

BBDD NoSQL - MongoDB



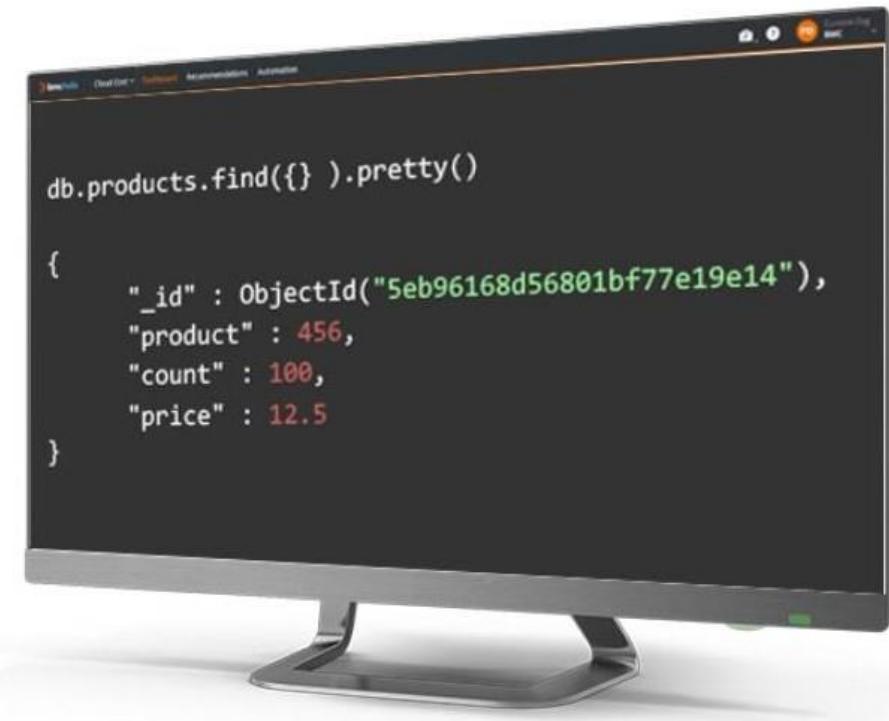
**CODE
SPACE**
ACADEMY

- Introducción a MongoDB
- Primeros pasos
- CRUD
- Tipos
- Condicionales
- MongoDB Compass

1. Introducción a MongoDB

MongoDB es una base de datos **JSON, NoSQL, big data**. Está disponible para ejecutarse en la nube o localmente, con versiones gratuitas y de pago.

Lanzado por primera vez en 2009, MongoDB resolvió un problema común a la mayoría de las empresas: cómo almacenar datos que varían en cada registro.



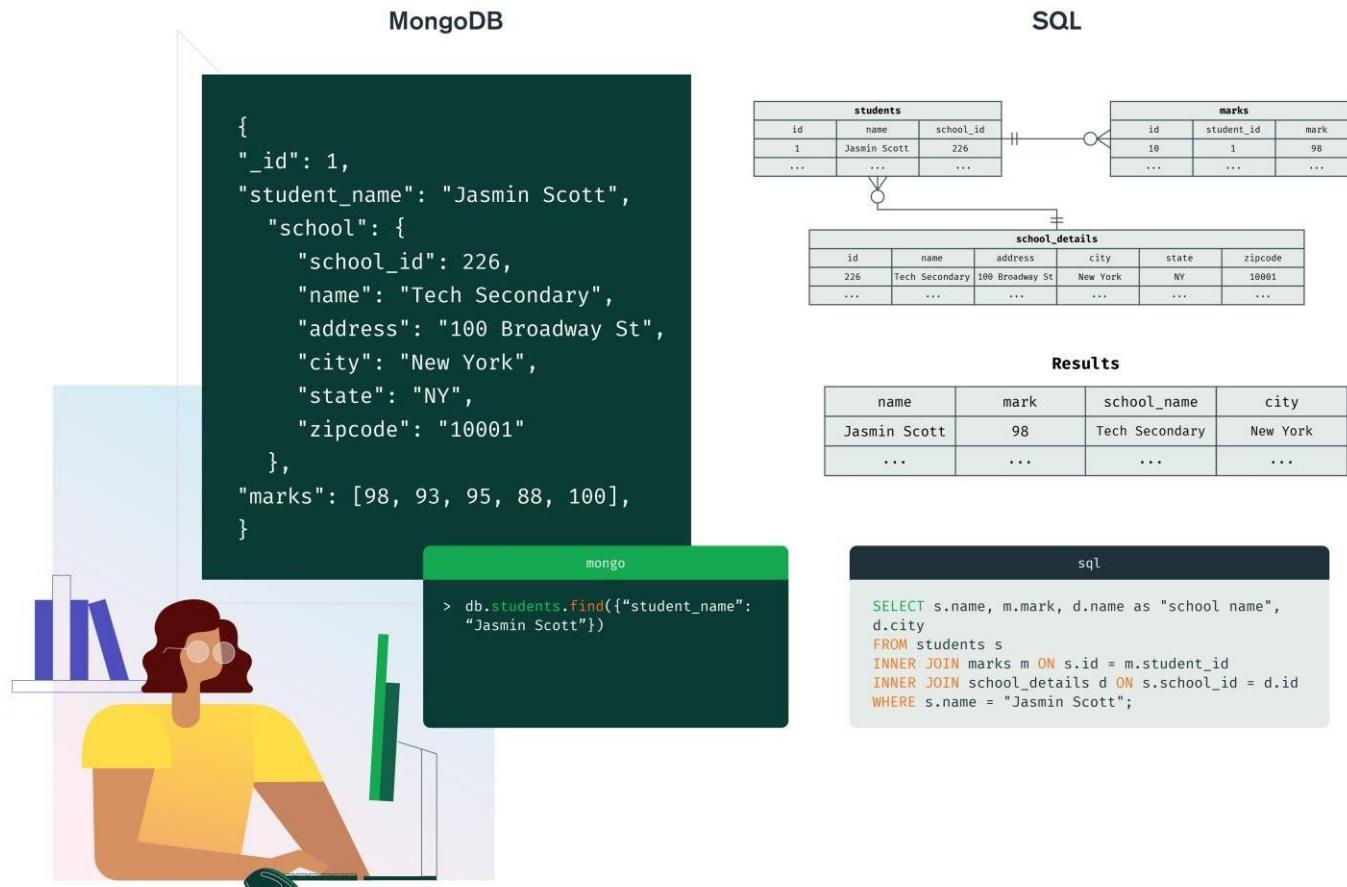
```
db.products.find({}).pretty()

{
    "_id" : ObjectId("5eb96168d56801bf77e19e14"),
    "product" : 456,
    "count" : 100,
    "price" : 12.5
}
```

