexId:** Recibe valor booleano, si es “true” crea un índice en el campo “\_id”, pero si es “false” se queda el valor por defecto.
* **tamaño:** Valor máximo de la colección con limites, solo es necesario si la opción de limite es “true” y hay que indicar el tamaño en bytes.
* **max:** especifica el número máximo de documentos diferentes que puede contener una colección con limite.

**Estructura de colección:**

{

name: “al”,

age: 18,

status: “D”,

}

Existen diferentes comandos para las colecciones:

* **show collections**: Muestra todas las colecciones
* **db.nombre\_coleccion.drop(**): Elimina una colección
* **db.nombre\_coleccion.insertOne(estructura de colección)**: Inserta datos a la colección
* **db.nombre\_coleccion.fin()**: Busca el documento en la colección

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente**Ejemplo**:

## 2.3 Creación de documentos

Para crear documentos, se utiliza las operaciones CRUD (Create, Read, Update, Detele).

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

## 2.4 Validación de la estructura de datos

Para asegurar una validación integral de los documentos se garantiza la integridad de cada campo, implementando un esquema específico de MongoDB. El esquema representa un conjunto de reglas para las propiedades (Keys) y valores de los documentos. La regla funciona en función de cada colección.

Por ejemplo, creamos un esquema de validación de los campos que hayamos definido:

* **Los campos**: Nombre de usuario, correo electrónico y contraseña deben estar presentes en cada documento (son obligatorios).
* **Los campos**: Nombre de usuario, correo electrónico y contraseña deben ser de tipo string, y la longitud de sus cadenas debe estar entre los límites mínimo y máximo.
* **El campo de correo electrónico**: Debe cumplir con un patrón de expresión regular específico.
* **El campo**: Fecha de registro (registrationdate) debe ser del tipo fecha.
* **Los campos**: Confirmado y cancelado deben ser de tipo booleano (Booleano: verdadero o falso).
* **Los campos typeid y countryid**: Deben ser de tipo int (entero), y sus valores deben estar entre un número mínimo y uno máximo.

Se definen todas reglas a través de varios métodos en mongo Shell CLI o mongosh CLI. Entonces se selecciona el esquema users (ejemplo, puede ser cualquier esquema), en la pestaña “Validation” y se pondría el esquema con los requisitos anteriormente mencionados.

**Documento de validación JSON:**

|  |
| --- |
| { |
|  | $jsonSchema: { |
|  | bsonType: 'object', |
|  | additionalProperties: false, |
|  | required: [ |
|  | 'username', |
|  | 'email', |
|  | 'password' |
|  | ], |
|  | properties: { |
|  | \_id: {}, |
|  | username: { |
|  | bsonType: 'string', |
|  | minLength: 6, |
|  | maxLength: 20, |
|  | description: 'It is required and it must be a string with length between 6 and 20' |
|  | }, |
|  | password: { |
|  | bsonType: 'string', |
|  | maxLength: 80, |
|  | description: 'It must be a string with max length 80' |
|  | }, |
|  | email: { |
|  | bsonType: 'string', |
|  | minLength: 6, |
|  | maxLength: 40, |
|  | pattern: '[a-z0-9.\_%+!$&\*=^|~#%{}/-]+@([a-z0-9-]+.){1,}([a-z]{2,22})', |
|  | description: 'It is required and it must be a string with length between 6 and 40 (regular expression pattern)' |
|  | }, |
|  | registrationdate: { |
|  | bsonType: 'date', |
|  | description: 'It must be a date' |
|  | }, |
|  | confirmed: { |
|  | bsonType: 'bool', |
|  | description: 'It can only be true or false' |
|  | }, |
|  | canceled: { |
|  | bsonType: 'bool', |
|  | description: 'It can only be true or false' |
|  | }, |
|  | typeid: { |
|  | bsonType: 'int', |
|  | minimum: 1, |
|  | maximum: 4, |
|  | description: 'It must be an integer in [ 1, 5 ]' |
|  | }, |
|  | countryid: { |
|  | bsonType: 'int', |
|  | minimum: 1, |
|  | maximum: 250, |
|  | description: 'It must be an integer in [ 1, 250 ]' |
|  | } |
|  | } |
|  | } |
|  | } |

Una vez listo el documento de validación, el mongosh debe estar conectado a la base de datos y con el comando **db.getCollectionInfos()**.

Para comprobar si funciona correctamente, se insertan datos erróneos que activarán el documento de validación.

## 2.5 Operaciones CRUD

El CRUD describe las funciones elementales de una base de datos, las siglas significan (Create, Read, Update, Detele). Las operaciones de crear e insertar nuevos documentos a una colección.

