

Clase 11: Material Complementario

Sitio: [Centro de E-Learning - UTN.BA](#)
Curso: Curso de Backend Developer - Turno
Noche
Libro: Clase 11: Material Complementario

Imprimido
por: Nilo Crespi
Día: Friday, 23 de January de 2026,
10:23

Tabla de contenidos

1. Modelado de Datos y Consultas Avanzadas en MongoDB

1.1. Modelado de Datos y Consultas Avanzadas en MongoDB

1. Modelado de Datos y Consultas Avanzadas en MongoDB

Bloques temáticos

1. Modelado de datos con documentos y esquemas flexibles
2. Consultas avanzadas con operadores y proyecciones
3. Uso de índices para mejorar el rendimiento de las consultas
4. Consultas de agregación para operaciones complejas
5. Consultas geoespaciales y uso de índices espaciales
6. Trabajo con fechas y consultas de rango de tiempo
7. Transacciones y operaciones atómicas en MongoDB

1.1. Modelado de Datos y Consultas Avanzadas en MongoDB

Modelado de datos con documentos y esquemas flexibles

MongoDB, como una base de datos NoSQL, se basa en un modelo de documentos. En lugar de tablas y filas, en MongoDB almacenamos los datos en documentos BSON (Binary JSON). Este enfoque brinda flexibilidad para el diseño de datos, lo que es fundamental en aplicaciones en constante evolución. Exploraremos a fondo cómo funcionan los documentos BSON y cómo aprovechar esta flexibilidad en el modelado de datos.

Documentos en MongoDB:

En MongoDB, un documento es una representación de datos en un formato BSON. BSON es una notación binaria que es similar a JSON (JavaScript Object Notation) pero más eficiente en términos de almacenamiento y velocidad de procesamiento.

Cada documento es una entidad autónoma que contiene uno o más campos, y los campos pueden ser de diferentes tipos, como cadenas de texto, números, arreglos, objetos anidados, y más. La capacidad de almacenar diferentes tipos de datos en un documento lo hace especialmente adecuado para datos semiestructurados o no estructurados.

Un ejemplo sencillo de un documento en MongoDB podría ser:

```
{
  "_id": 1,
  "nombre": "Ejemplo",
  "edad": 30,
  "correos": ["ejemplo@gmail.com", "otro@email.com"],
  "direccion": {
    "calle": "123 Main St",
    "ciudad": "Ciudad Ejemplo"
  }
}
```

En este ejemplo, el documento contiene campos como "_id" para identificación, "nombre" para el nombre de una persona, "edad" para la edad, "correos" para una lista de correos electrónicos y "direccion" para un objeto anidado que describe la dirección.

Esquemas Flexibles

Una de las características distintivas de MongoDB es su esquema flexible. A diferencia de las bases de datos relacionales que requieren un esquema fijo y predefinido, en MongoDB, los documentos en una colección no necesitan tener la misma estructura. Esto significa que se pueden agregar o eliminar campos de manera dinámica en los documentos sin afectar a otros documentos en la misma colección.

La flexibilidad en el esquema es útil en situaciones en las que los requisitos de la aplicación cambian con el tiempo. Puede adaptar el esquema de la base de datos para acomodar nuevos datos o requerimientos sin realizar migraciones complicadas.

Por ejemplo, imagina una base de datos de gestión de productos. En un enfoque de esquema rígido, tendrías que definir todos los campos posibles de un producto, como nombre, descripción, precio, categoría, etc., y cumplir con esa estructura para todos los productos.

Sin embargo, en MongoDB, podrías tener documentos de productos que difieran en términos de qué campos contienen. Por lo tanto, un producto puede tener una estructura diferente de otro, y la base de datos seguirá siendo coherente.

Diseño de Esquemas

A pesar de la flexibilidad, es importante diseñar un esquema que sea coherente y eficiente para su aplicación. Algunos consejos para el diseño de esquemas en MongoDB incluyen:

- Identificar los patrones de acceso a los datos.
- Anidar datos cuando sea necesario para simplificar las consultas.
- Utilizar índices para acelerar búsquedas.
- Evitar la duplicación excesiva de datos.

Un diseño de esquema eficiente garantiza que las consultas sean rápidas y los datos se almacenen de manera coherente. Aunque MongoDB permite esquemas flexibles, no significa que debas abandonar el diseño y la estructura de tus datos por completo.