1. Introducción a MongoDB

Actualmente se siguen utilizando bases de datos SQL porque son excelentes para sistemas de programación, inventario, ventas y gestión de pedidos.

Además, SQL es fácil de entender y su volumen de datos, incluso para los sistemas ERP más grandes, sigue siendo relativamente reducido.

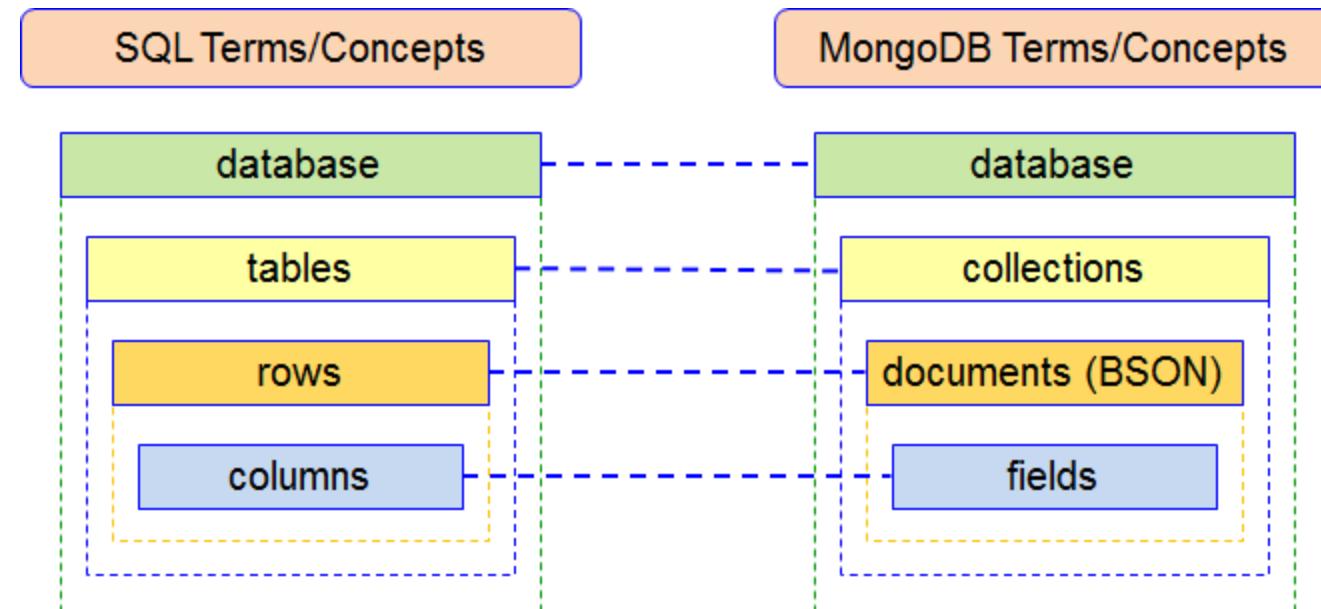


1. Introducción a MongoDB

Se puede trazar una analogía entre SQL y MongoDB.

Por lo general, un servidor MongoDB contendrá múltiples bases de datos, y cada base de datos es un contenedor físico para las colecciones.

A su vez, una colección es una agrupación de documentos, y cada documento representa un registro en una colección, por tanto, es la unidad básica de datos en MongoDB.



