

MongoDB Fundamentos

Módulo 3

Tipos de índices en MongoDB

Los índices son estructuras de datos que se empaquetan con un conjunto de datos parcial dentro de sí mismo. El índice almacena el valor de un campo en particular en los documentos de manera ordenada.

Por defecto, los índices en MongoDB se almacenan en **árboles B** y admiten valores duplicados en las claves. Sin embargo, esto se puede modificar añadiendo una opción durante la creación del índice (como segundo parámetro).



Es decir, la sintaxis general para los índices es:

```
db.collectionName.createIndex(claves, opciones)
```

claves

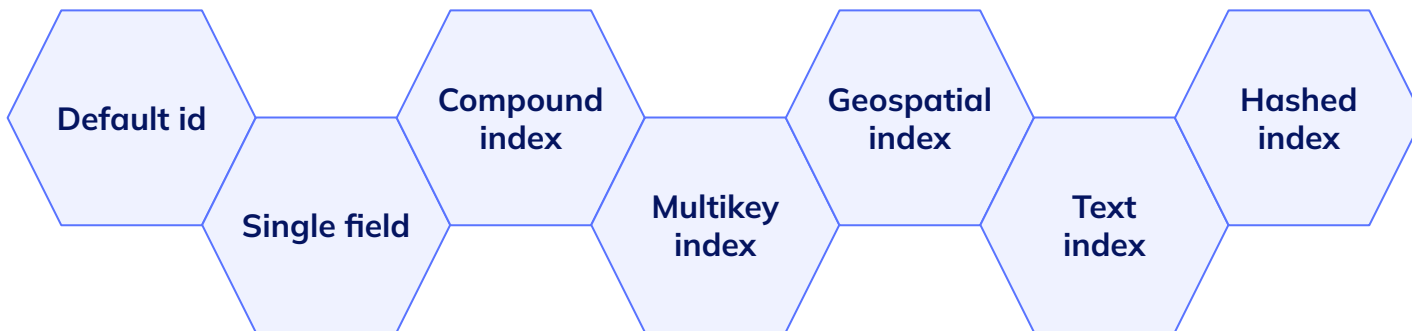
Es un documento que contiene un par de valores de campo, donde el campo es la clave de índice y el valor es el tipo de índice. El valor es 1 para ordenar la clave de índice en orden ascendente y -1 en orden descendente.

opciones

Es un documento que contiene un conjunto de opciones que influyen en la creación de índices. Este campo es opcional.

Tipos de índices

A continuación desarrollaremos los siguientes tipos de índices.



1. Identificación predeterminada (*Default id*)

Cada documento presente en la colección Mongo contiene un índice predeterminado llamado "**_id**". Se crea una identificación de objeto al crear el documento si no hay valores de índice presentes.



2. Campo único (*Single field*)

La indexación se realiza en un solo campo y la operación de clasificación es ascendente o descendente, ya que MongoDB puede atravesar en cualquier dirección.

Ejemplo:

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

También se puede crear un índice sobre un **subdocumento**. Veamos las siguientes slides.



```
> db.cuentas.createIndex({"cuentas.saldo":1})
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : true,
  "ok" : 1
}
> db.cuentas.insert({ cuentas: { saldo: 2000 } })
WriteResult({ "nInserted" : 1 })
> db.cuentas.find({"cuentas.saldo":{"$gt:1500}})
{ "_id" : ObjectId("626c0425f08ce3f407d15cee"), "cuentas" : { "saldo" : 2000 } }
```

```
> db.cuentas.explain().find({"cuentas.saldo":{$gt:1500}})
...
"stage" : "FETCH",
...
"inputStage" : {
  "stage" : "IXSCAN",
  "keyPattern" : {
    "cuentas.saldo" : 1
  },
  "indexName" : "cuentas.saldo_1",
  ...
}
```



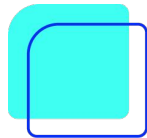
3. Índice compuesto (*Compound field*)

MongoDB admite índices definidos por el usuario en múltiples campos. El orden de los campos dados en el índice compuesto es bastante significativo. El orden de clasificación toma de izquierda a derecha, la prioridad para el primer campo mencionado en los índices compuestos es mayor que la del siguiente.

Ejemplo:

```
db.collection.createIndex( {"edad": 1, "dim.h": - 1} )
```

Todos los documentos con campo de edad se ordenan primero en orden ascendente y luego en orden descendente de altura en **dim**.



4. Índice Multikey (*Multikey Index*)

MongoDB usa el índice Multikey para indexar datos que están en formato de matriz. Durante la indexación, cada elemento de la matriz crea un índice separado y los datos se indexan en función de los elementos presentes en la matriz. MongoDB se encarga de que el campo de índice tenga una matriz por defecto.

