

MongoDB Fundamentos

Módulo 3



Índices en MongoDB

Índices en MongoDB

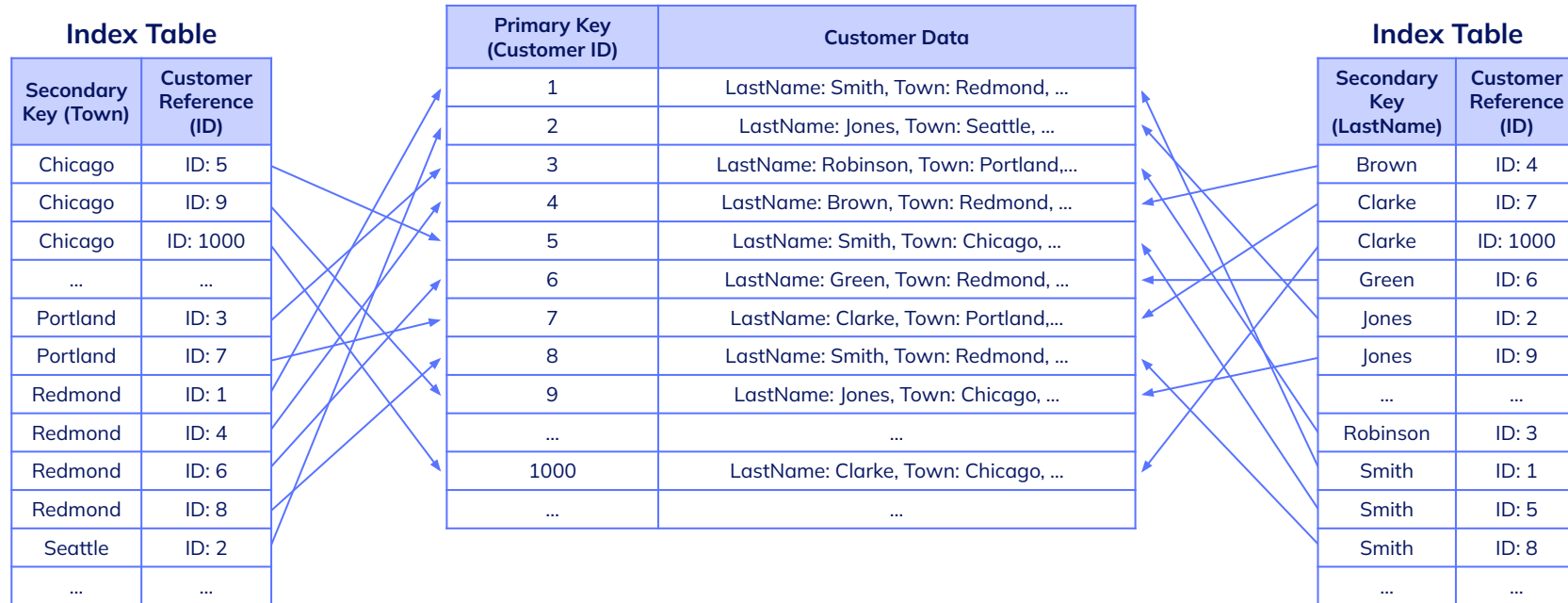
Concepto de Índice

Los índices son estructuras auxiliares que permiten a la base de datos acceder más rápido a ciertos documentos. Se pueden imaginar como una copia "ordenada" de ciertas claves de una colección.

Esta copia incluye, además de las claves ordenadas, un índice (puntero) al documento que la contiene.



Fact Table



Permiten mejorar la eficiencia de muchas operaciones CRUD y evitar los recorridos completos de las colecciones. Es el usuario el que debe decidir qué índices son los más adecuados para su base de datos, veremos algunos criterios.

Con MongoDB, no se puede pensar en tener una colección con millones de documentos, sin tener índices sobre uno o varios campos. La diferencia entre realizar una consulta sobre un campo con índice y una sin él, puede ser abismal.

Saber crear y configurar índices en las colecciones, es vital.

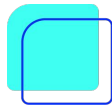
La mayoría de las bases de datos -y también Mongo- usan un tipo de árboles llamados **BTree** para almacenar internamente los índices. El motor WiredTiger usa una variante llamada **b+trees**.

Los índices en MongoDB se generan en forma de **Árbol-B** o **B-Tree**. Es decir, que los datos se guardan en forma de árbol, pero se mantienen los nodos balanceados. Esto incrementa la velocidad a la hora de buscar y también a la hora de devolver resultados ya ordenados. De hecho MongoDB es capaz de recorrer los índices en ambos sentidos, por lo que con un solo índice, se puede conseguir ordenación tanto ascendente como descendente.