1. Introducción a MongoDB (documentos)

```
{  
  "_id":  
    ObjectId("5ad88534e3632e1a35a58d00"),  
  "name": {  
    "first": "John",  
    "last": "Doe" },  
  "address": [  
    { "location": "work",  
      "address": {  
        "street": "16 Hatfields",  
        "city": "London",  
        "postal_code": "SE1 8DJ"},  
      "geo": { "type": "Point", "coord": [  
        51.5065752,-0.109081]}},  
  ],
```

```
  "phone": [  
    { "location": "work",  
      "number": "+44-1234567890"},  
    ],  
  "dob": ISODate("1977-04-01T05:00:00Z"),  
  "retirement_fund":  
    NumberDecimal("1292815.75")  
}
```

Los documentos son análogos a los objetos **JSON**, se componen de pares de campos y valores, pero existen en la base de datos en un formato más rico en [tipos](#) conocido como **BSON**.

BSON es un híbrido de las palabras "binary" y "JSON".

Podemos pensar en BSON como una representación binaria de documentos JSON (JavaScript Object Notation).

Características clave de BSON:

- 1Eficiencia Binaria:** BSON es más compacto que JSON, lo que permite operaciones de lectura y escritura más rápidas.
- 2Soporte para Tipos de Datos Adicionales:** Incluye soporte para tipos como Date, Binary, y ObjectId, que no están disponibles en JSON.
- 3Ordenación:** BSON guarda los campos en el orden en que se almacenan, lo que facilita operaciones en MongoDB.
- 4Flexibilidad:** Permite estructuras anidadas y datos jerárquicos similares a JSON.

1 Introducción a MongoDB (documentos)

El valor de un campo puede ser cualquiera de los tipos de datos de BSON, incluyendo otros documentos, arrays y arrays de documentos.

Los nombres de los campos son strings y deben cumplir las siguientes restricciones:

- ***_id*** debe ser la clave primaria, único en la colección, inmutable, y de cualquier tipo que no sea un array, regex, or undefined. Si el ***_id*** contiene subcampos, los nombres de los subcampos no pueden comenzar con el símbolo (\$).
- Los nombres de campos no pueden contener el carácter null (representado con la secuencia de escape \0 en el código fuente).
- Se pueden almacenar nombres de campos que incluyan el punto (.)

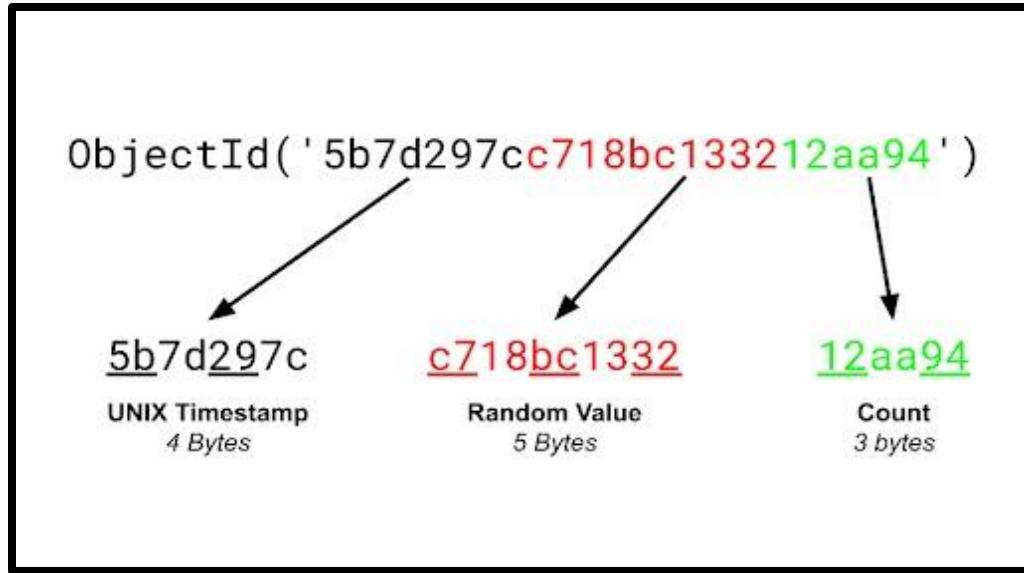
1. Introducción a MongoDB (documentos)

MongoDB usa la notación de punto para acceder a los elementos de un array o a los campos de un documento anidado.

En un documento anidado, concatenamos el nombre del documento con el punto (.) y el nombre del campo entre comillas:

```
{  
  name: { first: "Alan", last: "Turing"  
  }, contact: { phone: { type: "cell",  
    number:  
    "111-222-3333" } },  
}  
> "contact.phone.number" #111-222-  
3333
```

Por lo general, habitualmente permitiremos que mongod (el proceso daemon principal del sistema MongoDB) añada el campo _id y genere el ObjectId.



También podemos generar el id nosotros mismos mediante: un [ObjectId\(\)](#), designar un valor que sea único, generar un número [autoincremental](#), o generar un [UUID](#).

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de creación (insert)

En MongoDB, las operaciones de inserción tienen como objetivo una única colección. Además, todas las operaciones de escritura en MongoDB son atómicas, es decir, a nivel de un único documento. Desde la versión 3.2, están disponibles los métodos, para insertar 1 o varios documentos respectivamente:

```
db.collection.insertOne()
```

```
db.collection.insertMany()
```

En versiones anteriores db.collection.insert() hacía ambas cosas.

