Bases de datos NoSQL

//Introducción a Mongo DB

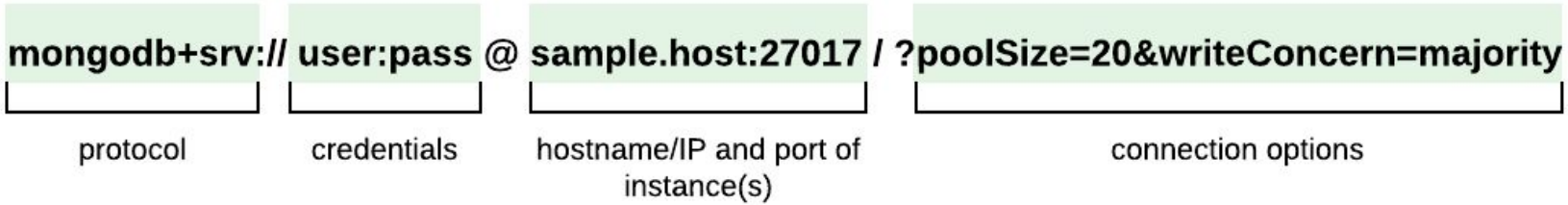
IT BOARDING

BOOTCAMP

// Mientras tanto...

1. Descargar e instalar Compass, un cliente con interfaz gráfica —
https://docs.mongodb.com/compass/current/install

2. Conectarse al cluster del curso, un conjunto de servidores, deployado en la nube —
mongodb+srv://dh:1234@cluster0.qpm27.mongodb.net/



Base de datos de documentos

Un registro en MongoDB es un documento, una estructura de datos compuesta por pares de campos y valores, similares a objetos JSON. Los valores de los campos pueden incluir:

- otros documentos,
- listas y
- listas de documentos.

```
{
    "_id": "59a47286cfa9a3a73e51e736",
    "theaterId": 1017,
    "location": {
        "address": {
            "street1": "4325 Sunset Dr",
            "city": "San Angelo",
            "state": "TX",
            "zipcode": "76904"
        },
        "geo": {
            "type": "Point",
            "coordinates": [-100.50107, 31.435648]
        }
    }
}
```

Las ventajas de usar documentos

- Los documentos —objetos— se corresponden con tipos de datos nativos en muchos lenguajes de programación.
- Los documentos embebidos y las listas reducen la necesidad de joins costosos.
- El esquema flexible permite polimorfismo.

De SQL a MongoDB

SQL	MongoDB
database	database
table	collection
row	document
column	field
index	index
table joins	$lookup, embedded documents
primary key (unique column or column combination)	primary key (_id field)
aggregation (group by)	aggregation pipeline

El campo _id

En MongoDB cada documento almacenado en una colección requiere un campo _id que actúa como llave primaria. Si un documento nuevo omite este campo, se genera automáticamente.

Create Read Update Delete — CRUD

Ver más: https://docs.mongodb.com/manual/crud

Ejemplos de queries de documentos. Asumimos que hay una colección llamada people que contiene documentos de esta forma:

```
{
    _id: "509a8fb2f3f4948bd2f983a0",
    user_id: "abc123",
    age: 55,
    status: "A"
}
```

SQL

```
SELECT *
FROM people
```

Mongo Shell

```
db.people.find()
```

SQL

```
SELECT id,
    user_id,
    status
FROM people
```

Mongo Shell

```
db.people.find(
  { },
  { user_id: 1, status: 1 }
)
```

En Mongo la selección de campos específicos se denomina **proyección**.

SQL

```
SELECT user_id, status
FROM people
```

Mongo Shell

```
db.people.find(
  { },
  { user_id: 1, status: 1, _id: 0 }
)
```

Si no queremos la *primary key* hay que pedirlo explícitamente.

SQL

```
SELECT *
FROM people
WHERE status = "A"
```

Mongo Shell

```
db.people.find(
   { status: "A" }
)
```

SQL

```
SELECT user_id, status
FROM people
WHERE status = "A"
```

Mongo Shell

```
db.people.find(
   { status: "A" },
   { user_id: 1, status: 1, _id: 0 }
)
```

SQL

```
SELECT *
FROM people
WHERE status <> "A"
```

Mongo Shell

```
db.people.find(
   { status: { $ne: "A" } }
)
```

$ne es *not equal*.

