Capítulo 3
Indexar en MongoDB

1. ¿Cómo funciona?

En terminología de bases de datos, relacionales o no, un índice es muy parecido al que se encuentra al final de cualquier libro: agrupa los términos importantes que aparecen en el libro con, enfrente, los números de página en los que se encuentran. Esto simplemente nos ahorra tener que releer todo el libro cuando buscamos un solo término.

Por analogía, un índice colocado en el campo o subcampo de una colección significa que no tenemos que recorrer toda nuestra colección para encontrar los valores de este campo (o subcampo) correspondientes a nuestra consulta y, por lo tanto, ayuda a mantener el tiempo de ejecución de nuestras consultas lo más corto posible.

Los índices presentan ventajas e inconvenientes: mejoran considerablemente los tiempos de ejecución de las solicitudes de lectura, pero ralentizan las operaciones de escritura, como las inserciones, supresiones o actualizaciones, que requieren su reconstrucción. Sin embargo, las ralentizaciones observadas son generalmente insignificantes en comparación con la reducción del tiempo de ejecución que generan.

Para saber qué campos de una colección deben indexarse, hay que tener una idea de las consultas que se hacen sobre ella y de su frecuencia. Los campos a los que se dirigen con frecuencia las consultas deben indexarse prioritariamente. Un sitio web que ofrece un motor de búsqueda de productos en forma de zona de entrada de texto debe tener una colección de productos en la que se indexe el nombre de los productos o la categoría a la que pertenecen, porque los usuarios suelen introducir nombres de productos («iPhone 10») o de categorías («cafeteras italianas»).

La naturaleza de nuestra aplicación tendrá un impacto directo en nuestra lógica de indexación: ¿está orientado a la escritura (*write-heavy*)? ¿O a la lectura (*read-heavy*)? ¿No se encuentra un sitio web comercial en una encrucijada? Leemos mucho (búsquedas de productos, campañas promocionales por correo electrónico de la base de clientes, API a la disposición de los socios), pero también escribimos mucho (cestas de compra, pedidos, reposición de nuestro catálogo de productos, actualización diaria de los precios en función de los criterios de los proveedores), por lo que es comprensible que no podamos decidir una estrategia de indexación en el reverso de una servilleta.

2. Índices simples

Cuando se crea una colección, MongoDB genera automáticamente un índice sobre el campo `_id`. Este índice no puede borrarse, ya que garantiza la propia unicidad de este identificador. Para crear un índice, se debe utilizar la función `createIndex`, cuya sintaxis es la siguiente:

```
db.collection.createIndex(< campo_y_tipo >, < opciones >)
```

Le presentamos la nueva versión de nuestra colección `personas`:

```
db.personas.drop()

db.personas.insertMany(
[
{"apellido": "Durand", "nombre": "René", "intereses": ["jardinería",
"bricolaje"], "edad": 77},
{"apellido": "Durand", "nombre": "Gisèle", "intereses": ["bridge",
"cocina"], "edad": 75},
{"apellido": "Dupont", "nombre": "Gaston", "intereses": ["jardinería",
"petanca"], "edad": 79},
{"apellido": "Dupont", "nombre": "Catherine", "intereses": ["cocina"], "edad": 66},
{"apellido": "Duport", "nombre": "Eric", "intereses": ["cocina",
"petanca"], "edad": 57},
```

```
{"apellido": "Duport", "nombre": "Arlette", "intereses": ["jardinería"], "edad":
80},
{"apellido": "Lejeune", "nombre": "Jean", "intereses": ["jardinería"], "edad": 75},
{"apellido": "Lejeune", "nombre": "Mariette", "intereses": ["jardinería", "bridge"],
"edad": 66}
]
)
```

Sin más preámbulos, vamos a crear un primer índice sobre el campo `edad` de los documentos de la colección. Cuando se crea un índice, se debe especificar el orden en que los valores de este campo aparecen en el índice: este orden es ascendente o descendente (1 y -1 respectivamente, exactamente igual que `sort()`). Hemos decidido que nuestro índice sobre el campo `edad` contendrá los valores tomados por `edad`, ordenados de forma descendente:

```
db.personas.createIndex({"edad": -1})
```

