



mongoDB

MongoDB

04.09.20XX

—

Tu nombre

Tu empresa

Tu calle 123

Tu ciudad, Provincia X1234XXX

Introducción a MongoDB	2
MongoDB	2
Comparación entre MongoDB y SQL	2
Usos y características	2
Documentos, Colecciones y Bases de Datos	3
Modelo de Documentos en MongoDB	3
Esquema	4
Sintaxis	4
Modelado de Datos en MongoDB	4
Diferencias entre bases de datos relacionales y no relacionales	5
Principios de MongoDB	5
Relaciones comunes	5
Formas de modelar datos (Embedding, Referencing)	7
Escalabilidad de un modelo de datos	8
Conexión a una base de datos MongoDB	8
Obtención de la Cadena de Conexión(Connection String)	9
Formato de la Cadena de Conexión URI (Connection String URI)	9
Operaciones CRUD en MongoDB	9
Inserción de documentos en una colección MongoDB (Create)	9
Búsqueda de documentos en una colección MongoDB (Read)	11
Actualización de documentos (Update)	12
Eliminación de documentos (Delete)	14
Aggregation en MongoDB	15
Definiciones Clave y Estructura	15
Stages mas usadas comúnmente y ejemplos	16
Índices en MongoDB en Colecciones	18
Tipos de índices más comunes	19
Otros métodos para realizar sobre índices	20
Eliminación de índices	21
Material de Presentación	22
Bibliografía	22

Introducción a MongoDB

MongoDB es una base de datos NoSQL ampliamente utilizada que se destaca por su flexibilidad y capacidad para gestionar datos no estructurados y semiestructurados. En esta introducción, exploraremos MongoDB y sus diferencias con las bases de datos SQL tradicionales. Además, discutiremos los usos, características clave y términos asociados con esta plataforma.

MongoDB

MongoDB es una base de datos NoSQL que se basa en el modelo de documentos. En contraste con las bases de datos SQL que utilizan tablas con filas y columnas para almacenar datos, MongoDB utiliza documentos JSON para representar la información. Cada documento se asemeja a un objeto JSON y puede contener datos de diferentes tipos sin un esquema estricto.

Comparación entre MongoDB y SQL

Una de las principales diferencias entre MongoDB y las bases de datos SQL radica en su modelo de datos. Mientras que las bases de datos SQL almacenan datos en tablas con filas y columnas, MongoDB utiliza documentos JSON. Esta distinción es fundamental ya que proporciona ventajas significativas en términos de:

- json objects vs. rows and columns: En MongoDB, los datos se almacenan en objetos JSON, lo que permite una representación más flexible y versátil en comparación con las tablas de SQL.

Usos y características

MongoDB se adapta a una variedad de casos de uso, desde proyectos educativos personales hasta aplicaciones empresariales. Algunos de los escenarios comunes donde MongoDB brilla incluyen:

- Personal educational projects: MongoDB es una elección popular para proyectos educativos personales debido a su facilidad de uso y flexibilidad.
- Start-ups: Muchas startups optan por MongoDB debido a su capacidad de escalar y adaptarse a las cambiantes necesidades de negocio.
- Enterprise applications: MongoDB también se utiliza en aplicaciones empresariales que requieren manejar grandes volúmenes de datos de manera eficiente.

Ejemplos:

- E-commerce: Para gestionar el catálogo de productos y los datos del cliente.
- Content management: Para crear sistemas de gestión de contenido ágiles.
- IoT and time-series data: Ideal para capturar datos generados por dispositivos IoT y datos de series temporales.
- Trading and Payments: Se utiliza en aplicaciones financieras para gestionar transacciones.
- Gaming: En juegos en línea para administrar perfiles de jugadores y puntuaciones.
- Mobile Apps: Para almacenar datos de aplicaciones móviles y proporcionar una experiencia rápida.
- Real-time analytics and AI: Permite el análisis de datos en tiempo real y aplicaciones de inteligencia artificial.

Algunas de las características clave que hacen que MongoDB sea atractivo incluyen:

- Escalabilidad: MongoDB es altamente escalable y puede manejar grandes cantidades de datos y tráfico.
- Resiliencia: Ofrece capacidades de replicación y recuperación ante fallos para garantizar la disponibilidad de los datos.
- Velocidad de desarrollo: La flexibilidad del modelo de datos permite un desarrollo ágil y rápido de aplicaciones.
- Privacidad y Seguridad: MongoDB ofrece características de seguridad avanzada, como autenticación y autorización.