Ojo con los **índices *multikey* compuestos**; si algún documento tiene arrays en más de una clave del índice se producirá un error.



Ejemplo no multikey:

```
> db.multi.createIndex({a:1,b:1})
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : true,
  "ok" : 1
}
> db.multi.insert({a:10,b:11})
WriteResult({ "nInserted" : 1 })
```

```
> db.multi.explain().find({a:4})
...
"stage" : "IXSCAN",
"indexName" : "a_1_b_1",
"isMultiKey" : false,
...
```


Ejemplo multikey:

```
> db.multi.insert({a:[5,4,3,2,1],b:-444})
WriteResult({ "nInserted" : 1 })
> db.multi.explain().find({a:4})
...
"stage" : "IXSCAN",
"indexName" : "a_1_b_1",
"isMultiKey" : true,
...
```

Ejemplo ERROR multikey:

```
> db.multi.insert({a:"hola",b:[123,56,7.68]})
WriteResult({ "nInserted" : 1 })

> db.multi.insert({a:[0,0,0],b:[123,56,7.68]})
WriteResult({
  "nInserted" : 0,
  "writeError" : { "code" : 171, "errmsg" : "cannot index parallel arrays [b] [a]" }
})
```



5. Índice geoespacial (*Geospatial Index*)

En general, las bases de datos NoSQL resultan adecuadas para búsquedas que tienen que ver con posiciones. MongoDB contiene sus propias librerías para este fin.

MongoDB utiliza la **indexación geoespacial** para buscar datos en función de su ubicación. Admite dos tipos de búsqueda, 2D (bidimensional) y 3D (tridimensional). Estos índices se utilizan para obtener resultados dentro de un rango. Esta funcionalidad no se habilita con una opción especial en el índice sino con un "*flag*" tras la clave.

Vamos a distinguir dos tipos: **2d-plano** y **2d-esférico**. Veamos las próximas pantallas.

Vamos a distinguir **dos tipos**:
2d-plano y **2d-esférico**.



Veamos las próximas pantallas.



2d-plano

Para crear un índice en 2d, es necesario que los documentos tengan **una clave que contenga un array de dos coordenadas**. Como en este código:

```
db.restaurants.insert{.... posicion:[23,-4]...}
```

Entonces se puede hacer:

```
db.restaurants.createIndex({ posicion: "2d" })
```

El flag 2d solo tiene sentido si `location` es una clave que tiene arrays con 2 números reales (la posición del objeto).

Estos índices son útiles cuando se usan operadores especiales como **\$near**

```
db.restaurants.find({posicion:{$near:[50,50]}).limit(5)
```

Este operador devuelve los cercanos a la posición por orden de cercanía. El número de resultados de búsqueda también se puede limitar mediante el uso de la función **limit ()**.

Otra característica interesante es la posibilidad de **consultar por los puntos que caen dentro de un polígono dado.**

Ejemplo: puntos dentro del triángulo (1,1), (1.5, 2)

```
db.plano.find({ posicion: { $geoWithin: { $polygon: [ [1,1 ], [1.5,2], [ 3 , 1 ] ] } } } )
```

```
db.plano.find({ posicion: { $geoWithin: { $polygon: [ [1,1 ], [1.5,2], [ 3 , 1 ] ] } } } )
```



2d-esférico

En casos sencillos, o para aplicaciones sobre el plano, las coordenadas 2d son suficientes. Para distancias sobre el globo terrestre, es necesario utilizar las coordenadas "2dsphere", que están formadas por parejas [longitud, latitud]

- **Longitud:** distancia en grados al meridiano de Grenwich. Se usa la notación (- 180,+180) (negativos al oeste, positivos al este) .
- **Latitud:** distancia en grados al ecuador (-90 hasta 90).

Los documentos son un poco más complicados que en el caso anterior. Veamos el ejemplo de la próxima pantalla.



```
db.restaurants.insert({
  "nombre" : "El espagueti enrollado",
  "comida" : "Italiana",
  "gps" : {
    "type" : "Point",
    "coordinates" : [
      12.5386254,
      41.8839509
    ]
  }
})
```

La parte que interesa, es la de la clave **gps**, que es la que servirá para crear el índice.

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

El nombre de esta clave puede ser cualquiera que se elija (gps, lugar, localización...). Lo importante es la estructura del valor que la acompaña, que debe ser una valor admitido en **GeoJSON**.

En particular:

- El valor **"type"** puede ser "Point", "polygon", y varios más.
- Si el valor **"type"** es **"Point"**, entonces debe haber una **clave coordinates**, con valores en un array de dos dimensiones. El primer valor será la longitud (entre -180 y +180) y el segundo, la latitud (entre -90 y 90).