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de creación (insert)

Veamos algunos ejemplos:

```
db.products.insertOne( { item: "card", qty: 15 } );
```

```
db.products.insertMany( [  
    { item: "tape", qty: 15 },  
    { item: "envelope", qty: 20 },  
    { item: "stamps" , qty: 30 }  
] );
```

```
db.users.insertMany([  
    {"_id":ObjectId(), "Name": "object id"},  
    {"_id":10, "Name": "explicit id"}  
])
```

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de lectura (read)

Las operaciones de lectura recuperan documentos de una colección, mediante el método `find()`:

```
db.collection.find()
```

Para seleccionar todos los documentos de la colección, pasamos un documento vacío como parámetro de filtro de consulta al método `find()`:

```
db.users.find({})
db.bios.find(
  { contribs: 'OOP' },
  { 'name.first': 0, birth: 0 }
)
```

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de lectura (read)

El método `find()` puede recibir 2 parámetros opcionales:

- `query` → hace de filtro para la consulta (sería el equivalente a `where`)
- `Projection` → especifica los campos que deberán mostrarse (igual que en `select` de SQL)

[Operadores de consulta](#)
[Parámetro de proyección](#)

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de actualización (update)

Las operaciones de actualización modifican los documentos existentes en una colección. MongoDB proporciona los siguientes métodos desde la versión 3.2:

`db.collection.updateOne(filter, update, options)` → Actualiza como máximo un único documento (el primero) que coincide con un filtro especificado, aunque pudieran coincidir varios.

`db.collection.updateMany(filter, update, options)` → Actualiza todos los documentos que coinciden con el filtro especificado.

`db.collection.replaceOne(filter, replacement, options)` → Reemplaza como máximo un único documento (el primero) que coincide con un filtro especificado, aunque pudieran coincidir varios.

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de actualización (update)

```
db.restaurant.updateOne(  
  { "name" : "Central Perk Cafe" },  
  { $set: { "coffe" : 3 } }  
)
```

```
db.restaurant.updateMany(  
  { capacity: { $gt: 200 } },  
  { $set: { "Review" : true } }  
)
```

```
db.restaurant.replaceOne(  
  { "name" : "Central Perk Cafe" },  
  { "name" : "Central Pork Cafe", "Borough" : "Manhattan" }  
)
```

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de borrado (delete)

Las operaciones de eliminación borran los documentos existentes en una colección. MongoDB proporciona los siguientes métodos desde la versión 3.2:

`b.collection.deleteMany()` → Elimina todos los documentos que coinciden con el filtro.

`b.collection.deleteOne()` → Elimina un único documento de una colección.

Manejo de colecciones y documentos - CRUD (Create, Read, Update, Delete)

Operaciones de borrado (delete)

```
db.orders.deleteMany( { "client" : "Crude Traders Inc." } )
```

```
db.orders.deleteOne( { "_id" : ObjectId("563237a41a4d68582c2509da") } )
```

```
{  
  "_id": ObjectId("64bd06a4f45bc4d5e75e8a76"), // Tipo ObjectId  
  "nombre": "Iván Moreno", // String  
  "edad": 30, // Number  
  "direccion": { // Object  
    "calle": "Calle Luna",  
    "ciudad": "Madrid",  
    "codigoPostal": 28001  
  },  
  "calificaciones": [90, 88, 95], //Array  
  "esEstudiante": false, // Boolean  
  "fechaRegistro": ISODate("2023-12-16T10:00:00Z"), // Date  
  "intereses": ["leer", "viajar", "cocinar"], // Array de Strings  
  "ultimaActualizacion": Timestamp(1692197245, 1) // Timestamp  
}
```

Operadores de comparación

\$gt : Greater than (Mayor que)

```
db.collection.find({ "edad": { "$gt": 25 } })
```

\$lt: Less than(Menor que)

```
db.collection.find({ "edad": { "$lt": 18 } })
```

\$eq :Equal to(Igual a)

```
db.collection.find({ "estado": { "$eq": "activo" } })
```

\$ne : No equal to (Diferente a)

```
db.collection.find({ "estado": { "$ne": "inactivo" } })
```

\$gte : Greater tan or equal to (Mayor o igual que)

```
db.collection.find({ "edad": { "$gte": 18 } })
```

\$lte : Less tan or equal to (Menor o igual que)

```
db.collection.find({ "edad": { "$lte": 12 } })
```

Condicionales lógicos

- \$and:** Retorna documentos que cumplen con todas las condiciones especificadas.

Ejemplo: `db.collection.find({ "$and": [{ "edad": { "$gt": 18 } }, { "ciudad": "Madrid" }] })`

Devuelve documentos donde edad sea mayor a 18 y ciudad sea "Madrid".

- \$or:** Retorna documentos que cumplen con al menos una de las condiciones especificadas

Ejemplo: `db.collection.find({ "$or": [{ "edad": { "$lt": 18 } }, { "ciudad": "Madrid" }] })`

Devuelve documentos donde edad sea menor a 18 o ciudad sea "Madrid".

- \$nor:** Retorna documentos que no cumplen con ninguna de las condiciones especificadas.

Ejemplo: `db.collection.find({ "$nor": [{ "edad": { "$gt": 18 } }, { "ciudad": "Madrid" }] })`

Devuelve documentos donde edad no sea mayor a 18 y ciudad no sea "Madrid".