Documentos, Colecciones y Bases de Datos

En MongoDB, los documentos son la unidad básica de datos y se asemejan a objetos JSON. Estos documentos se agrupan en colecciones, y las colecciones a su vez se almacenan en bases de datos. Esta estructura jerárquica permite una organización eficiente de los datos y facilita el acceso a la información.

Modelo de Documentos en MongoDB

MongoDB admite una variedad de tipos de datos que se pueden almacenar en documentos. Esto incluye tipos de datos JSON como cadenas de texto, números, booleanos, arreglos, objetos anidados, fechas (Date), y otros tipos de datos específicos de MongoDB, como ObjectID, BinData y Decimal128. Los tipos de datos de MongoDB son fundamentales para el almacenamiento de datos heterogéneos y complejos

Esquema

El esquema en MongoDB es flexible y dinámico. A diferencia de las bases de datos relacionales, no es necesario definir un esquema rígido antes de insertar datos. Puedes modificar el esquema de tus documentos en cualquier momento, lo que permite la adición de nuevos campos sin requerir cambios en la estructura de documentos existentes. Además, MongoDB ofrece la opción de validar esquemas de documentos de manera opcional, lo que te permite establecer restricciones en la estructura de los documentos si lo deseas.

Sintaxis

La sintaxis de los documentos en MongoDB se basa en la notación JSON (JavaScript Object Notation). Los documentos se representan con pares clave-valor, donde cada campo (clave) tiene un valor asociado. La sintaxis es intuitiva y fácil de entender. Aquí hay un ejemplo de cómo se ven los documentos en MongoDB:

```
{  
  "clave1": "valor1",  
  "clave2": "valor2",  
  "clave3": {  
    "subclave1": "valor3",  
    "subclave2": "valor4"  
  }  
}
```

Es importante destacar que MongoDB no requiere que todos los documentos en una colección tengan la misma estructura, lo que permite la existencia de documentos polimórficos que pueden contener campos diferentes.

Modelado de Datos en MongoDB

El proceso de modelado de datos en MongoDB difiere significativamente de las bases de datos relacionales. Mientras que en las bases de datos relacionales se siguen pasos como obtener los requisitos, modelar los datos y luego entregarlos al desarrollador para manipularlos, MongoDB adopta un enfoque más flexible y orientado a la interacción del usuario con la aplicación.

Diferencias entre bases de datos relacionales y no relacionales

En las bases de datos relacionales, se obtienen los requisitos y se modelan los datos de acuerdo con una estructura rígida, que a menudo se traduce en tablas y relaciones predefinidas. En contraste, MongoDB se adapta a los requisitos del usuario, centrándose en cómo el usuario interactuará con la aplicación en lugar de imponer una estructura de datos rígida. Esta flexibilidad se debe al modelo de documento en MongoDB, que no impone una estructura de documento específica por defecto, permitiendo la polimorfía, lo que significa que documentos incluso dentro de una misma colección pueden tener estructuras diferentes.

Principios de MongoDB

Los principios fundamentales de MongoDB son:

- 1. Datos que se acceden juntos deben almacenarse juntos: Este principio enfatiza la importancia de almacenar datos relacionados en un solo documento para mejorar el rendimiento y la eficiencia de las consultas.
- 2. Modelo de documento flexible: MongoDB permite la flexibilidad en la estructura de los documentos. A diferencia de las bases de datos relacionales, las colecciones en MongoDB no imponen ninguna estructura específica a los documentos, lo que permite la polimorfía y la adaptación a las necesidades cambiantes.
- 3. Enfocarse en cómo se usará la información: En lugar de enfocarse en cómo se almacena la información en la base de datos, MongoDB pone un énfasis particular en cómo se utilizará la información en la aplicación. Esto permite un diseño de datos más orientado al caso de uso específico.

