LABORATORIO DE BASES DE DATOS NoSQL CON MONGODB

Guía No.2.

Bases de datos No Relacionales
No SQL

Docente:

Julian René Muñoz Burbano

Universidad Autónoma de Occidente

2025

Objetivo del Laboratorio

Este laboratorio tiene como objetivo reforzar los conocimientos sobre bases de datos NoSQL con MongoDB. A través de preguntas teóricas y ejercicios prácticos, los estudiantes podrán:

- Comprender el funcionamiento de MongoDB.
- Crear, insertar, actualizar y eliminar documentos en una base de datos MongoDB.
- Realizar consultas avanzadas.
- Optimizar el uso de MongoDB mediante indexación y agregaciones.

Preguntas Teóricas

Responde las siguientes preguntas:

1. ¿Qué es una base de datos NoSQL y cuáles son sus principales ventajas?
2. ¿Cuáles son las diferencias entre una base de datos relacional y una NoSQL?
3. ¿Qué tipo de datos almacena MongoDB?
4. Explica la estructura de un documento en MongoDB.
5. ¿Qué es una colección en MongoDB y en qué se diferencia de una tabla en SQL?
6. ¿Cómo se maneja la consistencia en MongoDB?
7. ¿Cuál es la función de BSON en MongoDB?
8. Explica la diferencia entre `insertOne()` e `insertMany()`.
9. ¿Cómo se realiza una consulta en MongoDB para encontrar un documento específico?
10. ¿Cuál es la diferencia entre `updateOne()` y `updateMany()`?
11. ¿Cómo se eliminan documentos en MongoDB?
12. Explica el uso del operador `$set` en una actualización.
13. ¿Qué hace el operador `$gt` en una consulta?
14. ¿Cómo se indexan los documentos en MongoDB y para qué sirve la indexación?
15. ¿Qué es un `aggregation pipeline` en MongoDB y para qué se utiliza?
16. Explica el operador `$match` en un pipeline de agregación.
17. ¿Cuál es la utilidad del operador `$group` en un `aggregation pipeline`?
18. ¿Cómo se filtran documentos por múltiples condiciones en MongoDB?
19. Explica el concepto de replicación en MongoDB.
20. ¿Cómo se maneja la fragmentación (sharding) en MongoDB?

Ejercicios Prácticos

Realiza los siguientes ejercicios en MongoDB:

21. Crea una base de datos llamada `laboratorio_mongo`.
22. Crea una colección llamada `empleados` e inserta un documento con los siguientes campos:

```
{
    "nombre": "Luis Martínez",
    "edad": 30,
    "cargo": "Analista de Datos",
    "salario": 3500
}
```

23. Inserta 5 documentos adicionales con datos de empleados.
24. Consulta todos los empleados.
25. Encuentra el empleado con nombre "Luis Martínez".
26. Encuentra todos los empleados con salario superior a 3000.
27. Actualiza el salario de "Luis Martínez" a 4000.
28. Aumenta el salario de todos los empleados en 500 unidades.
29. Cambia el cargo de un empleado específico.
30. Elimina un empleado según su nombre.
31. Inserta una colección `productos` con 5 documentos representando productos con nombre, precio y stock.
32. Encuentra los productos con un stock menor a 10 unidades.
33. Aplica una agregación para obtener el precio promedio de los productos.
34. Filtra los empleados que tienen cargos que contienen la palabra "Ingeniero".
35. Indexa la colección `empleados` en el campo `nombre`.
36. Usa `$group` para obtener el salario promedio de todos los empleados.
37. Encuentra todos los empleados con edades entre 25 y 35 años.
38. Ordena los empleados por salario de mayor a menor.
39. Encuentra los empleados con salario entre 2500 y 4000.
40. Inserta una colección `ventas` con documentos que incluyan fecha, producto vendido y cantidad.
41. Usa `$match` para filtrar ventas de un producto específico.

42. Aplica `$group` en `ventas` para obtener el total de productos vendidos por categoría.
43. Ordena los productos en `productos` por precio ascendente.
44. Usa `$lookup` para unir `empleados` y `ventas` según el nombre del empleado que realizó la venta.
45. Implementa una búsqueda de texto en `productos` para encontrar aquellos con una palabra clave.

