1. Índices en MongoDB

MongoDB permite la creación de índices para optimizar las consultas. Sin índices, las búsquedas pueden ser ineficientes en bases de datos con grandes volúmenes de datos.

- **Crear un índice simple:**

```
db.usuarios.createIndex({ nombre: 1 })
```

Este índice mejora la velocidad de búsqueda en el campo `nombre`, ya que los documentos se almacenarán en orden ascendente basado en ese campo. Esto es especialmente útil cuando realizamos muchas búsquedas por este campo.

- **Crear un índice compuesto:**

```
db.usuarios.createIndex({ nombre: 1, edad: -1 })
```

Este índice mejora el rendimiento en consultas que filtran por `nombre` y ordenan por `edad` en orden descendente. Es útil cuando queremos recuperar registros ordenados por múltiples criterios de búsqueda.

- **Eliminar un índice:**

```
db.usuarios.dropIndex("nombre_1")
```

Borra el índice previamente creado en el campo `nombre`, lo que puede ralentizar futuras búsquedas si ese campo era clave en las consultas. Se recomienda eliminar índices solo cuando ya no sean necesarios.

- **Listar los índices de una colección:**

```
db.usuarios.getIndexes()
```

Devuelve un listado de todos los índices existentes en la colección `usuarios`, mostrando su estructura y optimización. Es útil para evaluar el rendimiento de la base de datos.

- **Selección de índices en consultas:**
  MongoDB decide automáticamente qué índice utilizar basándose en las estadísticas de uso y la estructura de la consulta. Sin embargo, se puede forzar el uso de un índice específico con `hint`:
- `db.usuarios.find({ nombre: "Carlos" }).hint({ nombre: 1 })`

Esto obliga a MongoDB a usar el índice sobre `nombre`, lo cual puede ser útil para pruebas de rendimiento o depuración de consultas lentas.

2. Agregaciones

La función `aggregate()` en MongoDB permite realizar operaciones avanzadas en los documentos de una colección. Se compone de una serie de etapas (`stages`), donde cada etapa transforma los datos en un proceso secuencial.

Funcionamiento de `aggregate()`

`aggregate()` recibe un array de etapas. Cada etapa aplica una transformación específica a los documentos y los pasa a la siguiente etapa en la secuencia.

Ejemplo de estructura básica:

```
 db.usuarios.aggregate([
   { $match: { edad: { $gte: 18 } } },  // Filtra documentos
   { $group: { _id: "$ciudad", totalUsuarios: { $sum: 1 } } },  //
Agrupa datos
   { $sort: { totalUsuarios: -1 } }  // Ordena resultados
 ])
```

Este ejemplo primero filtra usuarios mayores de 18 años (`$match`), luego agrupa los usuarios por ciudad (`$group`) y finalmente ordena los resultados en orden descendente (`$sort`).

Principales operadores de agregación

- **Filtrar datos con `$match`**

```
db.usuarios.aggregate([
  { $match: { edad: { $gte: 18 } } }
])
```

  Filtra los documentos para incluir solo aquellos usuarios cuya edad es 18 o más. Es similar a la cláusula `WHERE` en SQL y mejora el rendimiento de la agregación reduciendo el número de documentos procesados.

- **Agrupar datos con `$group`**

```
db.usuarios.aggregate([
  { $group: { _id: "$ciudad", totalUsuarios: { $sum: 1 } } }
])
```

  Agrupa los documentos por `ciudad` y cuenta cuántos usuarios hay en cada una, permitiendo generar estadísticas de uso o reportes.

- **Ordenar datos con `$sort`**

```
db.usuarios.aggregate([
  { $sort: { totalUsuarios: -1 } }
])
```

Ordena los resultados en orden descendente basado en `totalUsuarios`, útil para mostrar rankings o tendencias.

- **Proyectar y renombrar campos con `$project`**

```
db.usuarios.aggregate([
  { $project: { nombreCompleto: { $concat: ["$nombre", " ",
"$apellido"] }, _id: 0 } }
])
```

Permite modificar la estructura de salida combinando `nombre` y `apellido` en un nuevo campo `nombreCompleto`, ocultando `_id`. Esto es útil para mejorar la presentación de los datos en reportes.

3. Uniones entre colecciones con `$lookup`

El operador `$lookup` permite realizar una unión entre documentos de diferentes colecciones, equivalente a `JOIN` en SQL.

- **Unir datos entre `pedidos` y `usuarios`:**

```
db.pedidos.aggregate([
  {
    $lookup: {
      from: "usuarios",
      localField: "usuario_id",
      foreignField: "_id",
      as: "informacion_usuario"
    }
  }
])
```

Asocia cada pedido con la información del usuario correspondiente, almacenando los datos en un array `informacion_usuario`. Esto permite recuperar información detallada sin necesidad de hacer múltiples consultas.

- **Expandir los resultados con `$unwind`**

```
db.pedidos.aggregate([
  { $lookup: { from: "usuarios", localField: "usuario_id",
foreignField: "_id", as: "usuario" } },
  { $unwind: "$usuario" }
])
```

Convierte el array de usuarios en documentos individuales, facilitando la manipulación y análisis de datos.

4. Almacenamiento de resultados con `$merge`

El operador `$merge` permite almacenar los resultados de una agregación en una colección existente o crear una nueva.

- **Guardar resultados de agregación en una nueva colección:**

```
db.usuarios.aggregate([
  { $group: { _id: "$ciudad", totalUsuarios: { $sum: 1 } } },
  { $merge: { into: "resumen_ciudades", whenMatched: "merge",
whenNotMatched: "insert" } }
])
```

> *Agrupa los usuarios por `ciudad`, cuenta cuántos hay en cada una y almacena los resultados en `resumen_ciudades`. Si la colección ya existe, combina los datos.*