Para mejorar la eficiencia de los índices, es recomendable que estos tengan una cardinalidad alta. ¿Y qué es la cardinalidad? Como ejemplo, se puede pensar en un diccionario. Las palabras están ordenadas en orden alfabético, y son únicas. Por eso es relativamente fácil encontrar una palabra. Ahora imaginemos que el diccionario está solo agrupado y ordenado por la primera letra de cada palabra. Habrá miles de palabras que comienzan por A, otras miles por M, muchas otras por la P etc. Buscar en un diccionario así sería algo tedioso. Una vez encontrada la primera letra, habría que leer cada palabra de ese rango para encontrar la palabra buscada.

Eso es exactamente la cardinalidad. Cuantos más valores únicos tenga el campo, más alta será la cardinalidad. Y más eficiente será el índice.

En general añadir índices tiene un efecto sobre la mejora del rendimiento más apreciable que comprar más memoria, cambiar el disco o incluso un ordenador con CPU más rápida. ¡Y es gratis!



Sintaxis

```
db.collection.createIndex({<string field>:<1|-1 order> [,<string field>:<1|-1 order>]});
```

Observaciones

Impacto en el rendimiento: tener en cuenta que los índices mejoran el rendimiento de lectura, pero pueden tener un impacto negativo en el rendimiento de escritura, ya que la inserción de un documento requiere la actualización de todos los índices.



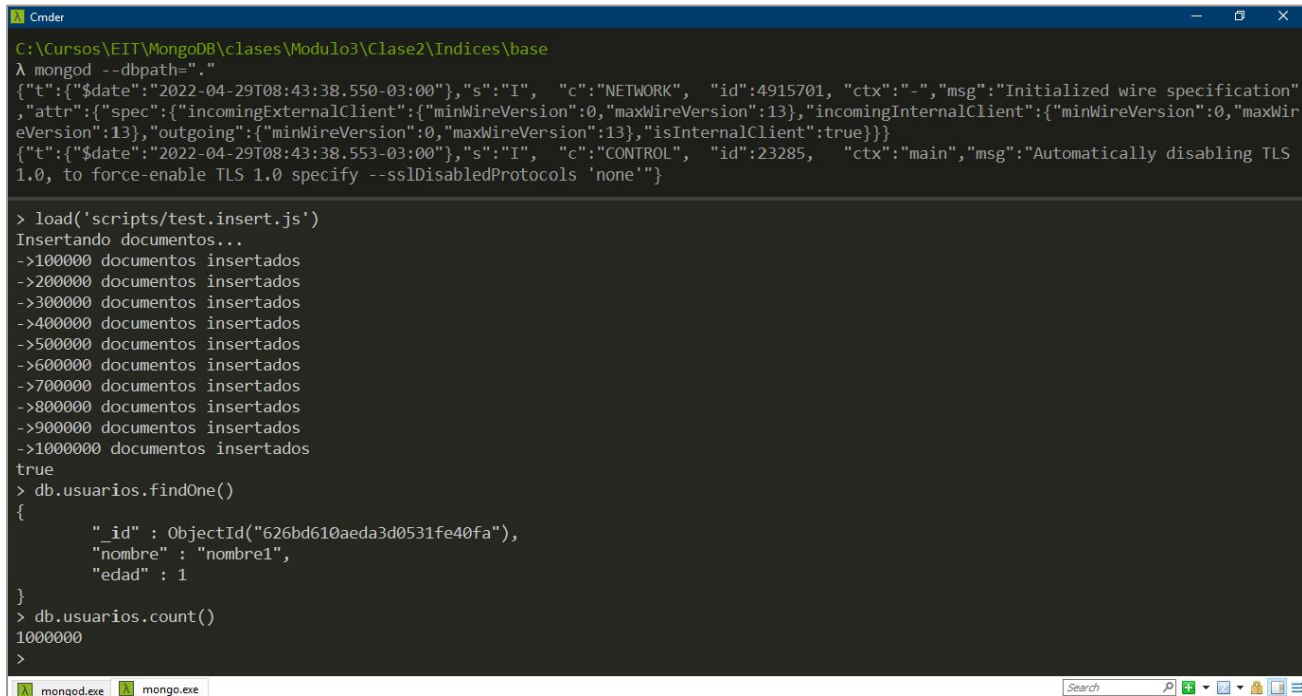
Test de rendimiento

A sólo efecto de verificar el rendimiento de búsqueda al aplicar ó no un índice, en la consola de MongoShell ejecutamos un pequeño script que nos va a crear 1000000 documentos simples:

```
var db = db.getSiblingDB('mibase')      // use mibase
db.usuarios.drop()

print('Insertando documentos...')
for(i=1; i<=1000000; i++) {
    if(i%100000 == 0) print('->' + i + ' documentos
insertados')
    db.usuarios.insertOne(
        {
            nombre: 'nombre' + i,
            edad: i % 100
        }
    )
}
```