- \$not:** Filtra documentos donde una condición no se cumple

Ejemplo: `db.collection.find({ "edad": { "$not": { "$gte": 18 } } })`

Devuelve documentos donde edad no sea mayor o igual a 18.

Operadores de array

- **\$all** : Selecciona documentos donde un campo contiene todos los valores especificados.

Ejemplo: `db.collection.find({ "intereses": { "$all": ["música", "deportes"] } })`

Devuelve documentos donde el campo intereses contiene "música" y "deportes".

- **\$elemMatch** : Retorna documentos donde al menos un elemento en un array cumple con múltiples condiciones

Ejemplo: `db.collection.find({ "notas": { "$elemMatch": { "$gte": 90, "$lt": 100 } } })`

Devuelve documentos donde al menos un elemento en el array notas está entre 90 y 99.

- **\$size** : Retorna documentos que no cumplen con ninguna de las condiciones especificadas.

Ejemplo: `db.collection.find({ "amigos": { "$size": 3 } })`

Devuelve documentos donde el array amigos tiene exactamente 3 elementos.

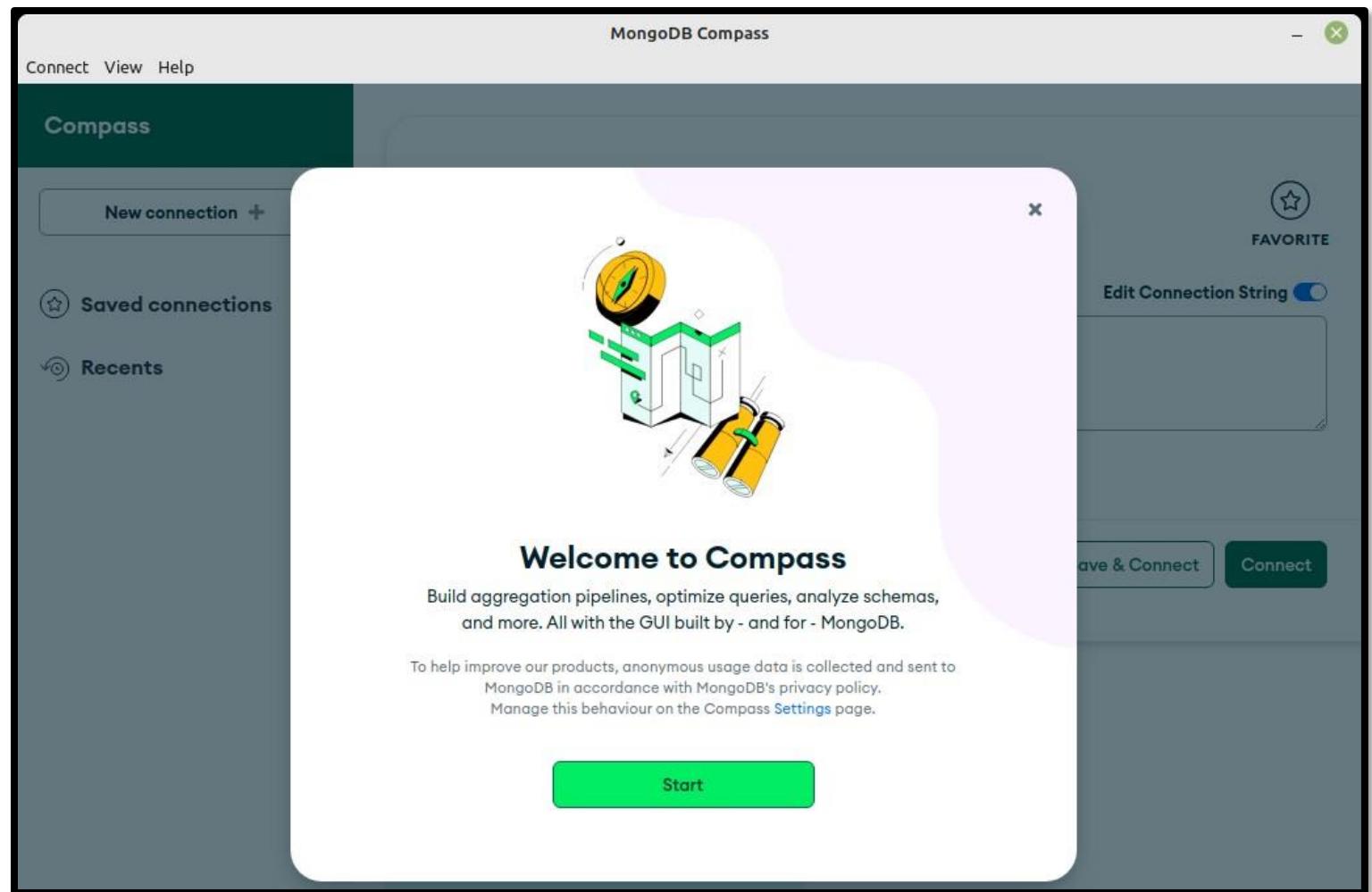
3. MongoDB Compass

Para trabajar en modo interactivo se pueden utilizar diversos clientes GUI.

El caso de MongoDB Compass es muy interesante y similar a clientes MySQL.