### 2.5.1 Inserción de datos

* **db.nombre\_coleccion.insertOne():** Inserta un documento a la colección.

Texto

Descripción generada automáticamente

* **db.nombre\_coleccion.insertMany():** Inserta varios documentos a las vez.

Texto

Descripción generada automáticamente

### 2.5.2 Buscador de datos (agregar expresiones regulares)

* **db.nombre\_coleccion.find()**: Imprime los primeros 20 documentos
  + **.limit(n):** Imprime el numero n de documentos
  + Texto

    Descripción generada automáticamente**.pretty()**: Imprime los documentos de forma más legible.
* Texto

  Descripción generada automáticamente**db.nombre\_coleccion.find({“clave”: “valor”}, {“clave”: “valor”})**: imprime los documentos que contenga la clave especificada.
* **db.nombre\_coleccion.find({“clave”:{$gt:”valor”}})**: imprime los documentos mayores al valor de alguna clave en un documento.
  + **$gt**: Significa mayor que (>)
  + **$lt**: Significa menor que (<)
  + **$gte**: Significa mayor o igual a (>=)
  + **$lte**: Significa menor o igual a (<=)
* Interfaz de usuario gráfica, Texto

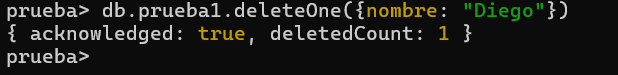
  Descripción generada automáticamente**db.nombre\_coleccion.findOne()**: Imprime solo el primer documento que encuentra.

### 2.5.3 Modificación de datos

* **db.nombre\_coleccion.updateOne({filtro}, {"clave": "valor"})**: Se actualizará el primer documentos que coincida con el filtro.
* **db.nombre\_coleccion.updateMany({filtro}, {"clave": "valor"})**: Se actualizará todos los documentos que coincida con el filtro

### 2.5.4 Eliminación de datos

* **db.nombre\_coleccion.deleteOne({"filter")}**: Elimina el primer documentos encontrao según el filtro



* **db.nombre\_coleccion.deleteMany({"filter")}**: Elimina todos los documentos encontrados según el filtro

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

* Texto

  Descripción generada automáticamente**db.nombre\_coleccion.drop()**: Elimina todos los documentos de una colección.

# 3. Comparaciones con otras bases de datos NO-SQL

Existen otros tipos de bases de datos que tienen la misma funcionalidad que MongoDB, pero cada una tiene características diferentes, se puede observar diferencias en 3 bases de datos diferentes.

## 3.1 MongoDB vs Hbase

Hbase es un sistema de almacenamiento de columnas amplias inspirado en Google BigTable, desarrollado en Java y que utiliza HDFS. Está diseñado para cargas de trabajo de valor calve con patrones de acceso de lectura y escrita aleatorios.

Funciona como un mapa multidimensional persistente, distribuido y disperso, indexado mediante una clave de fila, una calve columna y una marca de tiempo, cada valor en el mapa es una matriz de bytes no interpretada.

En el ámbito de las bases de datos no relacionales, tanto MongoDB como Hbase ofrecen flexibilidad en el modelo de datos, la capacidad de escalamiento horizontal con fragmentación y alto rendimiento de lectura y escritura, sin embargo, existen algunas diferencias fundamentales entre los que se destacan en la tabla siguiente.

**Escalabilidad:**

* **MongoDB:** Ofrece una escalabilidad horizontal a través de la fragmentación, distribuyendo datos a través de múltiples servidores.
* **HBase:** Diseñada para escalar de manera lineal y modular, puede manejar enormes cantidades de datos distribuyéndolos en un clúster de servidores.

**Modelo de Datos:**

* **MongoDB:** Utiliza un modelo de datos basado en documentos, lo que permite una gran flexibilidad en la estructura de los datos.
* **HBase:** Emplea un modelo de datos de clave-valor que es adecuado para patrones de acceso aleatorio y para datos que no se ajustan bien a un esquema rígido.

**Consistencia y Disponibilidad:**

* **MongoDB:** Proporciona consistencia fuerte en operaciones de un solo documento y soporte para transacciones multi-documento.
* **HBase:** Ofrece consistencia estricta en lecturas y escrituras, lo que es crucial para aplicaciones que necesitan garantizar la integridad de los datos a través de todas las operaciones.