Ahora se pueden hacer consultas como:

```
db.restaurants.find( { 'gps' : { $near : {  
  $geometry: { type: 'Point', coordinates: [<lon>, <lat>] } ,  $maxDistance: <meters> }} } );
```

que busca objetos alrededor de las coordenadas
<lon>, **<lat>** dentro de un radio de **<meters>**
metros. La sintaxis completa del comando es:

```
{ $near: {  
  $geometry: {  
    type: "Point" ,  
    coordinates: [ <longitude> , <latitude> ]  
  },  
  $maxDistance: <distance in meters>,  
  $minDistance: <distance in meters>  
}  
}
```



Acerca del GeoJSON

Es un formato para codificar una variedad de estructuras de datos geográficos.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

GeoJSON admite los siguientes tipos de geometría:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon

Los objetos geométricos con propiedades adicionales son **objetos Feature**. Los conjuntos de características están contenidos en **colecciones de objetos Feature**.

6. Índice de texto (*Text Index*)

MongoDB proporciona indexación de texto para admitir consultas en formato de cadena. El índice de texto puede tener cualquier campo que consista en elementos de cadena o una matriz de elementos de cadena.

Ejemplo:

```
db.movies.createIndex({actor:"text"})
```

Por ejemplo

```
db.movies.find( { $text: { $search:"Tom Hardy" } } )
```

encontrará los documentos que tienen nombres de actores como Tom Hanks, Tom Felton, Tom Hiddleston, Robert Hardy y John Hardy. MongoDB toma la cadena provista para la búsqueda y proporciona todos los documentos que tienen una cadena de búsqueda completa o parcial.

```
> db.movies.createIndex({actor:"text"})
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : true,
  "ok" : 1
}

> db.movies.insert({ actor: "tom hanks" })
WriteResult({ "nInserted" : 1 })
> db.movies.insert({ actor: "tom felton" })
WriteResult({ "nInserted" : 1 })
> db.movies.insert({ actor: "tom hiddelson" })
WriteResult({ "nInserted" : 1 })
> db.movies.insert({ actor: "robert hardy" })
WriteResult({ "nInserted" : 1 })
> db.movies.insert({ actor: "John Hardy" })
WriteResult({ "nInserted" : 1 })
> db.movies.insert({ actor: "Sylvester Stallone" })
WriteResult({ "nInserted" : 1 })
```



```
> db.movies.insert({ actor: "Chuck Norris" })
WriteResult({ "nInserted" : 1 })

> db.movies.find( { $text: { $search: "tom hardy" } } )
{ "_id" : ObjectId("626c0883f08ce3f407d15cfd"), "actor" : "John Hardy" } { "_id" :
ObjectId("626c087ef08ce3f407d15cfc"), "actor" : "robert hardy" } { "_id" :
ObjectId("626c0877f08ce3f407d15cfb"), "actor" : "tom hiddelton" } { "_id" :
ObjectId("626c0870f08ce3f407d15cfa"), "actor" : "tom felton" } { "_id" :
ObjectId("626c086cf08ce3f407d15cf9"), "actor" : "tom hanks" }

> db.movies.find( { $text: { $search: "tom" } } )
{ "_id" : ObjectId("626c0877f08ce3f407d15cfb"), "actor" : "tom hiddelton" } { "_id" :
ObjectId("626c0870f08ce3f407d15cfa"), "actor" : "tom felton" } { "_id" :
ObjectId("626c086cf08ce3f407d15cf9"), "actor" : "tom hanks" }

> db.movies.find( { $text: { $search: "hardy" } } )
{ "_id" : ObjectId("626c0883f08ce3f407d15cfd"), "actor" : "John Hardy" } { "_id" :
ObjectId("626c087ef08ce3f407d15cfc"), "actor" : "robert hardy" }

> db.movies.find( { $text: { $search: "chuck" } } )
{ "_id" : ObjectId("626c088df08ce3f407d15cff"), "actor" : "Chuck Norris" }
```

7. Índice hash (*Hash Index*)

MongoDB utiliza un índice **hash** para admitir la fragmentación. Los **índices hash** calculan el valor **hash** de los campos de índice utilizando una función **hash**. No es compatible con la indexación de múltiples claves (valores de matriz).

Los índices hash se crean utilizando la función **createIndex** y el valor del campo índice siempre debe ser '**hash**'.

Ejemplo:

```
db.collection.createIndex ( { campo: "hashed" } )
```

Para encontrar los documentos con valores **hash**, **db.collection.find ({ campo: Math.pow (2, 63) })** devolverá todos los documentos con índices **hash** en el rango $2 \wedge 63$.



**¡Sigamos
trabajando!**