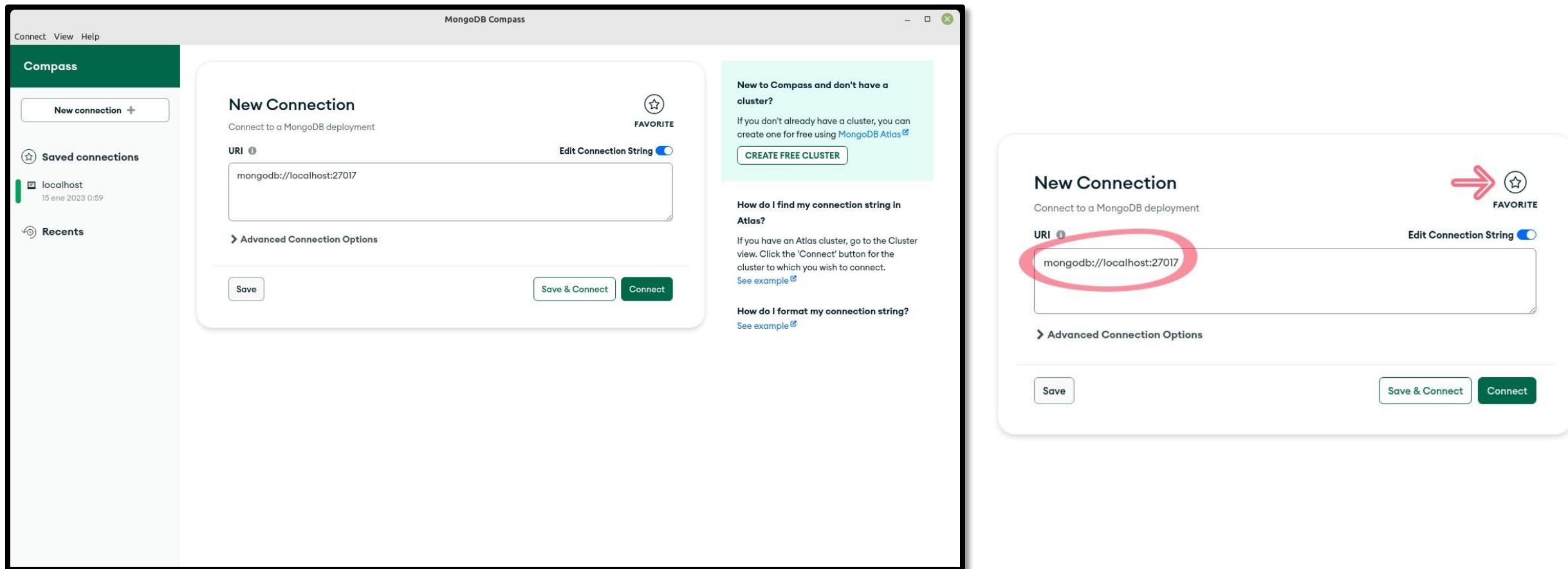
Descarga MongoDB Compass desde la siguiente URL:

<https://www.mongodb.com/products/compass>



3. MongoDB Compass

A continuación, nos pedirá la cadena de conexión.



The image shows two screenshots of the MongoDB Compass application. Both screenshots display the 'New Connection' dialog. In the left screenshot, the 'URI' field contains the value 'mongodb://localhost:27017'. In the right screenshot, the same 'URI' field is highlighted with a red oval, and a red arrow points to it from the left. The 'Save & Connect' button is visible at the bottom of both dialogs.

3. MongoDB Compass

En el cuadro de texto añadimos:

```
mongodb://localhost:27017
```

Componentes de la cadena de conexión

Ej: `mongodb://myDBReader:D1fficultP%40ssw0rd@mongodb0.example.com:27017/customers?connectTimeoutMS=300000&authSource=Admin`

- **mongodb://** → Un prefijo obligatorio para identificar que se trata de una cadena en el formato de conexión estándar.
- **username:password@** → Opcional. Credenciales de autenticación. Si se especifican, el cliente intentará autenticar al usuario en la base de datos asociada a las credenciales del usuario. Si no se especifica, el cliente intentará autenticar al usuario en la base de datos de autenticación por defecto. Y si tampoco se especifica, a la base de datos del administrador.

3 MongoDB Compass

- **host[:port]** → El host (y el número de puerto opcional) donde se ejecuta la instancia de mongod. Se puede especificar un nombre de host, una dirección IP o un socket de dominio UNIX.
- **/defaultauthdb** → Opcional. La base de datos de autenticación que se utilizará si la cadena de conexión incluye las credenciales de autenticación `username:password@`
- **?<options>** → Opcional. Una cadena de consulta que especifica las opciones específicas de la conexión como pares `<nombre>=<valor>`.

En caso de dudas, en la documentación oficial tenemos decenas de [ejemplos](#) de cadenas de conexión.