Relaciones comunes

En MongoDB, las relaciones entre datos se pueden modelar de varias maneras, dependiendo de las necesidades de tu aplicación. A continuación, se presentan ejemplos de relaciones comunes:

RELACIÓN	Referencing (Referencia)	Embedding (Incrustación)
One-to-One	<pre>{ "_id": ObjectId("12345"), "nombre": "Juan", "perfil_id": ObjectId("54321") }</pre>	<pre>{ "_id": ObjectId("12345"), "nombre": "Juan", "perfil": { "nombre": "Perfil de Juan",</pre>

	<pre>{ "_id": ObjectId("54321"), "nombre": "Perfil de Juan", "email": "juan@email.com" }</pre>	<pre> "email": "juan@email.com" }</pre>
One-to-Many	<pre>{ "_id": ObjectId("12345"), "nombre": "Juan", "comentarios": [ObjectId("98765"), ObjectId("87654")] }</pre> <pre>{ "_id": ObjectId("98765"), "texto": "¡Gran publicación!", "usuario_id": ObjectId("12345") }</pre> <pre>{ "_id": ObjectId("87654"), "texto": "Interesante artículo", "usuario_id": ObjectId("12345") }</pre>	<pre>{ "_id": ObjectId("12345"), "nombre": "Juan", "comentarios": [{ "texto": "¡Gran publicación!" }, { "texto": "Interesante artículo" }] }</pre>
Many-to-Many	<pre>{ "_id": ObjectId("111"), "nombre": "Ana", "cursos_inscritos": [ObjectId("999"), ObjectId("888")] }</pre> <pre>{</pre>	<pre> "_id": ObjectId("111"), "nombre": "Ana", "cursos_inscritos": [{ "nombre": "Matemáticas" }, { "nombre": "Historia" }] }</pre>

	<pre> "_id": ObjectId("999"), "nombre": "Matemáticas", "estudiantes_inscritos": [ObjectId("111"), ObjectId("222")] } { "_id": ObjectId("888"), "nombre": "Historia", "estudiantes_inscritos": [ObjectId("111"), ObjectId("333")] } </pre>	<pre> }] } { "_id": ObjectId("999"), "nombre": "Matemáticas", "estudiantes_inscritos": [{ "nombre": "Ana" }, { "nombre": "Eva" }] } { "_id": ObjectId("888"), "nombre": "Historia", "estudiantes_inscritos": [{ "nombre": "Ana" }, { "nombre": "Carlos" }] } </pre>
--	--	--

Formas de modelar datos (Embedding, Referencing)

- (Embedding) Incrustación de datos en documentos

Se utiliza cuando hay relaciones de uno a muchos(1:Many) o muchos a muchos(Many:Many) en los datos que se almacenan.

Ventajas:

- Simplifica las consultas y mejora el rendimiento general de las consultas.

- Los documentos incrustados son documentos anidados, lo que cumple con el principio de que los datos accedidos juntos deben almacenarse juntos.

Desventajas:

- Puede generar documentos grandes y sin límites, lo que podría exceder el límite de documento BSON de 16 MB.
- (Referencing) Referencia de datos en documentos

A veces, es preferible almacenar información relacionada en documentos separados o incluso en colecciones separadas.

Ventajas:

- No hay duplicación de datos.
- Los documentos son más pequeños.

Desventajas:

- Realizar consultas desde múltiples documentos consume recursos adicionales y afecta el rendimiento de lectura.

Escalabilidad de un modelo de datos

Cuando creas un modelo de datos en MongoDB, es fundamental considerar la escalabilidad para lograr la máxima eficiencia en cuanto a tiempos de respuesta de consultas, uso de memoria, uso de CPU y almacenamiento. Algunos principios clave son:

- El principio fundamental de que los datos accedidos juntos deben almacenarse juntos.
- Evitar documentos sin límites (documentos que crecen infinitamente), ya que pueden exceder el límite de documento BSON de 16 MB y causar problemas de rendimiento y uso excesivo de memoria.
- Tener cuidado con los documentos incrustados, ya que si superan el límite de 16 MB, pueden afectar negativamente el rendimiento de escritura y lectura.

Asegúrate de optimizar tu modelo de datos en función de estos principios para garantizar un rendimiento óptimo de tu base de datos MongoDB.

Conexión a una base de datos MongoDB

Las cadenas de conexión(Connection Strings) desempeñan un papel fundamental en la creación de conexiones efectivas entre la base de datos MongoDB y diversas entidades,

incluyendo aplicaciones y herramientas como MongoDB Compass y MongoDB Shell (mongosh). En este proceso, se utilizan controladores o drivers para establecer la comunicación. MongoDB ofrece pautas específicas para la construcción de estas cadenas de conexión. A continuación, se presenta una descripción detallada de cómo obtener y utilizar las cadenas de conexión en MongoDB.