SQL

```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

Mongo Shell

```
db.people.find(
   { status: "A", age: 50 }
)
```

SQL

```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

Mongo Shell

```
db.people.find(
   { $or: [ { status: "A" } , { age: 50 } ] }
)
```

SQL

```
SELECT *
FROM people
WHERE status IN ["A", "B"]
```

Mongo Shell

```
db.people.find(
   { status: { $in: ["A", "B"] } }
)
```

Para distintos valores de un mismo campo, usar $in en vez de $or.

SQL

```
SELECT *
FROM people
WHERE age > 25
```

Mongo Shell

```
db.people.find(
  { age: { $gt: 25 } }
)
```

$gt es *greater than*. $gte es *greater than or equal*.

SQL

```
SELECT *
FROM people
WHERE age < 25
```

Mongo Shell

```
db.people.find(
  { age: { $lt: 25 } }
)
```

$lt es *less than*. $lte es *less than or equal*.

SQL

```
SELECT *
FROM people
WHERE age > 25
AND   age <= 50
```

Mongo Shell

```
db.people.find(
  { age: { $gt: 25, $lte: 50 } }
)
```

SQL

```
SELECT *
FROM people
WHERE user_id LIKE "%bc%"
```

Mongo Shell

```
db.people.find(
  { user_id: /bc/ }
)
```

o también

```
db.people.find(
  { user_id: { $regex: /bc/ } }
)
```

$regex es *regular expression*.

SQL

```
SELECT *
FROM people
WHERE user_id LIKE "bc%"
```

Mongo Shell

```
db.people.find(
  { user_id: /^bc/ }
)
```

o también

```
db.people.find(
  { user_id: { $regex: /^bc/ } }
)
```

SQL

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
```

Mongo Shell

```
db.people.find(
   { status: "A" }
).sort(
   { user_id: 1 }
)
```

SQL

```
SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id DESC
```

Mongo Shell

```
db.people.find(
   { status: "A" }
).sort(
   { user_id: -1 }
)
```

SQL

```
SELECT COUNT(*)
FROM people
```

Mongo Shell

```
db.people.find().count()
```

o también

```
db.people.count()
```

SQL

```
SELECT COUNT(user_id)
FROM people
```

Mongo Shell

```
db.people.find(
  { user_id: { $ne: null } }
).count()
```

o también

```
db.people.count(
  { user_id: { $ne: null } }
)
```

SQL

```
SELECT COUNT(user_id)
FROM people
```

Mongo Shell

```
db.people.find(
  { user_id: { $exists: true } }
).count()
```

$exists funciona parecido a un *is not null*, con la diferencia de **si el campo existe pero guarda un valor nulo, cuenta**.

SQL

```
SELECT *
FROM people
WHERE status IS NULL
```

Mongo Shell

```
db.people.find(
  { status: null }
).count()
```

Devuelve documentos que contienen el **campo con valor nulo**, tanto como los que **no poseen el campo**.

SQL

```
SELECT COUNT(*)
FROM people
WHERE age > 30
```

Mongo Shell

```
db.people.find(
  { age: { $gt: 30 } }
).count()
```

o también

```
db.people.count(
  { age: { $gt: 30 } }
)
```

SQL

```
SELECT DISTINCT(status)
FROM people
```

Mongo Shell

```
db.people.distinct( "status" )
```

SQL

```
SELECT *
FROM people
LIMIT 1
```

Mongo Shell

```
db.people.findOne()
```

o también

```
db.people.find().limit(1)
```

SQL

```
SELECT *
FROM people
LIMIT 5
SKIP 10
```

Mongo Shell

```
db.people.find().limit(5).skip(10)
```

Consultas NoSQL

Ver más: https://docs.mongodb.com/manual/tutorial/query-documents

Para los siguientes ejemplos supongamos documentos como este en una colección llamada inventory.

