MongoDB

1 Operaciones CRUD de MongoDB

MongoDB ofrece un robusto conjunto de operaciones CRUD (Create, Read, Update, Delete) que son fundamentales para gestionar y manipular datos en este sistema de bases de datos NoSQL. En esta sección, profundizaremos en los conceptos básicos de las operaciones CRUD.

1.1 Introducción al CRUD

Las operaciones CRUD son la forma principal de interactuar con MongoDB. Estas operaciones permiten a los usuarios crear, recuperar, actualizar y eliminar datos en las colecciones de MongoDB. La flexibilidad de MongoDB permite realizar operaciones CRUD sobre documentos sin requerir un esquema fijo.

1.2 Crear (C en CRUD)

La operación "Create", como su nombre indica, se utiliza para añadir nuevos documentos a una colección. MongoDB proporciona dos métodos principales para crear documentos: **insertOne**() e **insertMany**(). Estos métodos permiten insertar documentos individuales o múltiples documentos en una colección.

```
db.students.insertOne({
name: "Alice",
age: 22,
major: "Biology",
enrolled: true
});
```

En este ejemplo, estamos añadiendo la información de un nuevo estudiante a una colección.

1.3 Leer (R en CRUD)

La operación "Leer" implica recuperar datos de colecciones MongoDB. MongoDB proporciona potentes capacidades de consulta para encontrar documentos específicos que coincidan con ciertos criterios. El método principal utilizado para leer datos es **find**().

Ejemplo de lectura de documentos:

```
db.students.find({ major: "Computer Science" });
```

Esta consulta recupera todos los documentos de la colección "estudiantes" cuya especialidad es "Informática".

1.4 Actualizar (U en CRUD)

La operación "Update" permite modificar documentos existentes. MongoDB ofrece métodos como **updateOne**() y **updateMany**() para actualizar uno o varios documentos con nuevos datos.

Ejemplo de actualización de un documento:

```
db.students.updateOne( { name: "Alice" }, { $set: { age: 23 } } );
```

En este ejemplo, estamos actualizando la edad de una estudiante llamada "Alice" a 23 años.

1.5 Suprimir (D en CRUD)

La operación "Delete" consiste en eliminar documentos de una colección. MongoDB ofrece métodos como **deleteOne**() y **deleteMany**() para este propósito.

Ejemplo de eliminación de un documento:

```
db.students.deleteOne({ name: "Alice" });
```

Esta consulta elimina un documento con el nombre "Alice" de la colección "students".

2 Aggregation Pipelines

Los Aggregation Pipelines de MongoDB son una potente herramienta para transformar y analizar datos de forma flexible y modular. Las canalizaciones de agregación proporcionan los medios para filtrar, remodelar y obtener información de sus datos. En esta sección, exploraremos los conceptos fundamentales de las canalizaciones de agregación.

2.1 El poder de la agregación

Los procesos de agregación permiten realizar transformaciones de datos en varias etapas. Puede considerar cada etapa como un paso de filtrado o transformación aplicado a los documentos de la canalización. Estas etapas pueden combinarse para crear flujos de trabajo de datos complejos.

2.2 Etapas clave de agregación

El marco de trabajo de agregación de MongoDB ofrece una variedad de etapas, entre las que se incluyen:

- **$match:** Filtra documentos basándose en criterios especificados.
- **$group:** Agrupa documentos por un campo concreto y realiza cálculos sobre los datos agrupados.
- **$project:** Da forma a los documentos seleccionando, excluyendo o transformando campos.
- **$sort:** Ordena los documentos en función de uno o varios campos.
- **$limit:** Restringe el número de documentos en la salida del pipeline.

2.3 La etapa $match

La etapa $match en los procesos de agregación de MongoDB se utiliza para filtrar documentos basándose en criterios específicos. Esta etapa es análoga a la cláusula WHERE de SQL y permite seleccionar sólo los documentos que coincidan con las condiciones definidas.

La sintaxis básica de la etapa $match es la siguiente:

```
{ $match: { field: value } }
```

Aquí, se especifica el campo que desea filtrar y el valor que debe coincidir. Los documentos que cumplan esta condición pasarán a la siguiente etapa del proceso.