Obtención de la Cadena de Conexión(Connection String)

Para obtener una cadena de conexión en MongoDB, se debe seguir un proceso específico. Lo primero es acceder a MongoDB Atlas, que es un servicio de base de datos en la nube de MongoDB. Luego, se debe seleccionar el método preferido de conexión, que puede variar según las necesidades del proyecto. A continuación, se describen las instrucciones generales para obtener la cadena de conexión:

1. Acceder a MongoDB Atlas: Inicie sesión en su cuenta de MongoDB Atlas o regístrese si es necesario.
2. Selección del Método de Conexión: Una vez dentro de MongoDB Atlas, elija el método de conexión que se ajuste a sus requerimientos. Puede optar por conexiones estándar o conexiones con replica sets, entre otras opciones.
3. Seguir las Instrucciones: MongoDB Atlas proporciona instrucciones detalladas sobre cómo configurar la cadena de conexión para el método seleccionado. Estas instrucciones varían según la opción elegida y suelen incluir detalles sobre la configuración, la autenticación y otros parámetros importantes.

Formato de la Cadena de Conexión URI (Connection String URI)

Las cadenas de conexión en MongoDB siguen un formato URI específico que contiene información esencial para la conexión, como la dirección del servidor, el puerto, el nombre de la base de datos y las credenciales de autenticación. Este formato es crucial para que las aplicaciones y herramientas puedan conectarse de manera efectiva a la base de datos.

Operaciones CRUD en MongoDB

Insertión de documentos en una colección MongoDB (Create)

MongoDB ofrece dos métodos para insertar documentos en una colección: `insertOne()` e `insertMany()`. Cada documento dentro de una colección debe tener un campo `_id`, y en caso de que no se proporcione, MongoDB lo generará automáticamente.

Método	Ejemplo
<code>`insertOne()`</code> Para insertar un solo documento en una colección, utilizamos el método <code>`insertOne()`</code> . En el interior de los paréntesis de <code>`insertOne()`</code> , se debe incluir un objeto que contenga los datos del documento.	<pre>db.grades.insertOne({ student_id: 654321, products: [{ type: "exam", score: 90, }, { type: "homework", score: 59, },], class_id: 550, })</pre>
<code>`insertMany()`</code> Para insertar múltiples documentos de una sola vez, utilizamos <code>`insertMany()`</code> . Dentro de <code>`insertMany()`</code> , los documentos se incluyen en un array, y cada documento debe separarse por una coma.	<pre>db.grades.insertMany([{ student_id: 546789, products: [{ type: "quiz", score: 50, }, { type: "homework", score: 70, },], class_id: 551, }, { student_id: 777777, products: [{ type: "quiz", score: 59, }, { type: "quiz", score: 72, },], class_id: 550, },])</pre>

Búsqueda de documentos en una colección MongoDB (Read)

Exploramos cómo insertar y buscar documentos en una colección en MongoDB. Además, construimos consultas utilizando una variedad de operadores y técnicas. A continuación, se detallan las operaciones clave.

La base a toda búsqueda de documentos es usando `db.<collection>.find({})`. He aquí un ejemplo:

```
db.zips.find({ _id: ObjectId("5c8eccclcaa187d17ca6ed16") })
```

1. Operadores de Comparación:

- `$gt` (Mayor que)

```
> db.sales.find({ "items.price": { $gt: 50} })
```

- `$lt` (Menor que)

```
> db.sales.find({ "items.price": { $lt: 50} })
```

- `$lte` (Menor o igual que)

```
> db.sales.find({ "customer.age": { $lte: 65} })
```

- `$gte` (Mayor o igual que)

```
> db.sales.find({ "customer.age": { $gte: 65} })
```

Estos operadores nos permiten encontrar documentos que cumplen con ciertas condiciones numéricas en los campos de la colección.

2. Operadores Lógicos:

- `$and` (Y)

- `$or` (O)

Los operadores lógicos nos ayudan a combinar múltiples expresiones en una sola consulta, permitiendo búsquedas más complejas. Por ejemplo, podemos buscar documentos que cumplan con varias condiciones simultáneamente utilizando `$and`, o seleccionar aquellos que cumplan con al menos una de las condiciones usando `$or`.

3. Operador `$in`:

```
> db.zips.find({ city: { $in: ["PHOENIX", "CHICAGO"] } })
```

El operador `$in` es especialmente útil para seleccionar documentos donde el valor de un campo sea igual a cualquiera de los valores especificados en un array. Por

ejemplo, podemos buscar ciudades que sean "PHOENIX" o "CHICAGO" en el campo "city".