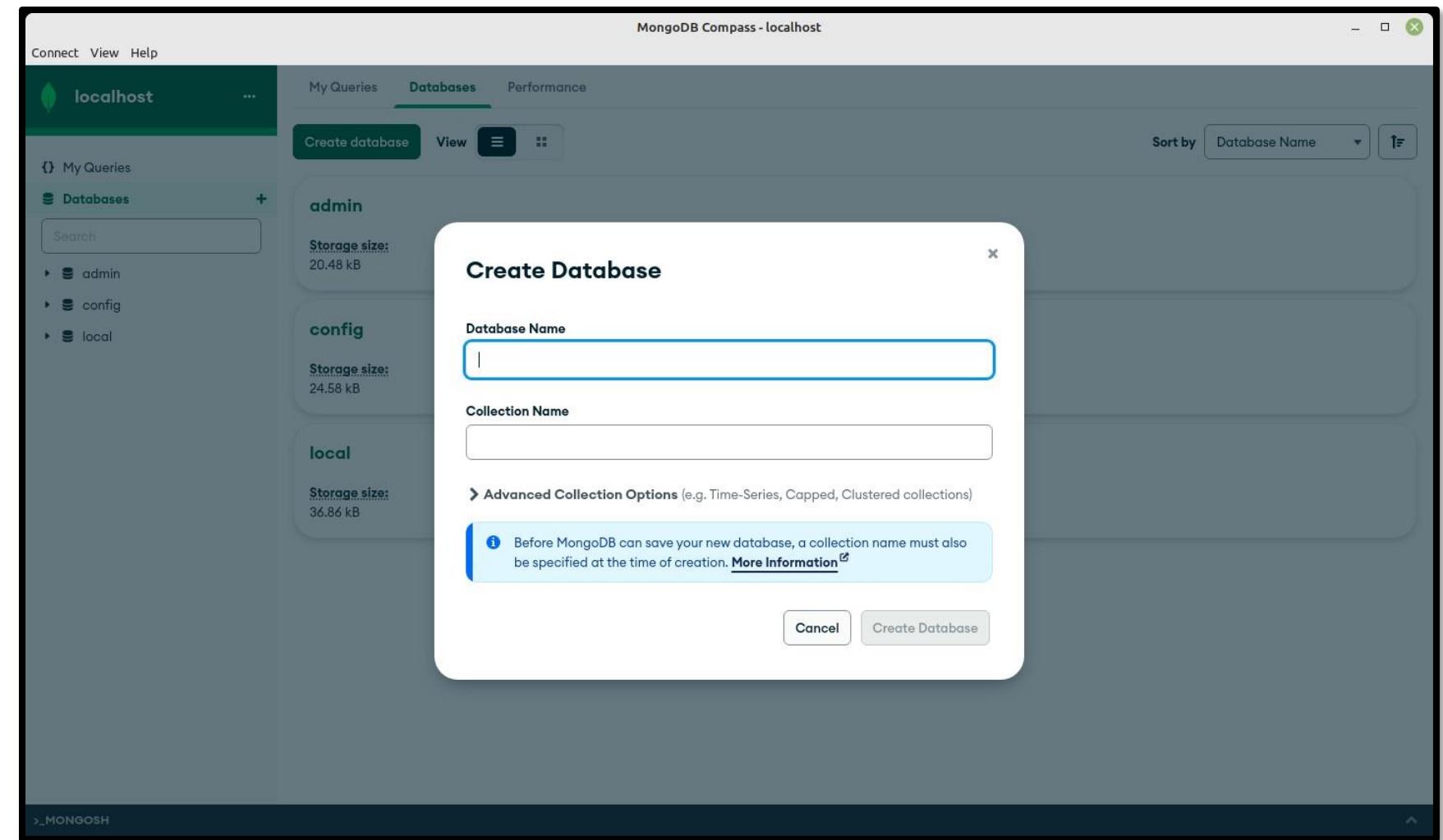
3. MongoDB Compass

Gestionar MongoDB desde Compass

Una vez hemos conectado con el servidor, ya podemos crear la primera base de datos.

Justo debajo de la pestaña **Databases**, tenemos el botón “**CREATE DATABASE**”.

Nos pedirá al crear la base de datos el nombre de la primera colección (obligatorio).



4 Shell MongoDB

La shell mongo es una interfaz Javascript interactiva para MongoDB. Se puede utilizar tanto para consultar y actualizar datos como para operaciones administrativas.

Manejo de la base de datos

- **db.getMongo()**: Devuelve el objeto de conexión para la conexión actual.
- **show databases / show dbs**: Muestra las bases de datos existentes
- **db / db.getName()**: Muestra la base de datos actual (por defecto test).
- **use <db name>**: Selecciona una base de datos y si no existe, la crea (si añadimos una colección).
- **show users / db.getUsers()**: muestra los usuarios existentes en una base de datos.

4 Shell MongoDB

- **show collections / db.getCollectionNames()**: Muestra todas las colecciones de la bases de datos seleccionada.
- **show roles**: muestra los roles existentes en una base de datos.
- **show users / db.getUsers()**: muestra los usuarios existentes en una base de datos.
- **db.dropDatabase()**: Elimina la base de datos seleccionada.
- **db.help()**: muestra información acerca de los métodos del objeto db.

Operadores de actualización

\$set: Se usa para cambiar el valor de un campo. En caso de que este no exista, lo crea. Sirve en campos simples y anidados.

Ejemplo:

```
db.coleccion.updateOne({ filtro }, { $set: { campo: valorNuevo } });
```

\$unset: Se usa para cambiar borrar un campo del documento. No pone a **null**, sino que lo elimina directamente del documento.