Supongamos que tiene una colección de "productos" y desea encontrar todos los productos con un precio inferior a 50 dólares. A continuación se muestra cómo puede utilizar la etapa $match para conseguirlo:

```
db.products.aggregate([ { $match: { price: { $lt: 50 } } } ]);
```

Este proceso de agregación sólo pasará documentos con un campo "precio" inferior a 50 a la siguiente etapa.

2.4 La etapa $group

La etapa $group en los procesos de agregación de MongoDB se utiliza para agrupar documentos basándose en un campo específico y realizar cálculos sobre los datos agrupados. Esta etapa es particularmente útil para el análisis y resumen de datos.

La sintaxis básica de la etapa $group es la siguiente:

```
{ $group: { _id: "$field", newField: { $function: "$field" } } }
```

Aquí se especifica el campo por el que se quieren agrupar los documentos y se definen los nuevos campos que resultan de los cálculos o agregaciones sobre los datos agrupados.

Supongamos que tiene una colección de "ventas" y desea calcular los ingresos totales de cada categoría de producto. A continuación te mostramos cómo puedes utilizar la etapa $group para conseguirlo:

```
db.sales.aggregate([
  {
    $group: {
      _id: "$productCategory",
      totalRevenue: { $sum: "$price" }
    }
  }
]);
```

Esta canalización de agregación agrupará las ventas por "productCategory" y calculará los ingresos totales de cada categoría.

2.5 La etapa $project

La etapa $project en las canalizaciones de agregación de MongoDB se utiliza para remodelar y transformar documentos seleccionando, excluyendo o creando nuevos campos. Es una etapa versátil que permite personalizar la estructura de los documentos en la salida de la canalización.

La sintaxis básica de la etapa $project es la siguiente:

```
{ $project: { campo1: 1, campo2: 1, newCampo: expresión, _id: 0 } }
```

Puede especificar qué campos incluir (1) o excluir (0) en la salida. Además, puede utilizar expresiones para crear nuevos campos.

Suponga que tiene una colección de "empleados" y desea proyectar sólo sus nombres y edades en la salida. Así es como puedes utilizar la etapa $project:

```
db.empleados.aggregate([
  {
    $project: {
      nombre: 1
      edad: 1,
      _id: 0
    }
  }
]);
```

Este proceso de agregación producirá documentos con sólo los campos "nombre" y "edad" y excluirá el campo por defecto "_id".

2.6 La etapa $sort

La etapa $sort en las cadenas de agregación de MongoDB se utiliza para ordenar documentos basándose en uno o más campos. La ordenación de documentos puede ser esencial para presentar los datos de forma significativa o prepararlos para su posterior procesamiento.

La sintaxis básica de la etapa $sort es la siguiente:

```
{ $sort: { campo1: 1, campo2: -1 } }
```

Especifica los campos por los que ordenar los documentos y su orden de clasificación. Utilice 1 para orden ascendente y -1 para orden descendente.

Suponga que tiene una colección de "productos" y desea ordenarlos por precio en orden descendente. Así es como puedes utilizar la etapa $sort:

```
db.products.aggregate([
  {
    $sort: { precio: -1 }
  }
]);
```

Este proceso de agregación devolverá la colección "products" con los documentos ordenados por precio en orden descendente.

2.7 La etapa $limit

La etapa $limit en las canalizaciones de agregación de MongoDB se utiliza para restringir el número de documentos en la salida de la canalización. Es particularmente útil para limitar la cantidad de datos devueltos por una consulta.

La sintaxis básica de la etapa $limit es la siguiente:

```
{ $limit: 5 }
```

Se especifica el número de documentos que se desea incluir en la salida.

Suponga que tiene una colección de "pedidos" y desea recuperar sólo los cinco primeros pedidos de la colección. A continuación se muestra cómo puede utilizar la etapa $limit:

```
db.pedidos.agregar([
  {
    $limit: 5
  }
]);
```

Este proceso de agregación devolverá los cinco primeros documentos de la colección "orders".

En esta sección, hemos cubierto las etapas esenciales de las canalizaciones de agregación de MongoDB. Estas etapas pueden combinarse de varias formas para realizar transformaciones y análisis de datos complejos.