4. Operador `$elemMatch`:

El operador `$elemMatch` nos permite encontrar documentos que contienen un subdocumento específico en un array. Por ejemplo, podemos buscar documentos que tengan un producto llamado "laptop" con un precio superior a 800 y una cantidad de al menos 1 en el campo "items".

Actualización de documentos (Update)

La actualización de documentos en MongoDB es un proceso fundamental para mantener la integridad de los datos y adaptarlos a las necesidades cambiantes de una aplicación. MongoDB ofrece varias opciones para llevar a cabo esta tarea, entre las que se incluyen los métodos `updateOne()`, `findAndModify()`, y `updateMany()`.

1. Actualización de Documentos con `updateOne()`

El método `updateOne()` permite actualizar un solo documento que cumple con un filtro específico. Puedes utilizar operadores de actualización para modificar los campos de los documentos. Algunos de los operadores más comunes son:

- `$set`: Reemplaza el valor de un campo con un valor especificado.
- `$push`: Agrega un valor a un campo de tipo arreglo, creándolo si no existe.
- `upsert`: Crea un nuevo documento con la información proporcionada si no se encuentran documentos que cumplan con el filtro.

A continuación, se presentan ejemplos prácticos de su uso:

```
db.podcasts.updateOne(  
  { title: "The Developer Hub" },  
  { $set: { topics: ["databases", "MongoDB"] } },  
  { upsert: true }  
)
```

2. Actualización de Documentos con `findAndModify()`

El método `findAndModify()` es útil para encontrar y reemplazar un único documento en MongoDB. A diferencia de `updateOne()`, `findAndModify()` devuelve el documento recién actualizado y combina las operaciones de actualización y búsqueda en una sola. Ejemplo:

```
db.podcasts.findAndModify({
  query: { _id: ObjectId("6261a92dfee1ff300dc80bf1") },
  update: { $inc: { subscribers: 1 } },
  new: true,
})
```

3. Actualización de Documentos con `updateMany()`

El método `updateMany()` es ideal para actualizar múltiples documentos que cumplen con un filtro determinado. Esto es especialmente útil cuando se necesita realizar cambios en lotes. Ejemplo:

```
db.books.updateMany(
  { publishedDate: { $lt: new Date("2019-01-01") } },
  { $set: { status: "LEGACY" } }
)
```

Es importante recordar que las actualizaciones en MongoDB son visibles tan pronto como se realizan y no se revierten automáticamente en caso de errores. Además, no se recomienda utilizar `updateMany()` en situaciones que requieran la misma consistencia que las transacciones financieras.

4. Reemplazo de Documentos

Para reemplazar documentos en MongoDB, se utiliza el método `replaceOne()`. Este método toma los siguientes parámetros:

- **Filter**: Una consulta que coincide con el documento a reemplazar.
- **Replacement**: El nuevo documento que reemplazará al antiguo.
- **Options**: Un objeto que especifica opciones para la actualización.

A continuación, se muestra un ejemplo de cómo usar `replaceOne()`:

```
db.books.replaceOne(
  {
    _id: ObjectId("6282afeb441a74a98dbbec4e"),
  },
  {
    title: "Data Science Fundamentals for Python and MongoDB",
    isbn: "1484235967",
```

```

    publishedDate: new Date("2018-5-10"),
    thumbnailUrl:
    "https://m.media-amazon.com/images/I/71opmUBc2wL._AC_UY218_.jpg",
    authors: ["David Paper"],
    categories: ["Data Science"],
  }
)

```

Eliminación de documentos (Delete)

La eliminación de documentos en MongoDB es una operación importante que se puede realizar utilizando los métodos `deleteOne()` y `deleteMany()`. Estos métodos permiten eliminar uno o varios documentos que cumplan con ciertos criterios definidos mediante un filtro. A continuación, se describen estos métodos y se proporciona un ejemplo de su uso.

Métodos `deleteOne()` y `deleteMany()`

Ambos métodos, `deleteOne()` y `deleteMany()`, son esenciales para la manipulación de datos en una base de datos MongoDB. Cada uno acepta un filtro de consulta y un objeto de opciones que permiten controlar el proceso de eliminación.

1. Eliminar un Documento - Método `deleteOne()`

El método `deleteOne()` se utiliza para eliminar un único documento que cumple con los criterios especificados en el filtro. A continuación, se muestra un ejemplo de cómo eliminar un documento utilizando `deleteOne()`:

```

db.podcasts.deleteOne({ _id:
ObjectId("6282c9862acb966e76bbf20a") })

```

En este ejemplo, se elimina un documento de la colección `podcasts` en función del valor del campo `_id`, que corresponde al identificador único del documento. Asegúrate de proporcionar el filtro adecuado para identificar el documento que deseas eliminar.