Para comprobar que este índice se ha creado como se esperaba, también podemos utilizar la función `getIndexes` y aplicarla a nuestra colección:

```
db.personas.getIndexes()
```

Esto muestra una tabla que contiene dos documentos con los detalles de nuestros índices:

```
[
  { v: 2, key : { _id: 1 }, name: '_id_' },
  { v: 2, key: { edad: -1 }, name: 'edad_-1' }
```

Veremos el campo en el que se basa el índice, el orden de indexación (ascendente o descendente) y el nombre que MongoDB ha dado a nuestro índice. De forma predefinida, nuestro índice tiene el nombre del campo objetivo con un carácter subrayado seguido del orden, que en este caso es `edad_-1`. Esto no es muy estético o significativo, así que vamos a nombrarlo de otra manera.

Para llevar a cabo la operación de cambio de nombre del índice, empezaremos por borrar el ya existente antes de volver a crear uno con un nuevo nombre: `idx_edad`. Para eliminar un índice, se utiliza el comando `dropIndex`, al que pasaremos el nombre del antiguo índice.

El nombre de nuestro nuevo índice se especificará en el documento que contiene los parámetros del índice, frente a la clave `name`. Por último, listaremos los índices de nuestra colección para comprobar que el nuevo índice con el nuevo nombre se ha creado correctamente.

```
db.personas.dropIndex("edad_-1")
db.personas.createIndex({"edad": -1}, {"name": "idx_edad"});
db.personas.getIndex()
[
  { v: 2, key: { _id: 1 },   name: '_id_' },
  { v: 2, key: { edad: -1 }, name: 'idx_edad' }
]
```

Nuestra colección ejemplo tiene un tamaño más que modesto y somos los únicos que la utilizamos, por lo que podemos permitirnos reconstruir el índice borrándolo. Es fácil imaginar que el impacto sería muy diferente en una colección de varios cientos de miles (¡o incluso millones!) de documentos utilizados por todo el personal de una gran empresa. A continuación, estos distintos procedimientos deben llevarse a cabo en una franja horaria propicia para el mantenimiento (en mitad de la noche, por ejemplo), utilizando la opción `Background` para indicar que se trata de una tarea en segundo plano de baja prioridad.

Por ejemplo, para crear un índice alfabético del campo `apellido` en segundo plano, se escribiría:

```
db.personas.createIndex( { "apellido": 1 }, { "background": true})
```

Al crear un índice, al igual que al crear una colección o una vista (que veremos más adelante), es posible especificar una intercalación (collation), pero esta funcionalidad solo está disponible desde la versión 3.4 de MongoDB. Si tomamos el índice anterior y le añadimos las opciones relativas a la naturaleza de la intercalación, obtenemos:

```
db.personas.createIndex(
  { "apellido": 1 },
  { "background": true   "intercalacion": {
       "local": "es"
  }

)
```

Cuando se crea un índice simple con una intercalación, solo las consultas que especifiquen esta misma intercalación podrán utilizarlo, de lo contrario se realizará una operación de búsqueda de colecciones (*collection scan*), ya que se utiliza una comparación binaria de forma predefinida. Tomemos el índice que acabamos de crear. La siguiente consulta lo utilizará:

```
db.personas.find({"apellido": "René"}).intercalacion({local: "es"})
```

Mientras que en el siguiente caso no se podrá contar con el índice:

```
db.personas.find({"apellido": "René"})
```

3. Índices compuestos

Un índice puede referirse a más de un campo: es lo que se conoce como índice-compuesto (*compound index*). En este tipo de índice, el orden en que se enumeran los campos es importante. Eliminemos nuestro índice `idx_edad` y creemos un índice compuesto llamado `idx_edad_nombre` que buscará la edad y luego el nombre de las personas:

```
db.personas.createIndex({"edad": 1, "nombre": 1},
{"name": "idx_edad_nombre"})
```

El índice se ordenará primero por valores de edad crecientes y después por orden alfabético del nombre dentro de cada uno de los diferentes valores de edad.