3 Modelado de datos

El modelado de datos en MongoDB es un aspecto crucial del diseño de bases de datos, ya que asegura que sus datos estén estructurados de forma que soporten consultas y recuperaciones eficientes. En esta sección, exploraremos los conceptos fundamentales del modelado de datos en MongoDB.

3.1 Datos orientados a documentos

MongoDB es una base de datos NoSQL orientada a documentos. Esto significa que los datos se almacenan en documentos BSON (Binary JSON), y cada documento puede tener una estructura diferente. Esta flexibilidad le permite diseñar sus modelos de datos basándose en los requisitos específicos de su aplicación.

3.2 Documentos incrustados y referencias

Las decisiones de modelado de datos a menudo giran en torno a si incrustar datos relacionados dentro de un documento o utilizar referencias para enlazar documentos. Los documentos incrustados son adecuados para datos a los que se accede frecuentemente de forma conjunta, mientras que las referencias se utilizan para crear relaciones entre documentos.

3.3 Mejores prácticas para el diseño de esquemas

Cuando diseñe su esquema MongoDB, tenga en cuenta las siguientes buenas prácticas:

- **Optimice para patrones de consulta:** Estructure sus datos para que coincidan con la forma en que su aplicación los consulta. Esto puede implicar la desnormalización o la creación de colecciones adicionales.
- **Evite el anidamiento profundo:** Limite la profundidad de los documentos incrustados para evitar consultas complejas y garantizar que sus datos sigan siendo accesibles.
- **Utilice índices:** Cree índices apropiados para acelerar el rendimiento de las consultas, especialmente para los campos a los que se accede con frecuencia.
- **Desnormalice cuando sea necesario:** En algunos casos, la duplicación de datos (desnormalización) puede mejorar el rendimiento de las consultas.

3.4 Documentos incrustados

Los documentos incrustados son una potente característica de MongoDB que permite almacenar datos relacionados dentro de un único documento. Este enfoque es útil cuando se tienen relaciones de uno a muchos y se desea recuperar datos relacionados en una única consulta.

Supongamos que tiene una aplicación de comercio electrónico y desea almacenar información sobre productos junto con las opiniones de los clientes. En lugar de crear colecciones separadas para productos y opiniones, puedes incrustar las opiniones dentro del documento del producto.

```
{
  _id: 1,
  nombre: "Smartphone",
  precio: 499.99,
  comentarios: [
    { nombre de usuario: "Usuario1", valoración: 4, comentario:
"¡Gran producto!" },
    { nombre de usuario: "Usuario2", valoración: 5, comentario:
"¡Excelente calidad!" }
  ]
}
```

Esta estructura permite recuperar tanto los detalles del producto como las reseñas asociadas con una sola consulta.

Ventajas de los documentos incrustados

- Eficaz para relaciones uno a muchos.
- Reduce la necesidad de uniones complejas.
- Operaciones de lectura más rápidas.

3.5 Referencias

Las referencias en el modelado de datos de MongoDB implican almacenar una referencia al campo _id de otro documento en lugar de incrustar los datos relacionados. Este enfoque es útil para manejar relaciones de muchos a muchos y simplificar las estructuras de los documentos.

Ejemplo de uso de referencias

Supongamos que tiene una aplicación de blog y desea asociar comentarios a las entradas del blog. En lugar de incrustar los comentarios en el documento de la entrada del blog, puede almacenar referencias a los documentos de comentarios.

```
{
  _id: 1,
  title: "Introducción a MongoDB",
  content: "MongoDB es una base de datos NoSQL flexible...",
  comentarios: [ObjectId("comment1"), ObjectId("comment2")]
}
```

Documento de comentarios:

```
{
  _id: ObjectId("comentario1"),
  text: "¡Gran artículo!",
  user: "Usuario1"
}
```

Mediante las referencias, puede recuperar los comentarios de una entrada de blog consultando los documentos de comentarios referenciados.

Ventajas de las referencias

- Adecuadas para relaciones de muchos a muchos.
- Simplifica las estructuras de los documentos.
- Reduce la duplicación de datos.