Consola:



```
C:\Cursos\EIT\MongoDB\clases\Modulo3\Clase2\Indices\base
λ mongod --dbpath="."
{"t":{"date":"2022-04-29T08:43:38.550-03:00"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"-", "msg":"Initialized wire specification",
"attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":13},
"outgoing":{"minWireVersion":0,"maxWireVersion":13},"isInternalClient":true}}}
{"t":{"date":"2022-04-29T08:43:38.553-03:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main", "msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}

> load('scripts/test.insert.js')
Insertando documentos...
->100000 documentos insertados
->200000 documentos insertados
->300000 documentos insertados
->400000 documentos insertados
->500000 documentos insertados
->600000 documentos insertados
->700000 documentos insertados
->800000 documentos insertados
->900000 documentos insertados
->1000000 documentos insertados
true
> db.usuarios.findOne()
{
  "_id" : ObjectId("626bd610aeda3d0531fe40fa"),
  "nombre" : "nombre1",
  "edad" : 1
}
> db.usuarios.count()
1000000
>
```

Luego, mediante el siguiente script, se buscará un documento en particular, dentro del millón de documentos que tiene la colección usuarios.

El cursor contiene un método **explain** mediante el cual se puede extraer el tiempo de ejecución de la búsqueda.

```
var db = db.getSiblingDB('mibase')  
  
// proceso de búsqueda  
var cursor = db.usuarios.find({ nombre: 'nombre5555' })  
printjson(cursor.next())  
  
var tiempoTranscurrido = cursor.explain('executionStats')  
                                .executionStats  
                                .executionTimeMillis  
  
print('El tiempo de búsqueda fue de ' + tiempoTranscurrido + 'mS')
```

Al ejecutarlo en la consola se obtiene un tiempo de 413ms de consulta. Si se busca ese mismo documento por su `_id`:

```
var db = db.getSiblingDB('mibase')
// proceso de búsqueda
//var cursor = db.usuarios.find({ nombre: 'nombre5555' })

var cursor = db.usuarios.find({ _id : ObjectId("626bd612aeda3d0531fe56ac")
})
printjson(cursor.next())

var tiempoTranscurrido = cursor.explain('executionStats')
                           .executionStats
                           .executionTimeMillis
print('El tiempo de búsqueda fue de ' + tiempoTranscurrido + 'mS')
```

Se obtendrá:

```
> load('scripts/test.read.js')
{
  "_id" : ObjectId("626bd612aeda3d0531fe56ac"),
  "nombre" : "nombre5555",
  "edad" : 55
}
El tiempo de búsqueda fue de 413ms
true
> load('scripts/test.read.js')
{
  "_id" : ObjectId("626bd612aeda3d0531fe56ac"),
  "nombre" : "nombre5555",
  "edad" : 55
}
El tiempo de búsqueda fue de 0ms
true
>
```

En este caso, el tiempo de búsqueda da 0 milisegundos porque hay un índice creado automáticamente dentro de esa colección, en su **objectId**, que se realiza al momento de insertar el primer documento.

Se reformula nuevamente el script de búsqueda. Se vuelve al **query** por campo nombre:

```
var cursor = db.usuarios.find({ nombre: 'nombre5555' })  
//var cursor = db.usuarios.find({ _id :  
ObjectId("626bd612aeda3d0531fe56ac") })
```

Se crea un índice para el campo nombre y se ejecuta:

```
> db.usuarios.createIndex({ nombre : 1 })
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : false,
  "ok" : 1
}
> load('scripts/test.read.js')
{
  "_id" : ObjectId("626bd612aeda3d0531fe56ac"),
  "nombre" : "nombre5555",
  "edad" : 55
}
El tiempo de búsqueda fue de 0mS
true
>
```

Se puede observar que el índice mejoró significativamente el query llevando el tiempo a cero.



Ahora, al borrar el índice y realizar nuevamente la consulta:

```
> db.usuarios.dropIndex({ nombre : 1 })
{ "nIndexesWas" : 2, "ok" : 1 }
> load('scripts/test.read.js')
{
  "_id" : ObjectId("626bd612aeda3d0531fe56ac"),
  "nombre" : "nombre5555",
  "edad" : 55
}
El tiempo de búsqueda fue de 406mS
true
> |
```

Se comprueba que sin el índice empeora el tiempo de búsqueda por campo nombre.