Cuando se utiliza con intercalación un índice compuesto cuyo prefijo no es una cadena de caracteres, una matriz o un subdocumento, una consulta que utilice una intercalación incorrecta para el campo de texto indexado puede seguir basándose en el prefijo del índice.

Supongamos que nuestro índice anterior se creó de esta forma:

```
db.personas.createIndex(
  { "edad": 1, "nombre": 1},
  { "nombre": "idx_edad_nombre", "intercalacion": { local: "es" }}
)
```

La siguiente consulta, que utiliza la intercalación binaria básica para comparar cadenas de caracteres (y no el idioma `es`), podrá sin embargo utilizar `idx_nombre_edad` porque edad es el prefijo:

```
db.personas.find({"edad": {$gt: 40}, "nombre": "Christophe"})
```

Prefijo del índice

Una consulta puede utilizar todos los campos que componen el índice compuesto, o solo una subsección, siempre que esté formada por campos que aparezcan al principio del índice. Esta subsección del índice se denomina *prefijo* y algunos sistemas de gestión de bases de datos relacionales utilizan el término prefijo izquierdo (*leftmost prefix*).

En este caso, nuestro índice cubre dos campos y será utilizado por consultas dirigidas a:

– solo el campo `apellido` (el campo situado más a la izquierda, que inicia el índice y constituye su *prefijo*);

– los campos de `nombre` y `edad` (es decir, todo el índice).

Una consulta dirigida únicamente a los valores del campo `edad` no se beneficiará de este índice compuesto, ya que este campo no forma parte del *prefijo*.

Escribamos un índice compuesto llamado `idx_apellido_nombre_edad` que se aplicará a la tripleta `apellido`, `nombre` y `edad` (en este orden):

```
db.personas.createIndex(
  {"apellido": 1, "nombre": 1, "edad": 1},
  {"name": "idx_apellido_nombre_edad"}
)
```

Podremos apoyarnos en el para:

– consultar el primer campo del *prefijo*, apellido:

```
db.personas.find({"apellido": "Lejeune"})
```

– consultas relativas a los dos campos que componen el *prefijo*, `apellido` y `nombre`:

```
db.personas.find({"apellido": "Lejeune", "nombre": "Juan"})
db.personas.find({"nombre": "Juan", "apellido": "Lejeune"})
```

– consultas relativas a todos los campos que componen el índice compuesto:

```
db.personas.find({"apellido": "Lejeune", "nombre": "Jean", "edad": 75})
db.personas.find({"apellido": "Lejeune", "edad": 75, "nombre": "Jean"})
db.personas.find({"edad": 75, "apellido": "Lejeune", "nombre": "Juan"})
db.personas.find({"edad": 75, "nombre": "John", "apellido": "Lejeune"})
```

– consultas que contengan el *prefijo* y otro campo:

```
db.personas.find({"edad": 75, "apellido": "Durand"})
```

Sin embargo, las siguientes consultas no podrán utilizar el índice triple:

– las dirigidas a un campo que no forma parte del *prefijo*:

```
db.personas.find({"edad": 75})
```

– aquellas que solo utilizan una parte del *prefijo* (recuerde que está formado por los campos `apellido` y `nombre`):

```
db.personas.find({"nombre": "Jean"})
db.personas.find({"nombre": "Jean", "edad": 75})
```

Los índices también pueden utilizarse para ordenar un cursor utilizando el método de `sort` visto anteriormente. Mientras que, en el caso de los índices simples, el orden ascendente o descendente no tiene efecto en la ordenación porque MongoDB es capaz de recorrer el índice en cualquier dirección, este no es el caso de los índices compuestos: los campos usados en el `sort` deben aparecer en el mismo orden que en el índice.

