Ordenamiento y Paginación en MongoDB – De SQL a NoSQL

Objetivo

Aprender a ordenar y paginar resultados en MongoDB, comprendiendo cómo funcionan estas operaciones, su diferencia con el enfoque relacional y cómo se expresan usando estructuras JSON.

¿Qué es ordenar en una consulta?

Cuando obtenemos muchos resultados de una búsqueda, a veces queremos que estén **ordenados de cierta manera**: por edad, por nombre, por fecha, etc.
En SQL usamos `ORDER BY`.
En MongoDB usamos el método `.sort()`.

SQL vs MongoDB – Ordenamiento

Descripción	SQL	MongoDB
Todos los empleados	`SELECT * FROM empleados`	`db.empleados.find({})`
Ordenados por edad ascendente	`ORDER BY edad ASC`	`.sort({ edad: 1 })`
Ordenados por edad descendente	`ORDER BY edad DESC`	`.sort({ edad: -1 })`

Ordenar por múltiples campos	`ORDER BY edad ASC, nombre DESC`	`.sort({ edad: 1, nombre: -1 })`

¿Por qué 1 y -1 y no ASC y DESC?

MongoDB no usa texto para definir el orden, sino **valores numéricos**:

- `1` significa **ascendente**
- `-1` significa **descendente**

Esto se debe a que en JSON no se pueden usar palabras clave como ASC o DESC dentro de los objetos. En cambio, usamos valores numéricos, que también resultan más fáciles de procesar por el motor de MongoDB.

Ejemplo práctico de ordenamiento

```
db.empleados.find({}).sort({ edad: 1 })
```

Traducción:
"Traeme todos los empleados, ordenados por edad de menor a mayor."

¿Qué es paginar en una consulta?

Cuando una colección tiene muchos documentos, no siempre queremos traerlos todos de una.
En SQL usamos `LIMIT` y `OFFSET`.
En MongoDB usamos los métodos **`.limit()`** y **`.skip()`**.

SQL vs MongoDB – Paginación

Descripción	SQL	MongoDB
Traer los primeros 5 empleados	`SELECT * FROM empleados LIMIT 5`	`.find({}).limit(5)`
Saltar los primeros 5 y traer los siguientes 5	`OFFSET 5 LIMIT 5`	`.find({}).skip(5).limit(5)`

Ejemplo de paginación

```
db.empleados.find({}).skip(5).limit(5)
```

Traducción:
"Salteate los primeros 5 resultados y traé los siguientes 5."

Esto es útil para mostrar los resultados **por páginas**, como si fuera una tabla con "Página 1, Página 2, Página 3..."

Tip: Ordenar antes de paginar

Si usás `.skip()` y `.limit()`, **siempre es recomendable ordenar antes con `.sort()`**, para garantizar que los resultados sean **consistentes** entre una página y otra.

```
db.empleados.find({}).sort({ apellido: 1 }).skip(10).limit(10)
```

¿Qué devuelve .sort() y .limit()?

Ambos métodos se aplican **sobre el resultado de un `.find()`**.
El flujo general es:

```
db.coleccion.find(filtro).sort(orden).skip(n).limit(m)
```

Se pueden encadenar como si fuera una receta paso a paso:

1. Buscar con `.find()`
2. Ordenar con `.sort()`
3. Saltar con `.skip()`
4. Limitar con `.limit()`

¿Cómo sería una paginación completa?

Supongamos que queremos mostrar de 3 en 3, ordenados por nombre:

```
// Página 1
db.empleados.find({}).sort({ nombre: 1 }).skip(0).limit(3)

// Página 2
db.empleados.find({}).sort({ nombre: 1 }).skip(3).limit(3)

// Página 3
db.empleados.find({}).sort({ nombre: 1 }).skip(6).limit(3)
```

Cada página salta 3 resultados más (`skip`) y muestra 3 (`limit`).

¿Qué pasa si el orden es por un campo que no todos tienen?

MongoDB ordena igual, pero:

- Los documentos **sin ese campo** aparecen **al final si es ascendente**, o **al principio si es descendente**.
- No genera error. Solo que esos campos no influyen en el orden.

Conclusiones clave

- MongoDB **no usa texto como ASC o DESC**, sino números: `1` (ascendente) y `-1` (descendente).
- `.sort()` permite ordenar por uno o más campos, igual que SQL.
- `.skip()` y `.limit()` simulan `OFFSET` y `LIMIT` para **paginación manual**.
- El orden es clave para que la paginación tenga sentido y sea estable.
- Las operaciones se pueden **encadenar fácilmente** en una única consulta.