4 Mejores prácticas de diseño de esquemas

Un modelado de datos efectivo en MongoDB implica adherirse a las mejores prácticas que aseguren que sus datos están estructurados para una consulta y recuperación eficientes. Exploremos estas prácticas en detalle:

4.1 Optimización de patrones de consulta

Analice cómo su aplicación consulta los datos: Diseñe su modelo de datos para que coincida con la forma en que su aplicación recupera la información. Tenga en cuenta los patrones de consulta más habituales y estructure los datos en consecuencia.

Utilice la desnormalización cuando sea necesario: En algunos casos, la duplicación de datos (desnormalización) puede mejorar el rendimiento de las consultas al reducir la necesidad de realizar uniones complejas.

4.2 Evitar el anidamiento profundo

Limite la profundidad de los documentos incrustados: Los documentos anidados en profundidad pueden complicar las consultas y reducir la accesibilidad de los datos. Busque un equilibrio entre la organización de los datos y la eficacia de las consultas.

Utilice referencias para los datos muy anidados: Cuando se tienen jerarquías profundas de datos, puede ser más práctico utilizar referencias en lugar de incrustaciones profundas.

4.3 Utilizar índices

Cree índices adecuados: Los índices son esenciales para acelerar el rendimiento de las consultas. Identifique los campos que se consultan con frecuencia y cree índices sobre ellos. Tenga cuidado de no crear demasiados índices, ya que esto puede afectar al rendimiento de escritura.

4.4 Desnormalización cuando sea necesario

Desnormalice para cargas de trabajo de lectura intensiva: En situaciones en las que las cargas de trabajo sean de lectura intensiva, considere la posibilidad de

desnormalizar los datos para reducir la necesidad de realizar uniones y agregaciones complejas.

Considere la compensación: la desnormalización puede mejorar el rendimiento de lectura, pero puede complicar las operaciones de escritura y aumentar los requisitos de almacenamiento. Sopese los pros y los contras en función de su caso de uso específico.

4.5 Mantener la coherencia de los datos

Utilice transacciones y operaciones multi-documento: MongoDB proporciona soporte para transacciones multi-documento, que pueden ayudar a mantener la consistencia de los datos en escenarios en los que es necesario coordinar múltiples operaciones.

5 Validación de esquemas en MongoDB

La validación de esquemas en MongoDB es una valiosa característica que le permite definir la estructura y las reglas para sus documentos, asegurando la consistencia e integridad de los datos.

5.1 Reforzar la consistencia de los datos

La validación de esquemas impone reglas sobre los datos que pueden insertarse o actualizarse en una colección. Ayuda a mantener la coherencia y la calidad de los datos garantizando que los documentos cumplen las restricciones predefinidas.

5.2 Validación de esquemas JSON

Supongamos que tenemos una colección de "empleados" y quiere asegurarse de que cada documento incluye un campo "nombre" como cadena y un campo "edad" como entero positivo. Puede definir un esquema JSON para aplicar estas reglas

```
db.createCollection("employees", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "age"],
      properties: {
```

```
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        age: {
          bsonType: "int",
          minimum: 0,
          description: "must be a positive integer and is required"
        }
      }
    }
  }
});
```

6 Índices

Los índices juegan un papel fundamental en el modelado de datos de MongoDB, ya que mejoran significativamente el rendimiento de las consultas y la recuperación de datos. En esta sección, exploraremos la importancia de los índices en una base de datos MongoDB.

6.1 ¿Qué son los índices?

Los índices en MongoDB son estructuras de datos que almacenan una pequeña porción de los datos de la colección, permitiendo al sistema de base de datos localizar documentos rápidamente. Sin índices, MongoDB necesitaría escanear toda la colección para encontrar los datos que busca, lo que puede llevar mucho tiempo para grandes conjuntos de datos.

6.2 Tipos comunes de índices

MongoDB soporta varios tipos de índices para satisfacer diferentes necesidades de modelado de datos:

Índice de campo único:
Supongamos que tiene una colección de libros y a menudo necesita buscarlos por su título. Puedes crear un índice de campo único en el campo "título" de la siguiente manera:

```
db.books.createIndex({ título: 1 });
```

Este índice acelerará significativamente las consultas que impliquen la búsqueda de libros por sus títulos. El 1 indica que el índice tiene orden ascendente.