Solución y Explicación de los Ejercicios

A continuación, se presenta la solución y explicación de cada ejercicio:

- **Ejemplo de consulta:**

  ```
  db.empleados.find({ "salario": { "$gt": 3000 } }).pretty()
  ```

  Explicación: Se usa `$gt` para encontrar empleados con salario mayor a 3000.

- **Ejemplo de agregación:**

  ```
  db.ventas.aggregate([
      { "$group": { "_id": "$producto", "total_vendido": { "$sum": "$cantidad" } }
  ])
  ```

  Explicación: Se agrupan las ventas por producto y se obtiene el total vendido.

Cada ejercicio incluye la solución correspondiente utilizando los comandos de MongoDB y una explicación detallada de su función.

Respuestas a las Preguntas Teóricas

1. **¿Qué es una base de datos NoSQL y cuáles son sus principales ventajas?**
   - Una base de datos NoSQL es un sistema de almacenamiento de datos que no usa un esquema fijo. Sus principales ventajas incluyen escalabilidad, flexibilidad y facilidad de uso para datos no estructurados.
2. **¿Cuáles son las diferencias entre una base de datos relacional y una NoSQL?**
   - Las bases de datos relacionales usan tablas y relaciones fijas, mientras que las NoSQL usan documentos, clave-valor, columnas o grafos para almacenar información.
3. **¿Qué tipo de datos almacena MongoDB?**
   - MongoDB almacena datos en documentos BSON, un formato binario de JSON que permite almacenar estructuras anidadas.
4. **Explica la estructura de un documento en MongoDB.**
   - Un documento en MongoDB es similar a un objeto JSON con pares clave-valor. Ejemplo:

     ```
     {
         "nombre": "Juan Pérez",
         "edad": 30,
         "cargo": "Desarrollador"
     }
     ```
5. **¿Qué es una colección en MongoDB y en qué se diferencia de una tabla en SQL?**
   - Una colección es un conjunto de documentos, similar a una tabla en SQL, pero sin esquema fijo.
6. **¿Cómo se maneja la consistencia en MongoDB?**
   - MongoDB maneja la consistencia con operaciones atómicas a nivel de documento y soporte para transacciones.
7. **¿Cuál es la función de BSON en MongoDB?**
   - BSON es un formato binario optimizado para el almacenamiento y recuperación de documentos en MongoDB.
8. **Explica la diferencia entre `** e **`.**
   - `insertOne()` inserta un solo documento, mientras que `insertMany()` permite insertar varios documentos en una sola operación.
9. **¿Cómo se realiza una consulta en MongoDB para encontrar un documento específico?**

   ```
   db.empleados.findOne({ "nombre": "Luis Martínez" })
   ```

10. **¿Cuál es la diferencia entre `** y **`?**
    - `updateOne()` actualiza un solo documento, mientras que `updateMany()` actualiza todos los documentos que cumplen la condición.
11. **¿Cómo se eliminan documentos en MongoDB?**
    ```
    db.empleados.deleteOne({ "nombre": "Luis Martínez" })
    ```
12. **Explica el uso del operador `` en una actualización.**
    - `$set` permite actualizar valores específicos sin afectar el resto del documento.
13. **¿Qué hace el operador `` en una consulta?**
    - Filtra valores mayores que un número dado. Ejemplo:
      ```
      db.empleados.find({ "salario": { "$gt": 3000 } })
      ```
14. **¿Cómo se indexan los documentos en MongoDB y para qué sirve la indexación?**
    - Se usa `createIndex()` para mejorar el rendimiento de las consultas.
15. **¿Qué es un `` en MongoDB y para qué se utiliza?**
    - Es una serie de etapas que procesan datos, útil para análisis avanzados.
16. **Explica el operador `` en un pipeline de agregación.**
    - Filtra documentos que cumplen ciertas condiciones.
17. **¿Cuál es la utilidad del operador `** en un **`?**
    - Agrupa documentos y calcula valores agregados como promedios o sumas.
18. **¿Cómo se filtran documentos por múltiples condiciones en MongoDB?**
    ```
    db.empleados.find({ "$and": [{ "edad": { "$gte": 25 } }, { "salario": { "$lt": 4
    ```
19. **Explica el concepto de replicación en MongoDB.**
    - La replicación permite mantener múltiples copias de datos para alta disponibilidad.
20. **¿Cómo se maneja la fragmentación (sharding) en MongoDB?**
    - Divide la base de datos en múltiples servidores para escalar horizontalmente.