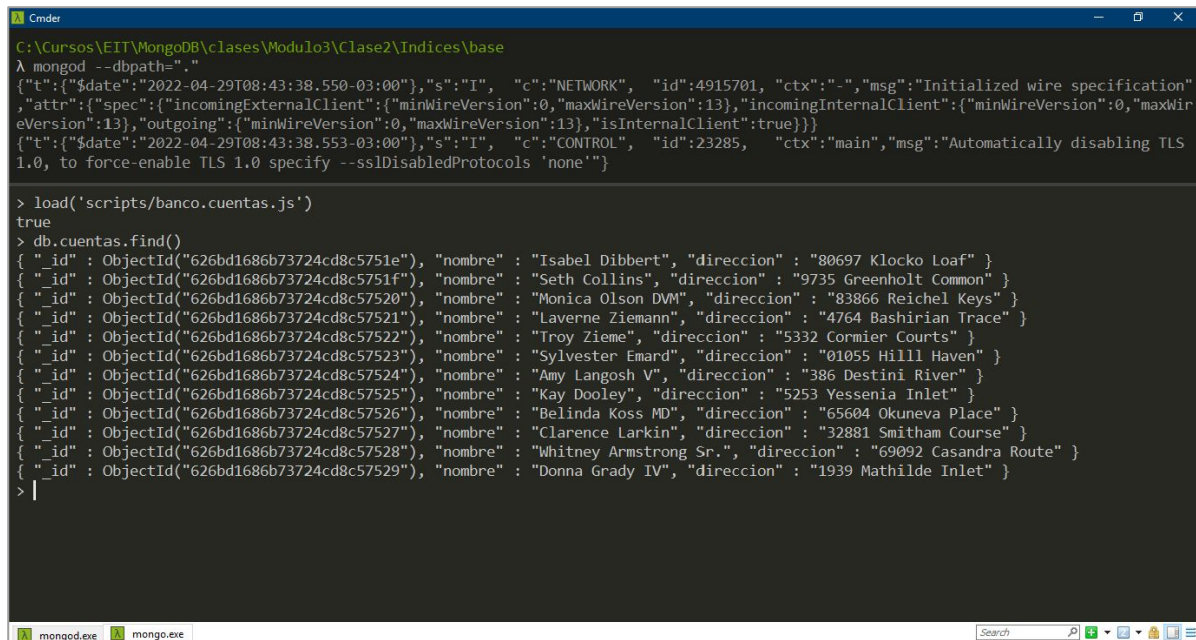
Creación y borrado de índices

Se crea el script que inserta documentos en una colección **cuentas** dentro de la base de datos **banco**. Veamos la siguiente pantalla.




```
db = db.getSiblingDB("banco"); // use banco
db.cuentas.drop();
db.cuentas.insertMany([
  { nombre: "Isabel Dibbert", direccion: "80697 Klocko Loaf" },
  { nombre: "Seth Collins", direccion: "9735 Greenholt Common" },
  { nombre: "Monica Olson DVM", direccion: "83866 Reichel Keys" },
  { nombre: "Laverne Ziemann", direccion: "4764 Bashirian Trace" },
  { nombre: "Troy Zieme", direccion: "5332 Cormier Courts" },
  { nombre: "Sylvester Emard", direccion: "01055 Hilll Haven" },
  { nombre: "Amy Langosh V", direccion: "386 Destini River" },
  { nombre: "Kay Dooley", direccion: "5253 Yessenia Inlet" },
  { nombre: "Belinda Koss MD", direccion: "65604 Okuneva Place" },
  { nombre: "Clarence Larkin", direccion: "32881 Smitham Course" },
  { nombre: "Whitney Armstrong Sr.", direccion: "69092 Casandra Route" },
  { nombre: "Donna Grady IV", direccion: "1939 Mathilde Inlet" }
])
```

