Filtrar datos en MongoDB – De SQL a NoSQL

Objetivo

Aprender a utilizar filtros en MongoDB para buscar documentos específicos, **entendiendo su lógica, su estructura y su diferencia con las consultas SQL**. Este apunte no solo explica la sintaxis, sino el **por qué se hace así**, comparando con el modelo relacional tradicional.

¿Qué es un filtro en una consulta?

Cuando trabajamos con una base de datos, **no siempre queremos traer todos los datos**, sino solamente aquellos que cumplen cierta condición. A eso lo llamamos **filtrar**.

- En SQL usamos la cláusula `WHERE` para filtrar.
- En MongoDB usamos el **primer parámetro** del método `.find()` para aplicar filtros.

Diferencia conceptual: condiciones SQL vs. objetos JSON

En SQL escribimos condiciones en forma de texto estructurado:

```
SELECT * FROM empleados WHERE edad = 30
```

En MongoDB, no usamos texto sino un **objeto JSON que representa la condición**:

```
db.empleados.find({ edad: 30 })
```

Acá, en vez de escribir `edad = 30` como una frase, **armamos un objeto donde el campo `edad` tenga el valor de `30`**.

Con esta consulta lo que estamos diciéndole a mongodb es "Traeme todos los **documentos** de la **colección empleados**, donde el campo **edad** sea **igual que 30**."

Esto es clave: **en MongoDB el filtro mismo es un mini-documento que describe a los documentos que queremos encontrar.**

¿Por qué se usan objetos y no texto como en SQL?

SQL fue pensado para trabajar con estructuras **tabulares y fijas** (tablas, columnas). En cambio, MongoDB está basado en documentos JSON, y su forma natural de expresarse es también con **estructuras de datos**, no con texto.

Por eso en MongoDB **las consultas se escriben como objetos**, y no como frases ya que es la forma natural de expresar documentos.

Equivalencias SQL vs. MongoDB

Descripción	SQL	MongoDB
Todos los empleados	`SELECT * FROM empleados`	`db.empleados.find({})`
Edad igual a 30	`WHERE edad = 30`	`{ edad: 30 }`
Edad mayor a 30	`WHERE edad > 30`	`{ edad: { $gt: 30 } }`
Edad mayor o igual a 30	`WHERE edad >= 30`	`{ edad: { $gte: 30 } }`
Edad menor a 30	`WHERE edad < 30`	`{ edad: { $lt: 30 } }`
Edad diferente de 30	`WHERE edad != 30`	`{ edad: { $ne: 30 } }`
Nombre dentro de una lista	`WHERE nombre IN ('Juan', 'Ana')`	`{ nombre: { $in: ["Juan", "Ana"] } }`
Nombre fuera de una lista	`WHERE nombre NOT IN ('Juan', 'Ana')`	`{ nombre: { $nin: ["Juan", "Ana"] } }`

Operadores relacionales más usados

En SQL usamos `>`, `<`, `!=`, `>=`… En MongoDB usamos **palabras clave precedidas por el signo $**:

Operador	Significado	Ejemplo MongoDB
`$gt`	Mayor que	`{ edad: { $gt: 40 } }`
`$lt`	Menor que	`{ edad: { $lt: 25 } }`
`$gte`	Mayor o igual	`{ edad: { $gte: 18 } }`
`$lte`	Menor o igual	`{ edad: { $lte: 65 } }`
`$ne`	Distinto de	`{ edad: { $ne: 30 } }`
`$in`	Dentro de un listado	`{ nombre: { $in: ["Ana", "Juan"] } }`
`$nin`	No está en un listado	`{ nombre: { $nin: ["Ana"] } }`

¿Por qué se usan $gt, $lt y no los símbolos directamente?

En JSON **no se pueden usar operadores como >, <, != dentro de las claves**. Entonces MongoDB define **operadores especiales como palabras**.

¿Qué estructura devuelve .find()?

`.find()` **no devuelve un solo dato**, sino un **cursor** que puede contener **uno o varios documentos**.

```
db.empleados.find({ edad: 30 }).pretty()
```

Si existen 3 empleados de 30 años, devuelve los 3. Si no hay ninguno, devuelve vacío.

Si querés ver solo uno, usá `.findOne()`, que devuelve **el primer documento que coincida**.

Filtros con múltiples condiciones

1. Todas las condiciones deben cumplirse (AND implícito)

```
db.empleados.find({ edad: { $gt: 30 }, puesto: "Desarrollador" })
```

Esto es equivalente a:

```
SELECT * FROM empleados WHERE edad > 30 AND puesto = 'Desarrollador'
```

2. Alguna de las condiciones debe cumplirse (OR explícito)

```
db.empleados.find({
  $or: [
    { puesto: "Diseñador" },
    { puesto: "Tester" }
  ]
})
```

Esto se traduce como:

```
SELECT * FROM empleados
WHERE puesto = 'Diseñador' OR puesto = 'Tester'
```

Otros operadores lógicos útiles

Operador	Significado	Ejemplo
`$and`	Todas las condiciones deben cumplirse	`{ $and: [ { edad: { $gte: 25 } }, { puesto: "Tester" } ]}`

`$or`	Alguna condición debe cumplirse	`{$or:[ {edad: { $lt: 25 } }, {puesto: "Tester" } ]}`
`$nor`	Ninguna de las condiciones debe cumplirse	`{ $nor: [ { puesto: "Jefe" }, { edad: { $lt: 18 }} ]}`
`$not`	Niega una condición específica	`{ edad: { $not: { $gt: 40 } } }` (trae menores o igual a 40)

`$and` muchas veces no es necesario, porque MongoDB ya lo aplica implícitamente si ponés varias condiciones al mismo nivel.

Filtros combinados más complejos

Buscar empleados entre 30 y 50 años

```
db.empleados.find({
  edad: { $gte: 30, $lte: 50 }
})
```

Dentro del mismo campo (`edad`) podés combinar operadores.

Empleados mayores de 40 que no sean "Jefes"

```
db.empleados.find({
  edad: { $gt: 40 },
  puesto: { $ne: "Jefe" }
})
```

Buscar por coincidencia parcial – $regex

```
db.empleados.find({
  nombre: { $regex: "ana", $options: "i" }
})
```

- `"ana"`: busca coincidencias con ese texto.
- `"i"`: ignora mayúsculas y minúsculas.

Equivalente en SQL:

```
WHERE nombre LIKE '%ana%'
```

Atención: las búsquedas con `$regex` no usan índices, así que pueden ser más lentas en colecciones grandes.

¿Qué pasa si el campo no existe?

Si buscás algo como:

```
db.empleados.find({ telefono: { $exists: false } })
```

Estás diciendo: "Buscá los documentos donde el campo `telefono` **no esté presente**".
Esto no existe en SQL tal como lo ves acá, ¡pero es muy útil en MongoDB!

Conclusiones clave

- Las condiciones de búsqueda en MongoDB se expresan como objetos JSON.
- Los filtros funcionan de forma similar al WHERE de SQL, pero con estructuras de datos, no con texto.
- Los operadores como `$gt`, `$in`, `$or` permiten construir filtros complejos de forma clara y flexible.
- MongoDB devuelve **todos los documentos que cumplan la condición**, no una fila única como en SQL Server (a menos que uses `.findOne()`).