Aquí hay algunas consideraciones clave para el diseño de esquemas en MongoDB:

- **Modelado según el uso:** Antes de diseñar un esquema, comprende cómo se utilizarán los datos en tu aplicación. Esto te ayudará a decidir qué información almacenar en documentos y cómo relacionarlos si es necesario.
- **Denormalización:** MongoDB permite la denormalización de datos, lo que significa que puedes almacenar información redundante en varios documentos para mejorar la velocidad de recuperación de datos. Sin embargo, es importante equilibrar la denormalización con la eficiencia de almacenamiento y la consistencia de los datos.
- **Optimización de consultas:** Considera las consultas que se realizarán con mayor frecuencia y diseña tu esquema para que estas consultas sean eficientes. Esto puede incluir la creación de índices o la organización de datos de una manera que minimice la necesidad de operaciones costosas.
- **Relaciones:** A pesar de que MongoDB es una base de datos NoSQL, aún puedes modelar

relaciones entre datos. Esto se puede hacer utilizando referencias o subdocumentos, dependiendo de la naturaleza de la relación y las necesidades de la aplicación.

- **Tamaño del documento:** Ten en cuenta que MongoDB tiene un límite de tamaño de documento (por defecto, 16 MB), por lo que, si prevés documentos muy grandes, es necesario dividir los datos o utilizar la técnica de referencias.
- **Escalabilidad:** Diseña tu esquema pensando en la escalabilidad a largo plazo. MongoDB facilita la escalabilidad horizontal, lo que significa que puedes distribuir los datos en múltiples servidores, y el diseño del esquema debe ser compatible con esta arquitectura.

Consultas avanzadas con operadores y proyecciones

Las consultas avanzadas con operadores y proyecciones son fundamentales en MongoDB para recuperar y manipular datos de manera específica. MongoDB proporciona una variedad de operadores y opciones de proyección que permiten refinar las consultas y recuperar solo la información necesaria. Aquí hay una explicación detallada de estos conceptos:

Operadores en MongoDB

- **Comparación:** MongoDB ofrece operadores como ``$eq`` (igual), ``$ne`` (no igual), ``$gt`` (mayor que), ``$lt`` (menor que), entre otros, para comparar valores en los documentos.
- **Lógicos:** Puedes utilizar operadores lógicos como ``$and``, ``$or``, y ``$not`` para combinar múltiples condiciones en una consulta.
- **Elemento dentro de un arreglo:** MongoDB permite buscar documentos en los que un campo sea un elemento en un arreglo utilizando el operador ``$in``. También puedes verificar si un campo no está en un arreglo usando ``$nin``.
- **Existencia de un campo:** Puedes usar ``$exists`` para buscar documentos basados en si un campo específico existe o no en el documento.
- **Expresiones regulares:** MongoDB admite expresiones regulares para buscar patrones específicos en los valores de los campos.
- **Elementos de un arreglo:** Los operadores ``$elemMatch`` y ``$all`` permiten buscar documentos donde al menos un elemento o todos los elementos de un arreglo cumplan ciertas condiciones.

Proyecciones en MongoDB

Las proyecciones se utilizan para definir qué campos deseas recuperar en el resultado de una consulta. Puedes incluir o excluir campos específicos en la salida. Esto es útil para minimizar el ancho de banda y reducir la cantidad de datos transferidos entre la base de datos y la aplicación.

Algunas de las opciones de proyección comunes incluyen:

- **Inclusión de campos:** Usando el operador `\$project`, puedes especificar explícitamente los campos que deseas incluir en los resultados de la consulta.
- **Exclusión de campos:** Puedes usar `\$project` con la opción `exclude` o el operador `\$unset` para excluir campos específicos del resultado.
- **Renombrar campos:** Utiliza `\$project` con la opción `as` para cambiar el nombre de los campos en los resultados.
- **Arrays:** Puedes utilizar el operador `\$slice` para limitar la cantidad de elementos que se devuelven en un arreglo.
- **Elementos de arreglo en proyecciones:** El operador `\$elemMatch` permite proyectar solo el primer elemento de un arreglo que cumple ciertas condiciones.

Algunos ejemplos de cómo usar proyecciones en una consulta son:

- `db.miColeccion.find({}, { campo1: 1, campo2: 1 })`

(Recuperará solo los campos campo1 y campo2 de los documentos encontrados).

- `db.miColeccion.find({}, { _id: 0, campo1: 1 })`

(Excluirá el campo _id y recuperará solo el campo campo1).