Se ejecuta con load en la consola Mongo Shell y se verifican los datos cargados:



```
C:\Cursos\EIT\MongoDB\clases\Modulo3\Clase2\Indices\base
λ mongod --dbpath="."
{"t":{"date":"2022-04-29T08:43:38.550-03:00"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"-", "msg":"Initialized wire specification",
"attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVersion":0,"maxWireVersion":13},"isInternalClient":true}}}
{"t":{"date":"2022-04-29T08:43:38.553-03:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main", "msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"

> load('scripts/banco.cuentas.js')
true
> db.cuentas.find()
{ "_id" : ObjectId("626bd1686b73724cd8c5751e"), "nombre" : "Isabel Dibbert", "direccion" : "80697 Klocko Loaf" }
{ "_id" : ObjectId("626bd1686b73724cd8c5751f"), "nombre" : "Seth Collins", "direccion" : "9735 Greenholt Common" }
{ "_id" : ObjectId("626bd1686b73724cd8c57520"), "nombre" : "Monica Olson DVM", "direccion" : "83866 Reichel Keys" }
{ "_id" : ObjectId("626bd1686b73724cd8c57521"), "nombre" : "Laverne Ziemann", "direccion" : "4764 Bashirian Trace" }
{ "_id" : ObjectId("626bd1686b73724cd8c57522"), "nombre" : "Troy Zieme", "direccion" : "5332 Cormier Courts" }
{ "_id" : ObjectId("626bd1686b73724cd8c57523"), "nombre" : "Sylvester Emard", "direccion" : "01055 Hilll Haven" }
{ "_id" : ObjectId("626bd1686b73724cd8c57524"), "nombre" : "Amy Langosh V", "direccion" : "386 Destini River" }
{ "_id" : ObjectId("626bd1686b73724cd8c57525"), "nombre" : "Kay Dooley", "direccion" : "5253 Yessenia Inlet" }
{ "_id" : ObjectId("626bd1686b73724cd8c57526"), "nombre" : "Belinda Koss MD", "direccion" : "65604 Okuneva Place" }
{ "_id" : ObjectId("626bd1686b73724cd8c57527"), "nombre" : "Clarence Larkin", "direccion" : "32881 Smitham Course" }
{ "_id" : ObjectId("626bd1686b73724cd8c57528"), "nombre" : "Whitney Armstrong Sr.", "direccion" : "69092 Casandra Route" }
{ "_id" : ObjectId("626bd1686b73724cd8c57529"), "nombre" : "Donna Grady IV", "direccion" : "1939 Mathilde Inlet" }
> |
```

Cada colección tiene su propio conjunto de índices. Para crear un índice nuevo se utiliza la instrucción **createIndex**:

```
> db.cuentas.createIndex({nombre:1,direccion:1})
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : false,
  "ok" : 1
}
```

La sintaxis recuerda a la de la función **sort**: el argumento especifica las claves por las que se indexará.

Las claves pueden ser de cualquier tipo, incluidos arrays y subdocumentos (aunque esto sólo tiene sentido algunas veces). El valor de cada clave puede ser 1 (orden ascendente) o -1 (orden descendente).

En todo momento podemos preguntar por los índices asociados a una colección, como vemos a la derecha. Se aprecia que devuelve un array que en este ejemplo tiene dos índices:

1. El primero corresponde al **_id**. Se crea **automáticamente** al insertar el primer valor en la colección.
2. El segundo es el creado recientemente.

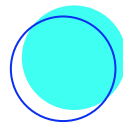
```
> db.cuentas.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_"
  },
  {
    "v" : 2,
    "key" : {
      "nombre" : 1,
      "direccion" : 1
    },
    "name" : "nombre_1_direccion_1"
  }
]
```

El valor "v" al principio es la versión del índice y tiene que ver con la versión de Mongo. Los índices creados con versiones anteriores a la 2.0 llevan versión "0" y los posteriores un valor superior.

Podemos ver además que internamente ambos índices tienen un nombre. Por ejemplo, el que se acaba de crear se llama "nombre_1_direccion_1".

Para borrar un índice se utilizará **dropIndex**, con el mismo parámetro que se usó para crearlo:

```
> db.cuentas.dropIndex({nombre:1,direccion:1})
{ "nIndexesWas" : 2, "ok" : 1 }
> db.cuentas.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
```



También se puede borrar el índice con **dropIndex(nombre)** con nombre como el valor name mostrado en `getIndexes`.

El valor del nombre también se puede indicar al crear el índice:



```
> db.cuentas.createIndex({nombre:-1},"inombresDesc")
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : false,
  "ok" : 1
}
> db.cuentas.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_"
  },
  {
    "v" : 2,
    "key" : {
      "nombre" : -1
    },
    "name" : "inombresDesc"
  }
]
```

Se borrará el índice por nombre:

```
> db.cuentas.dropIndex("inombresDesc")
{ "nIndexesWas" : 2, "ok" : 1 }

> db.cuentas.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
>
```

**Nota**

La creación/borrado de índices pueden ser operaciones muy costosas en colecciones grandes. Se suelen crear al principio y solo se borran en circunstancias especiales. En bases de datos, la creación y borrado de índices suele corresponder al DBA, no al programador.

**¡Sigamos
trabajando!**