Crearemos un índice compuesto `idx_ apellido_edad` definido como sigue:

```
{"edad": 1, " apellido": 1}
```

Esto significa que las siguientes clasificaciones lo utilizarán:

```
db.personas.find().sort({"edad": 1})
db.personas.find().sort({"edad": 1, "apellido": 1 })
```

Sin embargo, las siguientes clasificaciones no podrán contar con él:

```
db.personas.find().sort({"edad": 1})
db.personas.find().sort({"edad": 1, "apellido": 1 })
```

```
db.personas.find().sort({"apellido": 1, "edad": 1})
db.personas.find().sort({"apellido": 1})
```

Para utilizar un índice al ordenar, también hay que tener en cuenta el orden especificado cuando se creó el índice, ya que es este orden y su inverso exacto los que utilizarán el índice.
Tomemos el índice compuesto `idx_apellido_edad`:

```
{"apellido": 1, "edad": 1}
```

Su inversa será, por tanto:

```
{"apellido": -1, "edad": -1}
```

Las clasificaciones que utilizará el índice compuesto serán las siguientes:

```
db.personas.find().sort({"edad": 1, "apellido": 1 })
db.personas.find().sort({"edad": -1, "apellido": -1 })
```

Pero no podrán contar con él las siguientes:

```
db.personas.find().sort({"edad": -1, "apellido": 1 })
db.personas.find().sort({"edad": 1, "apellido": -1 })
```

Como regla general, cuando MongoDB utiliza un índice en una consulta, devuelve los documentos en el orden en que el índice los ha ordenado. Tomemos como ejemplo la siguiente consulta, que se dirige a documentos cuyo nombre empieza por «Du» (es una *expresión regular* contenida entre dos barras inclinadas, tenga en cuenta que sí se distingue entre mayúsculas y minúsculas):

```
db.personas.find({"apellido": /^Du/}, {"_id": 0, "apellido": 1, "edad": 1})
```

Con nuestro índice `idx_apellido_edad` de la forma {"apellido": 1, "edad": 1}, esto es lo que se obtiene:

```
{ "apellido" : "Dupont", "edad" : 66 }
{ "apellido" : "Dupont", "edad" : 79 }
{ "apellido" : "Duport", "edad" : 57 }
{ "apellido" : "Duport", "edad" : 80
{ "apellido" : "Durand", "edad" : 75 }
{ "apellido" : "Durand", "edad" : 77 }
```

Cuando cambiamos este índice por `{"apellido": 1, "edad": -1}`, vemos que efectivamente los resultados mostrados siguen el nuevo orden del índice, es decir, ordenan la edad en orden descendente dentro del mismo nombre:

```
{ "apellido" : "Dupont", "edad" : 79 }
{ "apellido" : "Dupont", "edad" : 66 }
{ "apellido" : "Duport", "edad" : 80 }
{ "apellido" : "Duport", "edad" : 57 }
{ "apellido" : "Durand", "edad" : 77 }
{ "apellido" : "Durand", "edad" : 75 }
```

4. Índices únicos

Los índices únicos garantizan que un valor determinado aparecerá como máximo una vez en el índice. Antes se vió que se colocaba sistemáticamente un índice de este tipo en el campo `_id` de los documentos de una colección, ¡y que era imposible borrarlo! En el estado actual de nuestra colección de `personas`, no podríamos poner este tipo de índice en el campo `apellido` porque la mayoría de los valores de este campo tienen varias apariciones. En cambio, podríamos crear un índice único para el campo `nombre`, donde todos los valores son únicos:

```
db.personas.createIndex({"nombre": 1}, {"unique": true})
```

A partir de ahora, cualquier intento de insertar a una persona con un nombre que ya esté presente en uno de los documentos de nuestra colección resultará en un fracaso estrepitoso. Evidentemente, colocar un índice único en un campo que muy probablemente contenga duplicados es una decisión bastante desacertada.

Además, ahora que nuestro campo `apellido` está sujeto a una restricción de unicidad, no podremos insertar más de un documento que no contenga este campo. La siguiente inserción falla en parte porque, como `apellido` se marcó como `null` al insertar el primer documento, no es posible insertar un segundo documento cuyo campo `apellido` también sea `null`.

```
db.personas.insertMany([{"apellido": "Ferrandez"}, {"nombre": "Pascal"}])
```

Por lo tanto, se insertará el primer documento y se rechazará el segundo.