```
{
   _id: "509a8fb2f3f4948bd2f983a0",
   item: "journal",
   size: { h: 14, w: 21, uom: "cm" },
   tags: ["red", "blank"],
   instock: [
      { qty: 5, warehouse: "A" },
      { qty: 1, warehouse: "B" }
   ]
}
```

Documentos anidados

Match exacto. Requiere que incluso el orden de los campos sea el mismo.

```
db.inventory.find(
  { size: { h: 14, w: 21, uom: "cm" } }
)
```

Campos anidados

Usamos *dot notation* (campo.campoAnidado) para especificar condiciones en campos anidados.

```
db.inventory.find(
  { "size.h": { $lt: 15 }, "size.uom": "cm" }
)
```

Listas

Match exacto. El orden de los elementos importa.

```
db.inventory.find(
   { tags: ["red", "blank"] }
)
```

Cualquier elemento

```
db.inventory.find(
   { tags: "blank" }
)
```

Elemento en posición específica

```
db.inventory.find(
  { "tags.0": "red" }
)
```

MongoDB empieza a contar las posiciones en las listas desde 0.

Largo de la lista

```
db.inventory.find(
  { tags: { $size: 3 } }
)
```

Listas de documentos

Match exacto. La lista debe contener al menos un documento con mismo orden de campos y mismos valores.

```
db.inventory.find(
  { "instock": { qty: 5, warehouse: "A" } }
)
```

Campos anidados

```
db.inventory.find(
  { "instock.qty": { $lte: 20 }, "instock.warehouse": "A" }
)
```

Documento en posición específica

```
db.inventory.find(
  { "instock.0.qty": { $lte: 20 } }
)
```

Modelo de datos

Ver más: https://docs.mongodb.com/manual/core/data-modeling-introduction

Esquema flexible

A diferencia de las bases de datos SQL, donde las tablas poseen un esquema predefinido, las colecciones de MongoDB —por defecto— **no requieren que sus documentos posean el mismo esquema**.

- No necesitan tener el mismo conjunto de campos ni mantener el tipo de dato en un mismo campo.
- Para agregar nuevos campos, remover existentes, o cambiar el tipo de dato de un valor, solo hay que actualizar el documento a su nueva estructura.

Esta flexibilidad facilita el mapeo de documentos a entidades u objetos.

En la práctica los documentos de una colección comparten una estructura similar y es posible de ser conveniente establecer reglas de **validación de esquemas**.

De-normalizado

En general, usar **documentos embebidos** cuando

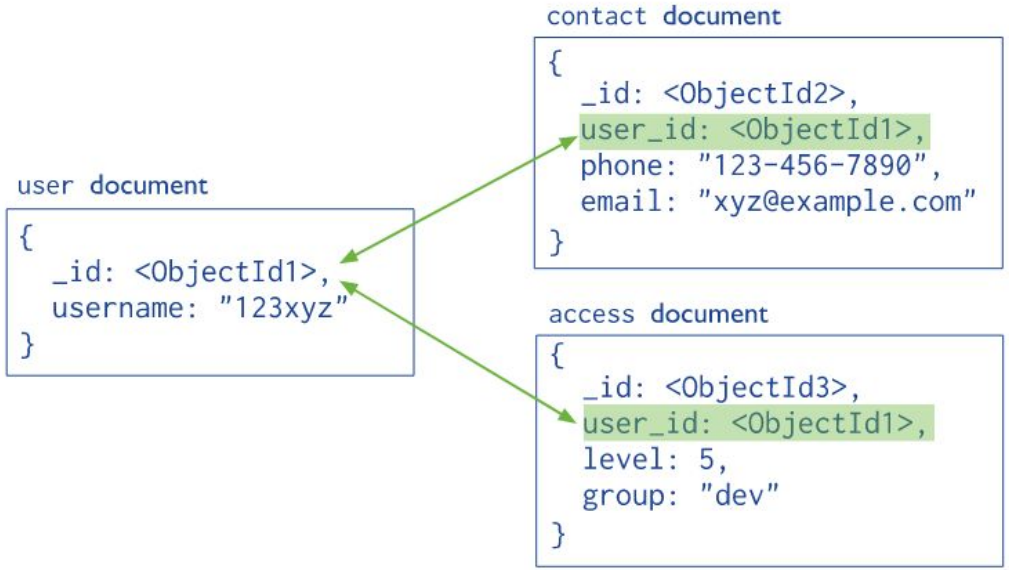
- hay relaciones uno-a-uno,
- y en relaciones uno-a-muchos si es que "los muchos" siempre hacen falta en contexto "del uno".

La lectura es más eficiente al devolver menos documentos. Asimismo hace que la escritura de datos relacionados sea atómica.

Normalizado

En general, **normalizar** documentos cuando:

- Anidarlos resulta en duplicación de los datos pero no en suficiente rendimiento de lectura como para justificar la duplicación,
- Hay relaciones muchos-a-muchos,
- Jerárquicas (árboles),
- Complejas (redes).



Agregaciones

Ver más: https://docs.mongodb.com/manual/aggregation

MongoDB utiliza una forma de procesamiento de datos llamada *aggregation pipeline* en la que los **documentos** atraviesan distintas etapas que los van transformando y agregando.

Colección orders