Ejemplo:

```
db.coleccion.updateOne({ filtro }, { $unset: { campo: '' } });
```

\$inc: Se usa para sumar o restar una cantidad a un campo numérico. Si no existe, lo crea con ese valor. Si no es numérico, dà error.

Ejemplo:

```
db.coleccion.updateOne({ filtro }, { $inc: { campoNumerico: cantidad } });
```

\$push: Añade un nuevo elemento al final de un array. Permite añadir 1 o más valores con **\$each**.

Ejemplos:

```
db.coleccion.updateOne({ filtro }, { $push: { array: valor } });
```

```
db.coleccion.updateOne({ filtro }, { $push: { array: { $each: ["value1", "value2"] } } });
```

Operadores de actualización

\$addToSet: Añade un nuevo elemento a una rray solo si no existe ya. Es un push inteligente que evita duplicados.

Ejemplo: `db.coleccion.updateOne({ filtro }, { $addToSet: { array: valor } });`

\$pull: Elimina del array todos los elementos que coincidan con una condición.

Ejemplo: `db.coleccion.updateOne({ filtro }, { $pull: { array: condicion } });`

\$pop: Se usa para eliminar un solo elemento del array. Con 1 elimina el último y con -1 el primero

Ejemplo: `db.coleccion.updateOne({ filtro }, { $pop: { array: 1 } });`

`db.coleccion.updateOne({ filtro }, { $pop: { array: -1 } });`

Índices

Los índices optimizan y aceleran el acceso y lectura de los datos en las colecciones de MongoDB. Cuando intuyamos que se va a buscar mucho un campo, es buena práctica crear un índice de él.

Crear un índice: `db.Usuarios.createIndex({ email: 1 });`

1: Orden ascendente **-1:** Orden descendente

Crear un índice compuesto: `db.Usuarios.createIndex({ ciudad: 1, edad: -1 })`

Ver los índices: `db.Usuarios.getIndexes()`

Borrar un índice: `db.Usuarios.dropIndex("email_1")`

Regex

Se usa para buscar parecidos en los documentos.

Ejemplos:

Valores que empiezan por ["**xxxx**"]:

`db.moviles.find({ modelo: { $regex: '^Find' } });`

Valores que terminan por ["**xxx\$**"]:

`db.moviles.find({'especificaciones.version': { $regex: '2.0$' } });`

Función aggregate

La función aggregate transforma los datos paso a paso a través de etapas, donde se realizan operaciones.

Stage 1:

\$match: Se usa para filtrar documentos. Es como un find dentro del aggregate

Ejemplo: `db.Usuarios.aggregate([{$match: { activo: true }}]);`

Stage 2:

\$project: Se usa para seleccionar que campos se van a mostrar y permite crear nuevos campos.

Ejemplo: `db.Usuarios.aggregate([
 {
 $project: {
 _id: 0,
 nombre: 1,
 edad: 1,
 mayorDeEdad: { $gte: ['$edad', 18] },
 },
 },
]);`

Función aggregate

Stage 3:

\$unwind: Se usa para deshacer arrays, convirtiendo cada elemento del array en un documento separado. Util para cuando debemos distinguir o buscar por separado esos valores.

Ejemplo:

```
db.Usuarios.aggregate([{$unwind: '$intereses'}]);
```

Stage 4:

\$group: Se usa para agrupar documentos. ES parecido al group by SQL.

Se suele usar con operadores típicos como: **\$sum, \$avg, \$max, \$min, \$first, \$last**

Ejemplo:

```
db.Usuarios.aggregate([
  {
    $group: {
      _id: '$direccion.ciudad',
      totalUsuarios: { $sum: 1 },
    },
  },
]);
```

Función aggregate

Stage 5:

\$sort: Se usa para ordenar los resultados.

Ejemplo:

```
db.Usuarios.aggregate([
  {
    $project: {
      _id: 0,
      nombre: 1,
      edad: 1,
      mayorDeEdad: { $gte: ['$edad', 18] },
    },
  },
  {
    $sort: { edad: -1 },
  },
]);
```

1: Orden ascendente **-1:** Orden descendente

Función aggregate

Stage 6:

\$limit: Se usa para limitar la cantidad de resultados.

Ejemplo:

```
db.Usuarios.aggregate([
  {
    $project: {
      _id: 0,
      nombre: 1,
      edad: 1,
      mayorDeEdad: { $gte: ['$edad', 18] },
    },
  },
  { $limit: 5 },
]);
```

EXTRA:

\$cond: Se usa para establecer una condición dentro del \$project.
Funciona como un if... else

```
$project: {
  _id: 0,
  categoria: {
    $cond: {
      if: { $gt: ["$media", 800 ]},
      then: "Caro",
      else: "Barato"
    }
  }
}
```

Función aggregate

EXTRA:

Usar variables de **\$group** luego en **\$project**.

A veces, cuando usamos el valor de un campo como variable de agrupación `$_id` del `$group`, podemos posteriormente rescatar esa variable en el `$project` y hacer que sea visible.

```
{  
  $group: {  
    _id: "$marca",  
    media: { $avg: "$precio"}  
  },  
  {  
    $project: {  
      _id: 0,  
      media: 1,  
      marca: "$_id"  
    }  
  },
```

BBDD NoSQL - MongoDB



**CODE
SPACE**
ACADEMY