**Rendimiento:**

* **MongoDB:** Tiene un alto rendimiento en operaciones de lectura y escritura, especialmente con datos que están relacionados y pueden ser almacenados juntos en un solo documento.
* **HBase:** Optimiza el rendimiento en grandes volúmenes de datos y cargas de trabajo intensivas de lectura/escritura, gracias a su integración con Hadoop y el uso de HDFS.

**Casos de Uso:**

* **MongoDB:** Es ideal para aplicaciones que necesitan una rápida iteración de desarrollo, manejo de diferentes tipos de datos y estructuras complejas, y donde la escalabilidad es una preocupación clave.
* **HBase:** Es más adecuada para aplicaciones analíticas y de procesamiento de big data que requieren un acceso rápido a grandes volúmenes de datos dispersos.

## 3.2 MongoDB vs Neo4j

Es un almacén de datos grafos nativos creado desde cero para aprovechar no solo los datos, sino también las relaciones entre los datos. A diferencia de otros tipos de bases de datos, conecta los datos a medida que se almacenan.

Contienen un gran rendimiento que las relacionales y las no relacionales, la clave es que las consultas de datos aumenten exponencialmente y el rendimiento no descienda.

**Escalabilidad:**

* **MongoDB:** Ofrece una escalabilidad horizontal a través de la fragmentación, distribuyendo datos a través de múltiples servidores.
* **Neo4j**: Es ideal para escenarios donde las relaciones y consultas complejas son fundamentales. Es posible que no escale tan horizontalmente como MongoDB para manejar cantidades masivas de datos.

**Modelo de Datos:**

* **MongoDB:** utiliza un modelo de datos basado en documentos, donde los datos se almacenan en colecciones de documentos similares a JSON. Cada documento puede tener estructuras anidadas.
* **Neo4j:** utiliza un modelo de datos basado en gráficos, donde los datos se representan como nodos, relaciones y propiedades. Esto es ideal para representar y consultar datos altamente interconectados.

**Consistencia y Disponibilidad:**

* **MongoDB:** Proporciona consistencia fuerte en operaciones de un solo documento y soporte para transacciones multi-documento.

**Rendimiento:**

* **MongoDB:** Tiene un alto rendimiento en operaciones de lectura y escritura, especialmente con datos que están relacionados y pueden ser almacenados juntos en un solo documento.
* **Neo4j:** Optimiza el rendimiento en grandes volúmenes de datos y cargas de trabajo intensivas de lectura/escritura, gracias a su integración con Hadoop y el uso de HDFS.

**Casos de Uso:**

* **MongoDB:** Es ideal para aplicaciones que necesitan una rápida iteración de desarrollo, manejo de diferentes tipos de datos y estructuras complejas, y donde la escalabilidad es una preocupación clave.
* **Neo4j:** es fundamental comprender las relaciones y recorrer caminos entre entidades, como redes sociales, sistemas de recomendación y gráficos de conocimiento.

## 3.3 MongoDB vs Zope

Es un almacenamiento de forma transparente y persistente objetos en el lenguaje de programación de Python. Parte de Zope, es un servidor de aplicaciones web, pero también puede ser utilizada de manera independiente.

**Escalabilidad**:

* **MongoDB:** Ofrece una escalabilidad horizontal a través de la fragmentación, distribuyendo datos a través de múltiples servidores.
* **Zope**: También es escalable, no es tan flexible, pero puede requerir más esfuerzo para escalar horizontalmente.

**Modelo de Datos**:

* **MongoDB:** utiliza un modelo de datos basado en documentos, donde los datos se almacenan en colecciones de documentos similares a JSON. Cada documento puede tener estructuras anidadas.
* **Zope**: Emplea un modelo de datos **relacional** con tablas y relaciones predefinidas. Es más rígido en términos de esquema.

**Consistencia y Disponibilidad**:

* **MongoDB:** Proporciona consistencia fuerte en operaciones de un solo documento y soporte para transacciones multi-documento.
* **Zope**: Proporciona consistencia ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) pero puede ser menos flexible en términos de disponibilidad.

**Rendimiento**:

* **MongoDB:** Tiene un alto rendimiento en operaciones de lectura y escritura, especialmente con datos que están relacionados y pueden ser almacenados juntos en un solo documento.
* **Zope**: El rendimiento depende de la complejidad de las consultas y la optimización del esquema. Puede ser más lento en algunas operaciones.

**Casos de Uso**:

* **MongoDB:** Es ideal para aplicaciones que necesitan una rápida iteración de desarrollo, manejo de diferentes tipos de datos y estructuras complejas, y donde la escalabilidad es una preocupación clave.
* **Zope** Se utiliza comúnmente en aplicaciones empresariales, sistemas transaccionales y sistemas de informes.