Ejemplo de consulta avanzada:

Imagina que tienes una colección de usuarios y deseas encontrar todos los usuarios con una edad mayor de 25 años y mostrar solo sus nombres y correos electrónicos. Puedes hacerlo con una consulta avanzada que utiliza operadores y proyecciones:

```
db.usuarios.find({ edad: { $gt: 25 } }, { nombre: 1, correo: 1, _id: 0 })
```

Esta consulta buscará usuarios con una edad mayor de 25 y mostrará solo los campos nombre y correo, excluyendo el campo _id.

Uso de índices para mejorar el rendimiento de las consultas

El uso de índices es una estrategia fundamental para mejorar el rendimiento de las consultas en MongoDB. Los índices son estructuras de datos que permiten a MongoDB acelerar la búsqueda y recuperación de documentos en una colección. Aquí se describen los conceptos clave relacionados con el uso de índices:

Índices en MongoDB

En MongoDB, los índices se crean en uno o varios campos de una colección. Estos campos se

seleccionan según los criterios de búsqueda comunes en tus consultas.

Cuando se crea un índice en un campo, MongoDB crea una estructura de datos especial que almacena las referencias a los documentos y sus valores correspondientes para ese campo. Esto permite que las consultas que involucran ese campo sean mucho más eficientes.

Ventajas de utilizar índices:

- **Rendimiento mejorado:** Los índices permiten a MongoDB buscar y recuperar documentos mucho más rápido, especialmente en colecciones grandes. Sin índices, MongoDB tendría que realizar exploraciones completas de la colección, lo que puede ser costoso en términos de recursos y tiempo.
- **Reducción de la carga en el servidor:** Al acelerar las consultas, los índices reducen la carga en el servidor de MongoDB, lo que mejora la capacidad de respuesta y la escalabilidad del sistema.
- **Optimización de la experiencia del usuario:** Las consultas más rápidas significan una experiencia de usuario más eficiente y satisfactoria en las aplicaciones que utilizan MongoDB.

Creación de índices en MongoDB

Puedes crear índices en MongoDB utilizando el método `createIndex()` o la opción `createIndex` en una operación `createCollection`. Al crear un índice, especificas los campos en los que se creará el índice y, opcionalmente, opciones adicionales como la unicidad o la dirección de orden del índice.

Ejemplo de creación de un índice:

Supongamos que tienes una colección de libros y deseas crear un índice en el campo `titulo` para acelerar las consultas por título. Puedes hacerlo de la siguiente manera:

```
db.libros.createIndex({ titulo: 1 })
```

Este comando crea un índice ascendente en el campo `titulo`. Cuando realices consultas que involucren el campo `titulo`, MongoDB utilizará este índice para acelerar la búsqueda.

Consultas de agregación para operaciones complejas

Las consultas de agregación en MongoDB son una potente herramienta que te permite realizar operaciones complejas de análisis y transformación de datos en tus colecciones.

Estas operaciones son especialmente útiles cuando necesitas realizar cálculos, agrupaciones, filtrado y otras tareas de procesamiento de datos en tus documentos.

- Las consultas de agregación se realizan mediante el uso del framework de agregación de MongoDB, que proporciona una variedad de operadores y etapas para manipular los datos.

· Estas consultas son flexibles y versátiles, lo que las hace adecuadas para realizar tareas que van más allá de las consultas simples.

Operadores de Agregación

MongoDB ofrece una amplia gama de operadores de agregación que puedes utilizar en tus consultas. Algunos de los operadores comunes incluyen `\$match`, `\$group`, `\$sort`, `\$project`, `\$limit`, `\$skip`, `\$unwind`, entre otros. Estos operadores te permiten realizar tareas como filtrar documentos, agruparlos, ordenar resultados, proyectar campos específicos y más.

Pipeline de Agregación

En MongoDB, las consultas de agregación se construyen utilizando un "pipeline" que consiste en una secuencia de etapas de procesamiento de datos. Cada etapa toma los resultados de la etapa anterior y los modifica o transforma según las necesidades. Puedes encadenar múltiples etapas en el pipeline para crear consultas complejas y realizar operaciones avanzadas.

Ejemplo de Consulta de Agregación:

Supongamos que tienes una colección de ventas y deseas calcular el total de ventas por producto. Puedes utilizar una consulta de agregación para lograrlo:

```
db.ventas.aggregate([  
  { $group: { _id: "$producto", total: { $sum: "$monto" } } },  
  { $sort: { total: -1 } }  
])
```

En este ejemplo, la consulta de agregación primero agrupa las ventas por producto y luego suma los montos para cada producto. Luego, ordena los resultados en orden descendente según el total.