```
db.orders.insertMany([
   { _id: 1, customer_id: 1, item: "almonds", price: 12, quantity: 2 },
   { _id: 2, customer_id: 1, item: "pecans",  price: 20, quantity: 1 },
   { _id: 3, customer_id: 2, item: "pecans",  price: 20, quantity: 5 }
])
```

SQL

```
SELECT customer_id AS _id,
    SUM(price * quantity) AS total
FROM orders
GROUP BY customer_id
```

Mongo Shell

```
db.orders.aggregate([
  { $project: { customer_id: 1, subtotal: { $multiply: [ "$price", "$quantity" ] } } },
  { $group: { _id: "$customer_id", total: { $sum: "$subtotal" } } }
])
```

Resultado

{ _id: 1, total: 44 }
{ _id: 2, total: 100 }

Primera etapa: $project pasa documentos a la siguiente etapa con los campos requeridos, existentes o nuevos.

Segunda etapa: $group es como la cláusula GROUP BY de SQL, agrega por los campos definidos en _id.

El *aggregation pipeline* es realmente flexible. Existen etapas para una gran cantidad de casos de uso.

$sum es uno de los varios operadores de agregación; dentro de $group se comporta como acumulador. Otros acumuladores son $avg (promedio), $min y $max.

Combinaciones

Ver más: https://docs.mongodb.com/manual/reference/operator/aggregation/lookup

$lookup es una etapa de agregación que realiza un left join entre colecciones. A cada documento de la izquierda se le **agrega un nuevo campo** del tipo lista con documentos "joineados" de la derecha.

Colección orders

```
db.orders.insertMany([
  { _id: 1, item: "almonds", price: 12, quantity: 2 },
  { _id: 2, item: "pecans",  price: 20, quantity: 1 },
])
```

Colección inventory

```
db.inventory.insertMany([
  { _id: 1, sku: "almonds", description: "product 1", instock: 120 },
  { _id: 2, sku: "bread",   description: "product 2", instock: 80 },
])
```

Agregación