Índice compuesto:
Supongamos que tienes una colección de perfiles de usuario, y que a menudo se consulta a los usuarios en función tanto de su "país" como de su "edad". Puedes crear un índice compuesto en ambos campos de la siguiente manera:

```
db.userProfiles.createIndex({ país: 1, edad: 1 });
```

Este índice compuesto es útil para optimizar las consultas que filtran usuarios tanto por su país como por su edad.

Índice de texto:
Si tienes una colección de artículos y quieres realizar búsquedas de texto completo en el campo "contenido", puedes crear un índice de texto como el siguiente

```
db.articles.createIndex({ content: "text" });
```

Un índice de texto permite realizar una búsqueda de texto completo eficaz, facilitando la búsqueda de artículos basada en el texto de su contenido.

Índice geoespacial:
Supongamos que está creando un servicio basado en la localización y almacena las ubicaciones de los restaurantes con coordenadas de latitud y longitud. Puedes crear un índice geoespacial como este

```
db.restaurants.createIndex({ location: "2dsphere" });
```

Este índice te permite realizar consultas geoespaciales, como encontrar restaurantes cerca de una ubicación específica.

Índice comodín:
Un índice comodín indexa todos los campos de un documento. Esto puede ser beneficioso en situaciones en las que se tienen varias consultas con diferentes campos. Para crear un índice comodín, puedes utilizar el carácter comodín especial "$**":

```
db.collection.createIndex({ "$**": "text" });
```

Este índice permite realizar consultas flexibles sobre cualquier campo de los documentos.

6.3 Elegir los índices adecuados

Seleccionar los índices adecuados es crucial para optimizar el rendimiento de las consultas. Tenga en cuenta lo siguiente a la hora de elegir qué campos indexar:

- **Frecuencia de consulta:** Indexe campos que se utilicen con frecuencia en las consultas.
- **Índices compuestos:** Para consultas con varias condiciones, cree índices compuestos en los campos utilizados en esas condiciones.
- **Campos de ordenación y agregación:** Indexe campos que ordene o agregue con frecuencia.

- **Utilice índices comodín:** En situaciones en las que tenga una variedad de consultas y no conozca los campos exactos de antemano, los índices comodín pueden ser una solución flexible.

6.4 Índices y rendimiento de escritura

Aunque los índices mejoran significativamente el rendimiento de las consultas, pueden afectar al rendimiento de escritura. Cada índice creado añade sobrecarga a las operaciones de escritura, por lo que es esencial equilibrar los beneficios de la indexación con las necesidades de su aplicación.

7 Temas avanzados de MongoDB

MongoDB es un sistema de gestión de bases de datos versátil y completo que ofrece una amplia gama de características y capacidades más allá de los conceptos fundamentales que hemos cubierto.

7.1 Consultas geoespaciales

MongoDB ofrece un sólido soporte para datos y consultas geoespaciales, lo que lo convierte en la opción perfecta para aplicaciones basadas en localización. Puede almacenar coordenadas geográficas y realizar consultas como encontrar las ubicaciones más cercanas o ubicaciones dentro de un área específica.

7.2 GridFS

GridFS de MongoDB es una especificación para almacenar y recuperar archivos de gran tamaño, como imágenes, vídeos y documentos, en la base de datos. GridFS permite gestionar y servir de forma eficiente archivos de gran tamaño sin las limitaciones del tamaño de los documentos.

7.3 Replicación y fragmentación

MongoDB ofrece replicación y fragmentación para garantizar una alta disponibilidad y escalabilidad. La replicación implica la creación de múltiples copias de sus datos para la tolerancia a fallos, mientras que la fragmentación divide sus datos en múltiples servidores para la escalabilidad horizontal.

7.4 Seguridad

La seguridad es una prioridad en MongoDB. Ofrece características como autenticación, autorización y encriptación para proteger tus datos. Es crucial configurar los parámetros de seguridad para salvaguardar tu despliegue de MongoDB.

7.5 Búsqueda de texto completo

MongoDB también soporta capacidades de búsqueda de texto completo, lo que le permite buscar y clasificar datos de texto de manera eficiente.

7.6 Integración

MongoDB puede integrarse fácilmente con varios lenguajes y herramientas de programación. Dispone de controladores oficiales para lenguajes como Python, Node.js y Java, lo que lo hace accesible para desarrolladores de diferentes ecosistemas.