Consultas geoespaciales y uso de índices espaciales

Las consultas geoespaciales en MongoDB son una característica poderosa que te permite trabajar con datos geoespaciales, como coordenadas de ubicación (latitud y longitud). Esto es útil para aplicaciones que requieren funcionalidades basadas en la ubicación, como búsqueda de puntos de interés, mapas interactivos, seguimiento de vehículos y mucho más.

A continuación, se describen los conceptos clave relacionados con las consultas geoespaciales y el uso de índices espaciales en MongoDB:

Datos Geoespaciales en MongoDB

MongoDB admite datos geoespaciales a través de campos especiales que almacenan coordenadas. Los datos geoespaciales se pueden representar en dos formatos: GeoJSON y pares de coordenadas (latitud y longitud).

Las ubicaciones geoespaciales se almacenan en documentos de MongoDB y se pueden indexar para mejorar el rendimiento de las consultas.

Tipos de Datos Geoespaciales

MongoDB admite varios tipos de datos geoespaciales que se utilizan para representar diferentes formas geográficas y características.

Estos tipos incluyen Point (punto), LineString (línea), Polygon (polígono), MultiPoint (multipunto), MultiLineString (multilínea), MultiPolygon (multipolígono) y GeometryCollection (colección de geometrías).

Operadores Geoespaciales

MongoDB proporciona una variedad de operadores geoespaciales que puedes utilizar en tus consultas, como `$geoWithin`, `$geoIntersects`, `$near`, `$nearSphere`, entre otros.

Estos operadores te permiten realizar consultas específicas en función de la ubicación y la proximidad.

Índices Espaciales

Para acelerar las consultas geoespaciales, MongoDB permite la creación de índices espaciales. Estos índices mejoran el rendimiento al organizar los datos geoespaciales de manera eficiente.

Los índices espaciales se pueden crear en campos que almacenan datos geoespaciales, y MongoDB los utiliza para realizar consultas rápidas.

Ejemplo de Consulta Geoespacial

Supongamos que tienes una colección de ubicaciones de restaurantes y deseas encontrar los restaurantes que se encuentran a menos de 5 kilómetros de una ubicación específica. Puedes utilizar una consulta geoespacial para lograrlo:

```
db.restaurantes.find({
  ubicacion: {
    $near: {
      $geometry: {
        type: "Point",
```

```
    coordinates: [longitud, latitud]
  },
  $maxDistance: 5000 // Distancia en metros (5 km)
}
}
})
```

En este ejemplo, la consulta ``\$near` busca restaurantes cerca de la ubicación especificada, y ``\$maxDistance` define la distancia máxima.

Trabajo con fechas y consultas de rango de tiempo

El manejo de fechas y consultas de rango de tiempo en MongoDB es fundamental cuando se trata de aplicaciones que requieren gestionar datos temporales, como registros de eventos, seguimiento de actividades o cualquier otro tipo de información relacionada con fechas.

MongoDB permite almacenar fechas en campos específicos dentro de documentos. Esto es especialmente útil para llevar un registro de cuándo se crearon o actualizaron documentos. Luego, puedes realizar consultas para recuperar documentos que caen dentro de un rango de tiempo específico.

Para realizar consultas de rango de tiempo en MongoDB, puedes utilizar diversos operadores y funciones. Algunos de los operadores más comunes incluyen:

- **``\$date`**: Te permite representar una fecha en un documento.
- **``\$gte` (mayor o igual que)**: Encuentra documentos con fechas mayores o iguales a la fecha especificada.
- **``\$lte` (menor o igual que)**: Encuentra documentos con fechas menores o iguales a la fecha especificada.
- **``\$gt` (mayor que)**: Encuentra documentos con fechas mayores que la fecha especificada.
- **``\$lt` (menor que)**: Encuentra documentos con fechas menores que la fecha especificada.
- **``\$and`**: Te permite combinar múltiples condiciones de fecha para realizar consultas más complejas.

Estos operadores te brindan la flexibilidad necesaria para recuperar datos dentro de un rango de tiempo particular o realizar cálculos relacionados con fechas.

A continuación haremos un ejemplo sencillo de cómo realizar consultas de rango de tiempo en MongoDB utilizando el shell de MongoDB.

Supongamos que tienes una colección llamada "eventos" con documentos que contienen un campo "fecha" que almacena la fecha y hora de cada evento.