```
db.orders.aggregate([
  {
    $lookup:
      {
        from: "inventory",
        localField: "item",
        foreignField: "sku",
        as: "inventory_docs"
      }
  }
])
```

Resultado

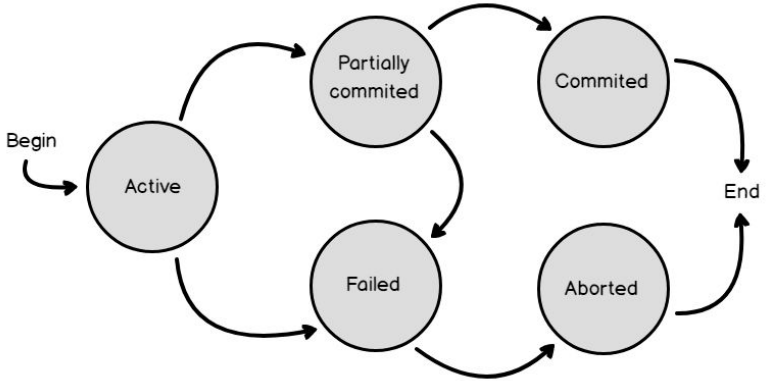
```
{
  _id: 1,
  item: "almonds",
  price: 12,
  quantity: 2,
  inventory_docs: [
    { _id: 1, sku: "almonds", description: "product 1", instock: 120 }
  ]
}
```

Transacciones / Atomicidad

Ver más: https://docs.mongodb.com/manual/core/transactions

En MongoDB una operación sobre un único documento es **atómica**. Es posible usar documentos embebidos o anidados para capturar las relaciones entre los datos **en un único documento** en vez de normalizar los datos en múltiples documentos y colecciones. Esta característica evita recurrir a transacciones multi-documento en muchos casos de uso.