2. Eliminar Múltiples Documentos - Método `deleteMany()`

El método `deleteMany()` es útil cuando se necesita eliminar varios documentos que cumplan con un conjunto específico de condiciones definidas en el filtro. A

continuación, se muestra un ejemplo de cómo eliminar múltiples documentos utilizando `deleteMany()`:

```
db.podcasts.deleteMany({ category: "crime" })
```

En este caso, todos los documentos de la colección podcasts que tengan el campo category con el valor "crime" serán eliminados. Ten en cuenta que la utilización de `deleteMany()` puede resultar en la eliminación de múltiples documentos a la vez, por lo que es importante definir con precisión el filtro de acuerdo a tus necesidades.

Aggregation en MongoDB

En esta sección, se explorarán los conceptos básicos de la Aggregation en MongoDB, que es una técnica utilizada para analizar y resumir datos de manera efectiva. La Aggregation se lleva a cabo mediante una serie de etapas, que se realizan en orden en un proceso conocido como Aggregation Pipeline.

Definiciones Clave y Estructura

- Aggregation : La Aggregation se refiere a la recolección y resumen de datos de una colección en MongoDB.
- Stage: En el contexto de la agregación, una etapa es una operación que se realiza en los datos sin modificar permanentemente la colección.
- Aggregation Pipeline: Es una serie de etapas que se aplican en orden a los datos de una colección.

Una Aggregation Pipeline se compone de una serie de etapas, cada una con su propia lógica de procesamiento. A continuación, se muestra un ejemplo de la estructura de una tubería de agregación:

```
db.collection.aggregate([
  {
    $stage1: {
      { expression1 },
      { expression2 }...
    },
    $stage2: {
      { expression1 }...
    }
  })
```



```
}})
```

Stages mas usadas comúnmente y ejemplos

Stage	Definition	Sintaxis
<code>\$match</code>	Se utiliza para filtrar documentos que cumplan con ciertos criterios o condiciones específicas	<pre>{ \$match: { "field_name": "value" } }</pre>
<code>\$group</code>	Se utiliza para crear un único documento para cada valor distinto, según una clave de grupo especificada	<pre>{ \$group: { _id: <expression>, // Clave de grupo <field>: { <acumulador> : <expression> } } }</pre>
<code>\$sort</code>	Ordena los documentos de entrada y los devuelve en orden ascendente o descendente. Se utiliza "1" para representar el orden ascendente y "-1" para el orden descendente.	<pre>{ \$sort: { "field_name": 1 } }</pre>
<code>\$limit</code>	Devuelve solo un número especificado de registros.	<pre>{ \$limit: 5 }</pre>
<code>\$project</code>	Permite especificar qué campos deben incluirse o excluirse en los documentos de salida. Se utiliza "1" para incluir un campo y "0" para excluirlo. También se pueden asignar nuevos valores a campos.	<pre>{ \$project: { state: 1, zip: 1, population: "\$pop", _id: 0 } }</pre>

		}
\$set	Se utiliza para crear nuevos campos o modificar el valor de campos existentes en los documentos y, luego, mostrar los documentos con los nuevos campos.	<pre>{ \$set: { place: { \$concat: ["\$city", ",", "\$state"] }, pop: 10000 } }</pre>
\$count	Crea un nuevo documento con el número de documentos en esa etapa de la tubería de agregación y lo asigna a un campo específico.	<pre>{ \$count: "total_zips" }</pre>
\$out	Se utiliza para escribir los documentos resultantes de una tubería de agregación en una colección, almacenando la salida en una nueva colección. Es importante destacar que esta etapa debe ser la última en la tubería de agregación. Si la colección de destino ya existe, \$out la reemplazará con los nuevos datos.	<pre>\$out: { db: "<base_de_datos>", coll: "<nueva_colección>" }</pre>

1. Ejemplo de Uso de \$match y \$group en una Aggregation Pipeline:

El siguiente ejemplo muestra cómo se pueden utilizar las etapas \$match y \$group en una tubería de agregación para encontrar documentos con un campo llamado "state" que coincide con el valor "CA" y luego agrupar esos documentos según la clave de grupo "\$city", mostrando el número total de códigos postales en el estado de California.

```
db.zips.aggregate([
{
  $match: {
    state: "CA"
  }
},
{
  $group: {
    $sum: "$pop"
  }
}
```

```
{
  $group: {
    _id: "$city",
    totalZips: { $count : { } }
  }
}
])
```