// Consulta para encontrar eventos ocurridos en un rango de tiempo específico

```
db.eventos.find({
  fecha: {
    $gte: ISODate("2023-01-01T00:00:00Z"), // Mayor o igual a esta fecha
    $lt: ISODate("2023-02-01T00:00:00Z") // Menor que esta fecha
  }
})
```

- ``$gte`` se usa para buscar eventos con fechas mayores o iguales a "2023-01-01T00:00:00Z".
- ``$lt`` se utiliza para buscar eventos con fechas menores que "2023-02-01T00:00:00Z".

Puedes personalizar las fechas y la consulta según tus necesidades específicas. Esta consulta recuperará todos los eventos que ocurrieron en enero de 2023, por ejemplo. Ten en cuenta que debes ajustar las fechas y los nombres de las colecciones a tu configuración real de MongoDB.

Transacciones y operaciones atómicas en MongoDB

En MongoDB, las transacciones y las operaciones atómicas son esenciales para mantener la integridad de los datos y garantizar que las modificaciones se realicen de manera segura y confiable. Las transacciones permiten agrupar una serie de operaciones en una unidad lógica y garantizan que todas se ejecuten correctamente o que ninguna lo haga, lo que mantiene la coherencia de los datos.

Aquí hay una descripción general de cómo funcionan las transacciones en MongoDB:

Múltiples Operaciones en una Transacción

Puedes agrupar varias operaciones de lectura y escritura en una transacción. Esto es útil cuando necesitas realizar cambios en varios documentos y deseas que todas las operaciones tengan éxito o ninguna se ejecute.

Operaciones Atómicas

Todas las operaciones dentro de una transacción se consideran atómicas, lo que significa que se ejecutan en su totalidad o no se ejecutan en absoluto. Si ocurre un error en cualquier parte de la transacción, MongoDB se encargará de deshacer todas las operaciones anteriores.

Rollbacks

Si una operación dentro de la transacción falla o se revierte, MongoDB realiza un rollback para garantizar que los datos vuelvan a su estado original.

Consistencia de Datos

Las transacciones en MongoDB garantizan que los datos sean coherentes antes y después de su ejecución. Esto es especialmente importante en situaciones de alta concurrencia y para mantener la integridad de la base de datos.

Aislamiento

Las transacciones ofrecen niveles de aislamiento, lo que significa que puedes controlar cómo las operaciones de una transacción interactúan con las operaciones concurrentes de otros usuarios.

Para utilizar transacciones en MongoDB, debes utilizar un controlador de base de datos que las admita, como el controlador oficial de MongoDB para Node.js. A continuación, te muestro un ejemplo simple de cómo usar transacciones en MongoDB con Node.js:

```
const session = client.startSession();

session.startTransaction();

try {
  await collection1.updateOne({ _id: 1 }, { $set: { balance: 500 } });
  await collection2.updateOne({ _id: 2 }, { $set: { balance: 600 } });

  // Si no hay errores, confirma la transacción
  await session.commitTransaction();
} catch (error) {
  // Si ocurre un error, realiza un rollback
  await session.abortTransaction();
} finally {
  session.endSession();
}
```

Este ejemplo inicia una transacción, realiza actualizaciones en dos colecciones diferentes y confirma la transacción si no hay errores. Si se produce un error, se revierten las operaciones en ambas colecciones.

Recuerda ajustar el código según tus necesidades específicas y la configuración de tu aplicación. Las transacciones son una herramienta poderosa para garantizar la integridad de los datos en MongoDB.

Conclusión

En esta unidad, hemos abordado temas esenciales relacionados con MongoDB, desde la flexibilidad del modelado de datos hasta la realización de consultas avanzadas con operadores y proyecciones.

Aprendimos a optimizar el rendimiento de las consultas mediante el uso de índices y a realizar operaciones complejas con consultas de agregación. También exploramos el poder de MongoDB en el manejo de datos geoespaciales y cómo trabajar con fechas y consultas de rango de tiempo. Finalmente, nos sumergimos en la importancia de las transacciones y operaciones atómicas en MongoDB.

Bibliografía utilizada y sugerida

- ¿Qué es el modelado de datos? | IBM. (s. f.). <https://www.ibm.com/mx-es/topics/data-modeling>
- Mejores prácticas de seguridad de bases de datos. (s. f.-b). <https://www.oracle.com/ar/database/nosql/what-is-nosql/>
- MongoDB. (s. f.-a). MongoDB Architecture. <https://www.mongodb.com/es/mongodb-architecture>
- MongoDB CRUD Operations — MongoDB Manual. (s. f.). <https://www.mongodb.com/docs/manual/crud/>
- MongoDB query operators. (s. f.). https://www.w3schools.com/mongodb/mongodb_query_operators.php