En situaciones en las que se requiera atomicidad para leer y escribir múltiples documentos, sin importar si se encuentran en la misma colección o no), MongoDB soporta **transacciones multi-documento** (es una proposición *todo o nada*).

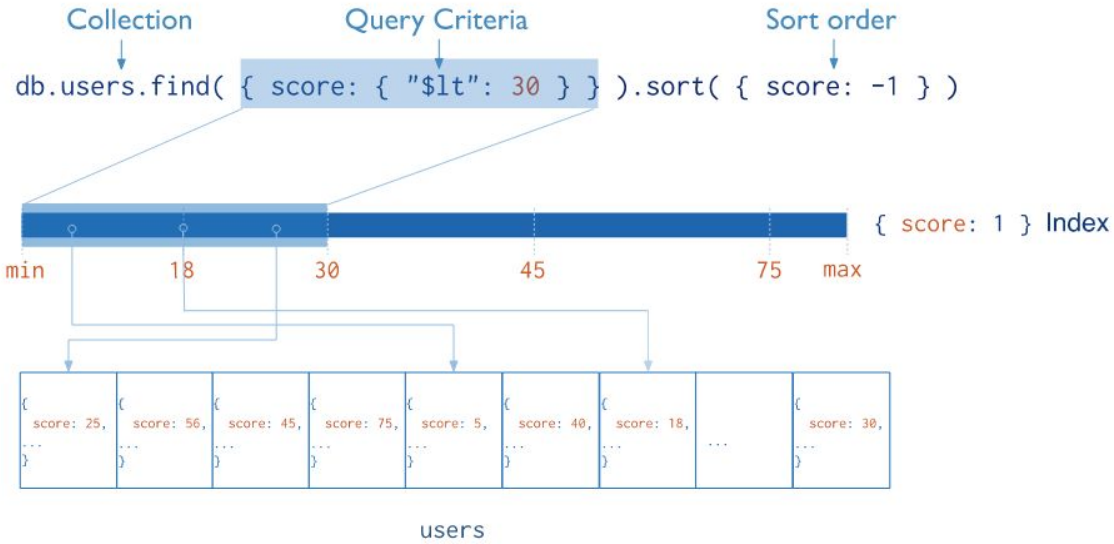


Índices

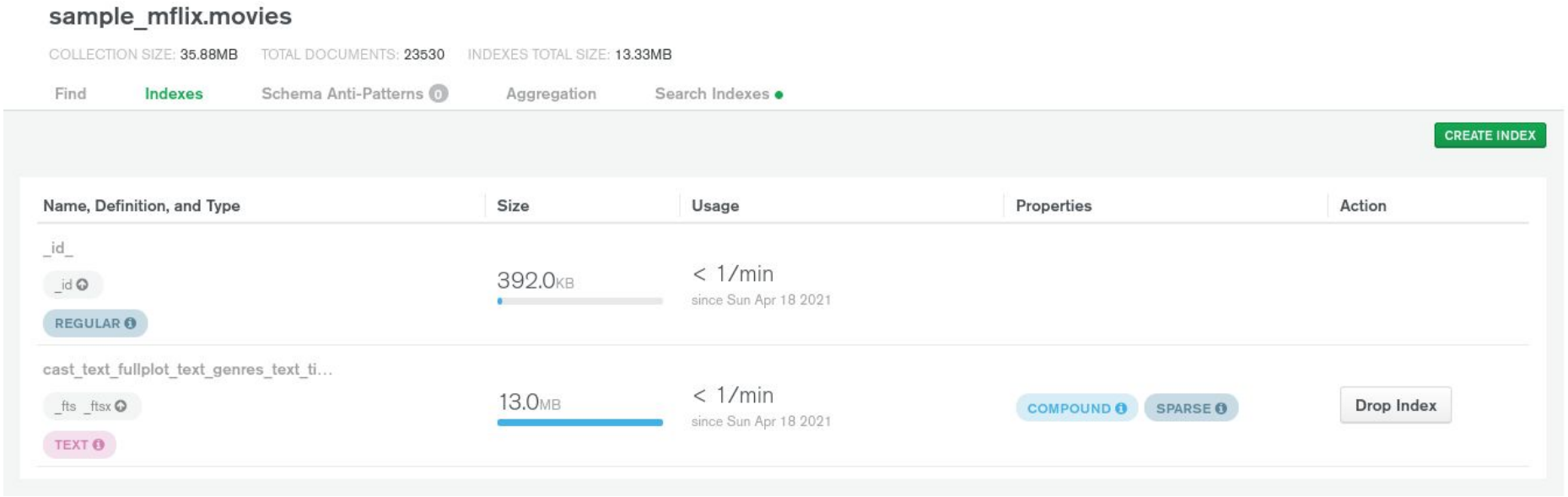
Ver más: https://docs.mongodb.com/manual/indexes

Habilitan la ejecución eficiente de consultas. Sin índices el motor de MongoDB debe revisar cada documento de la colección para seleccionar aquellos que cumplen con el filtro.

En cambio, si existen índices apropiados para la consulta, el motor los puede usar para limitar la cantidad de documentos que debe inspeccionar.



Los índices aceleran las consultas. Como contrapartida, son estructuras que ocupan espacio de almacenamiento.

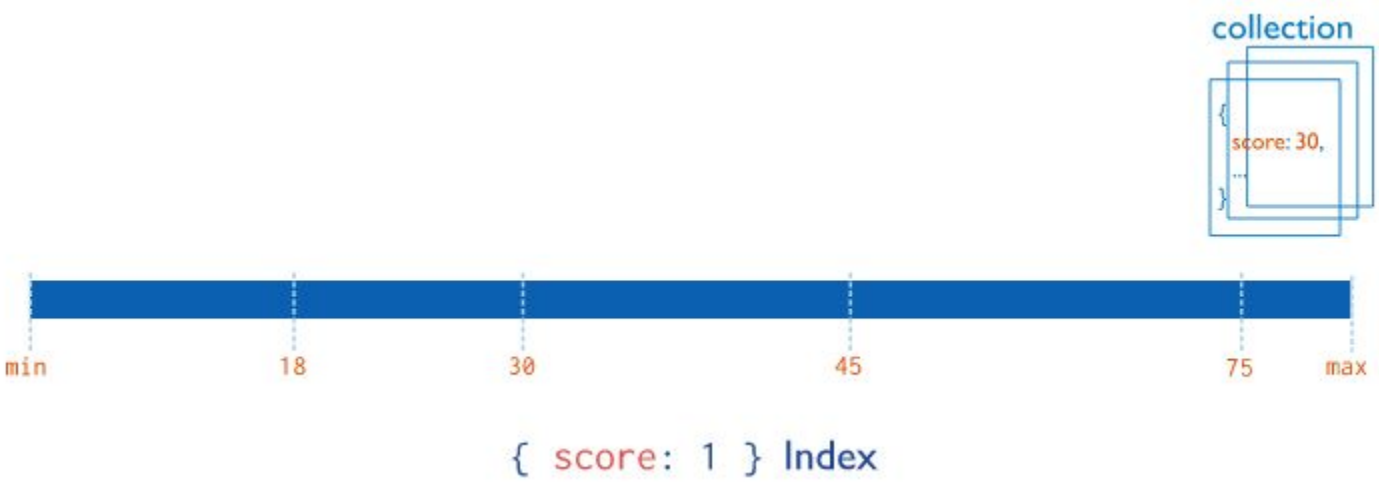


Índice por defecto

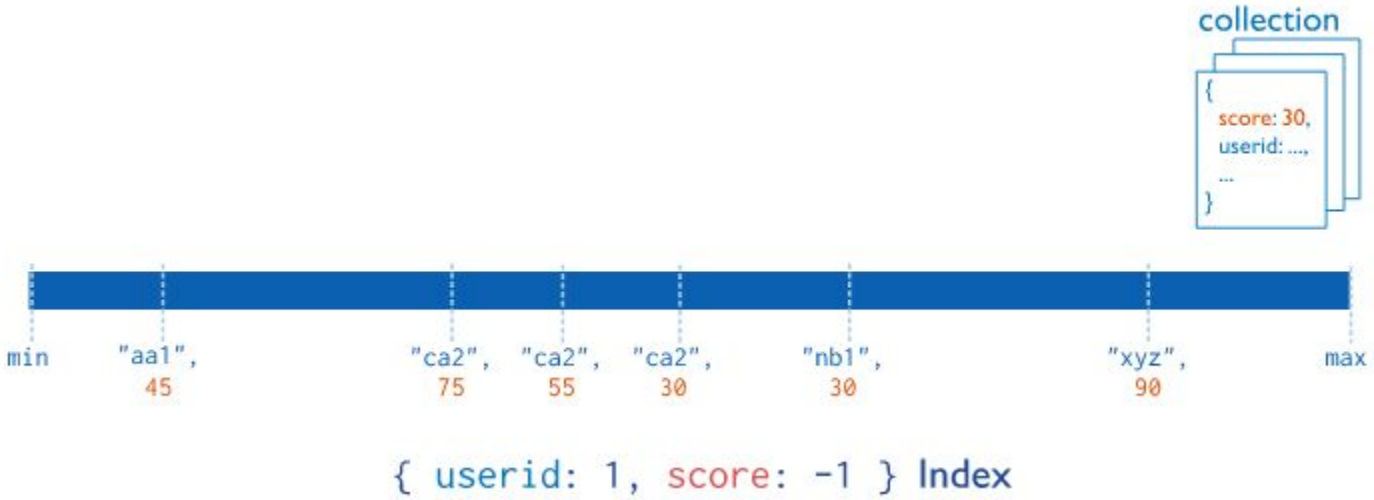
MongoDB al crear una colección crea un índice sobre el campo _id que previene la inserción de dos documentos con el mismo valor. Este índice no puede ser eliminado.