Solución a los Ejercicios Prácticos

21. **Crear una base de datos:**

```
use laboratorio_mongo
```

22. **Crear una colección e insertar un documento:**

```
db.empleados.insertOne({ "nombre": "Luis Martínez", "edad": 30, "cargo": "Analis
```

23. **Insertar múltiples documentos:**

```
db.empleados.insertMany([...])
```

24. **Consultar todos los empleados:**

```
db.empleados.find().pretty()
```

25. **Encontrar un empleado específico:**

```
db.empleados.findOne({ "nombre": "Luis Martínez" })
```

26. **Buscar empleados con salario mayor a 3000:**

```
db.empleados.find({ "salario": { "$gt": 3000 } }).pretty()
```

27. **Actualizar salario de un empleado:**

```
db.empleados.updateOne({ "nombre": "Luis Martínez" }, { "$set": { "salario": 400
```

28. **Incrementar salario de todos los empleados:**

```
db.empleados.updateMany({}, { "$inc": { "salario": 500 } })
```

29. **Cambiar cargo de un empleado:**

```
db.empleados.updateOne({ "nombre": "Luis Martínez" }, { "$set": { "cargo": "Gere
```

30. **Eliminar un empleado:**

```
db.empleados.deleteOne({ "nombre": "Luis Martínez" })
```

31. **Insertar productos:**

```
db.productos.insertMany([...])
```

32. **Buscar productos con stock menor a 10:**

```
db.productos.find({ "stock": { "$lt": 10 } })
```

33. **Calcular precio promedio de productos:**

```
db.productos.aggregate([ { "$group": { "_id": null, "precioPromedio": { "$avg":
```

34. **Filtrar empleados con cargo de "Ingeniero":**

```
db.empleados.find({ "cargo": /Ingeniero/ })
```

35. **Crear índice en nombre:**

```
db.empleados.createIndex({ "nombre": 1 })
```

36. **Obtener salario promedio:**

```
db.empleados.aggregate([ { "$group": { "_id": null, "salarioPromedio": { "$avg":
```

37. Encuentra todos los empleados con edades entre 25 y 35 años.

```
db.empleados.find({ "edad": { "$gte": 25, "$lte": 35 } })
```

38. Ordena los empleados por salario de mayor a menor.

```
db.empleados.find().sort({ "salario": -1 })
```

39. Encuentra los empleados con salario entre 2500 y 4000.

```
db.empleados.find({ "salario": { "$gte": 2500, "$lte": 4000 } })
```

40. Inserta una colección `ventas` con documentos que incluyan fecha, producto vendido y cantidad.

```
db.ventas.insertMany([
    { "fecha": "2024-01-15", "producto": "Laptop", "cantidad": 3 },
    { "fecha": "2024-01-20", "producto": "Celular", "cantidad": 5 }
])
```

41. Usa `$match` para filtrar ventas de un producto específico.

```
db.ventas.aggregate([
    { "$match": { "producto": "Laptop" } }
])
```

42. Aplica `$group` en `ventas` para obtener el total de productos vendidos por categoría.

```
db.ventas.aggregate([
    { "$group": { "_id": "$producto", "total_vendido": { "$sum": "$cantidad" } } }
])
```

43. Ordena los productos en `productos` por precio ascendente.

```
db.productos.find().sort({ "precio": 1 })
```

44. Usa `$lookup` para unir `empleados` y `ventas` según el nombre del empleado que realizó la venta.

```
db.ventas.aggregate([
    {
        "$lookup": {
            "from": "empleados",
            "localField": "empleado",
            "foreignField": "nombre",
            "as": "detallesEmpleado"
        }
    }
])
```

45. Implementa una búsqueda de texto en `productos` para encontrar aquellos con una palabra clave.

```
db.productos.createIndex({ "nombre": "text" })
db.productos.find({ "$text": { "$search": "Laptop" } })
```