2. Ejemplo de Uso de `$sort` y `$limit` en una Aggregation Pipeline

El siguiente ejemplo ordena los documentos en orden descendente según el valor de "pop" y limita la salida a solo los primeros cinco documentos después de la ordenación.

```
db.zips.aggregate([
{
  $sort: {
    pop: -1
  }
},
{
  $limit: 5
}
])
```

3. Ejemplo de uso de `$out` en una Aggregation Pipeline

```
{
  $out: { db: "myDB", coll: "newCollection" }
}
```

La etapa `$out` se utiliza para almacenar los resultados de la agregación en una nueva colección llamada "newCollection" en la base de datos "myDB".

Índices en MongoDB en Colecciones

Los índices en MongoDB son estructuras de datos especiales que almacenan una pequeña porción de los datos de una colección. Estos índices están ordenados y permiten realizar búsquedas eficientes, mejorando el rendimiento de las consultas. Además, apuntan a la identidad del documento en la colección.

1. Ventajas de los índices

Los índices en MongoDB ofrecen varias ventajas, incluyendo:

- Aceleración de consultas, lo que reduce el tiempo de respuesta.
- Reducción de la E/S en disco, ya que se necesita leer menos información.
- Uso eficiente de recursos, lo que disminuye la carga del sistema.
- Soporte para búsquedas de igualdad y operaciones basadas en rangos, devolviendo resultados ordenados.

2. Desventajas de la falta de índices

- Cuando no se utilizan índices, MongoDB debe realizar una exploración completa de la colección para satisfacer las consultas, lo que puede ser costoso en términos de tiempo y recursos.

Tipos de índices más comunes

Los índices en MongoDB pueden ser de varios tipos, pero los más comunes son:

1. Índices de campo único:

- Se crean con `createIndex()`.
- Permiten búsquedas y ordenamiento en función del campo especificado.
- Ejemplo:

```
db.customers.createIndex({  
  birthdate: 1  
})
```

- También se pueden crear índices únicos para evitar valores duplicados:

```
db.customers.createIndex({  
  email: 1  
},  
{
```

```
unique: true
})
```

2. Índices Multikey:

- Se crean de forma similar a los índices de campo único.
- Operan en campos de tipo array, permitiendo buscar valores en arrays.
- Ejemplo:

```
db.customers.createIndex({
  accounts: 1
})
```

3. Índices Compuestos:

- Se crean con múltiples campos para búsquedas y ordenamientos más complejos.
- El orden de los campos en el índice es importante: igualdad, ordenamiento y rango.
- Ejemplo:

```
db.customers.createIndex({
  active: 1,
  birthdate: -1,
  name: 1
})
```

- El orden de los campos en un índice compuesto es importante y se recomienda seguir el orden de: **Igualdad, Orden y Rango.**

Otros métodos para realizar sobre índices

1. Visualización de los índices en una colección

Puedes ver todos los índices creados en una colección utilizando el método

```
getIndexes():
```

```
db.customers.getIndexes()
```

1. Comprobar si se está utilizando un índice en una consulta

Puedes usar el método `explain()` para verificar si una consulta está utilizando un índice y cuál está seleccionando. Esto proporciona información detallada sobre las etapas de ejecución, como "IXSCAN" para indicar el uso de un índice o "COLLSCAN" para indicar una exploración completa de la colección. Por ejemplo:

```
db.customers.explain().find({
  birthdate: {
    $gt: ISODate("1995-08-01")
  }
})
```

1. Cobertura de una consulta mediante el índice

Un índice cubre una consulta cuando MongoDB no necesita buscar los datos en la memoria, ya que toda la información requerida ya está presente en el índice. Esto se logra mediante proyecciones que incluyen solo los campos necesarios que están presentes en el índice. Por ejemplo:

```
db.customers.explain().find({
  birthdate: {
    $gt: ISODate("1995-08-01")
  },
  active: true
}, {
  name: 1,
  birthdate: 1,
  _id: 0
}).sort({
  birthdate: -1,
  name: 1
})
```

Eliminación de índices

Puedes eliminar índices de una colección utilizando el método `dropIndex()`. También puedes eliminar todos los índices, excepto el índice predeterminado en el campo `_id`, con `dropIndexes()`. Por ejemplo:

1. Eliminar un índice por nombre

```
db.customers.dropIndex('active_1_birthdate_-1_name_1')
```

2. Eliminar un índice por clave

```
db.customers.dropIndex({  
  active: 1,  
  birthdate: -1,  
  name: 1  
})
```

3. Eliminar todos los índices excepto el índice predeterminado en `_id`

```
db.customers.dropIndexes()
```

Es importante recordar que el índice predeterminado en `_id` no se puede eliminar, ya que MongoDB lo utiliza internamente para identificar los documentos de la colección.