Tipos de índices

Simple



Compuesto


collection
{
score: 30,
userid: ...,
...
}
min
"aa1",
45
"ca2",
75
"ca2",
55
"ca2",
30
"nb1",
30
"xyz",
90
max
{ userid: 1, score: -1 } Index

Geoespacial

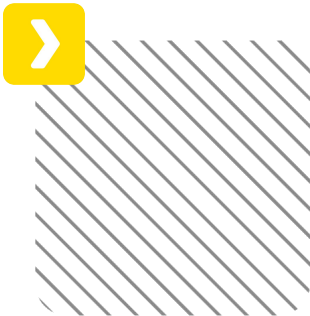
Índices bidimensionales sobre geometría planar o esférica.

Texto

No repara en *stop words* y procesa las palabras para almacenar únicamente sus raíces.

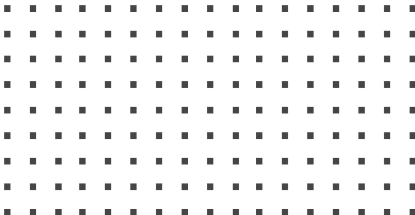
Hash

Distribuye aleatoriamente los valores en su rango. Útil para *sharding*.



Propiedades de los índices

- **Único**. Rechaza valores duplicados en el campo indexado.
- **Parcial**. Solo indexa documentos que cumplen con un filtro determinado.
- **TTL** (*time-to-live*). Remueve automáticamente documentos luego de cierto tiempo.
- **Ocultos**. No pueden ser utilizados en las consultas. Sirve para desactivarlo temporalmente.



Gracias.

IT BOARDING

BOOTCAMP