Material de Presentación

La presentación estará acompañada de los siguientes materiales:

- Presentación:
 - Genially: A Través de este se presentarán las diapositivas necesarias para explicar a nuestros compañeros el tema que nos tocó.

A continuación adjuntamos el link de la presentación previamente dicha la cual ya esta creada:
<https://view.genial.ly/654950d16dfa440011df2a1b/presentation-presentacion-uni-educacion>
- Actividades:
 - Archivo de python.
 - Kahoot: Mediante esta página realizaremos una encuesta sobre la presentación previamente dada. De esta forma comprobaremos la atención que se dio a los conceptos que explicamos anteriormente.

Bibliografía

Este informe se realizó con material bibliográfico de la documentación oficial de MongoDB, y del curso de Python y MongoDB de MongoDB university.

Links y referencias:

- [MongoDB Docs: Get Connection String](#)
- [MongoDB Docs: Connection String URI Format](#)
- [MongoDB Docs: The MongoDB Shell](#)
- [MongoDB Docs: MongoDB Compass](#)
- [MongoDB Docs: Connect via Your Application](#)
- [MongoDB Docs: Troubleshoot Connection Issues](#)
- [MongoDB Docs: insertOne\(\)](#)
- [MongoDB Docs: insertMany\(\)](#)
- [MongoDB Docs: find\(\)](#)
- [MongoDB Docs: \\$in](#)
- [MongoDB Docs: Comparison Operators](#)
- [MongoDB Docs: \\$elemMatch](#)
- [MongoDB Docs: Querying Arrays](#)
- [MongoDB Docs: Logical Operators](#)
- [MongoDB Docs: replaceOne\(\)](#)
- [MongoDB Docs: Update Operators](#)
- [MongoDB Docs: \\$set](#)
- [MongoDB Docs: \\$push](#)
- [MongoDB Docs: upsert](#)
- [MongoDB Docs: findAndModify\(\)](#)
- [MongoDB Docs: updateMany\(\)](#)
- [MongoDB Docs: deleteOne\(\)](#)
- [MongoDB Docs: deleteMany\(\)](#)
- [MongoDB Docs: Aggregation Operations](#)
- [MongoDB Docs: Aggregation Pipelines](#)
- [MongoDB Docs: \\$match](#)
- [MongoDB Docs: \\$group](#)
- [MongoDB Docs: \\$sort](#)
- [MongoDB Docs: \\$limit](#)
- [MongoDB Docs: \\$project](#)
- [MongoDB Docs: \\$count](#)
- [MongoDB Docs: \\$set](#)
- [MongoDB Docs: \\$out](#)
- [What Is MongoDB? | MongoDB](#)
- [What is MongoDB? — MongoDB Manual](#)
- [MongoDB Python Developer Learning Path](#)
-

Acreditaciones ganadas durante la capacitación:

1. <https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/f6d4b15e-2a0e-4632-9af8-d191e936eb99-facundo-oliva-marchetto-4c7a5c46-989f-414a-9ee3-641247900a9d-certificate.pdf>
2. <https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/f6d4b15e-2a0e-4632-9af8-d191e936eb99-facundo-oliva-marchetto-233581d8-06a4-4eef-a070-5461b902452b-certificate.pdf>
3. <https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/f6d4b15e-2a0e-4632-9af8-d191e936eb99-facundo-oliva-marchetto-139aaada-91d0-4c19-96bf-6eb2c9a90068-certificate.pdf>
4. <https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/2f59ca0e-a46b-4ec3-95af-e38007a696ca-llmtech-n-a-3c93076e-9f33-4df5-a6c7-cbd14401ef0d-certificate.pdf>
5. <https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/2f59ca0e-a46b-4ec3-95af-e38007a696ca-llmtech-n-a-427f2270-22f2-4248-93ed-f1bc57fa2c9e-certificate.pdf>
6. Y otras más de 15 certificaciones similares respecto a Mongo University, perteneciente a los integrantes de este grupo.