

Base de datos



Comparamos SQL y NoSql

...



- ▶ Las bases de datos **NoSQL** son estructuras de almacenamiento de información que **no** cumplen con el esquema **entidad-relación**.
- ▶ Tampoco utilizan una estructura de datos en forma de **tablas**.
- ▶ No utilizan el **lenguaje SQL** como lenguaje de consultas
- ▶ No realizan operaciones de **JOIN**
- ▶ Tienen una arquitectura **distribuida**.

SQL	NoSQL
SQL	Json de JavaScript
Tablas	Colecciones de documentos
Relaciones	No relacionales
Esquema Fijo	Esquema Libre
Centralizadas en una única máquina	Compartida en varias máquinas

Clasificación de las base de datos noSql



Valor-Clave



Documentos



Columnas Anchas



Grafos

Clasificación de las base de datos noSql

CLAVE- VALOR

Una base de datos clave-valor es un tipo de base de datos no relacional que utiliza un método simple de clave-valor para almacenar datos. Almacena datos como un conjunto de pares clave valor, donde una clave sirve como un identificador único.

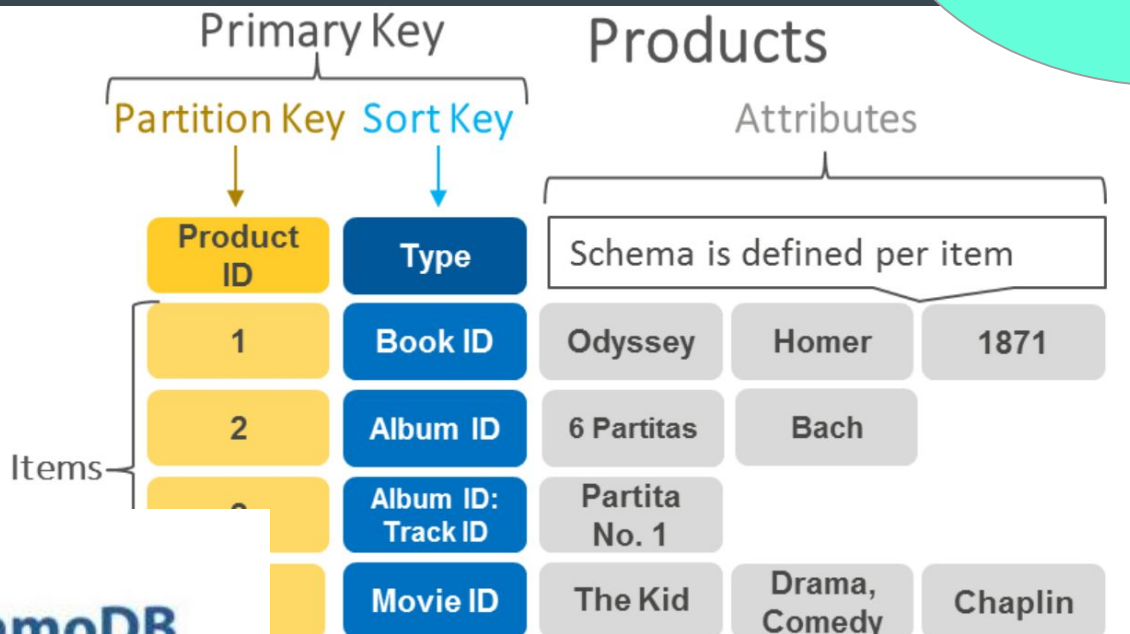
Clave: es un identificador único que puede ser de cualquier tipo.

Valor: puede ser texto, número, código php, imágenes, listas, etc

El contenido de la base de datos puede ser muy heterogéneo, por lo que es posible incluir objetos distintos en una misma columna. Lo mismo se aplica a los rasgos identificativos.

Clasificación de las base de datos noSql

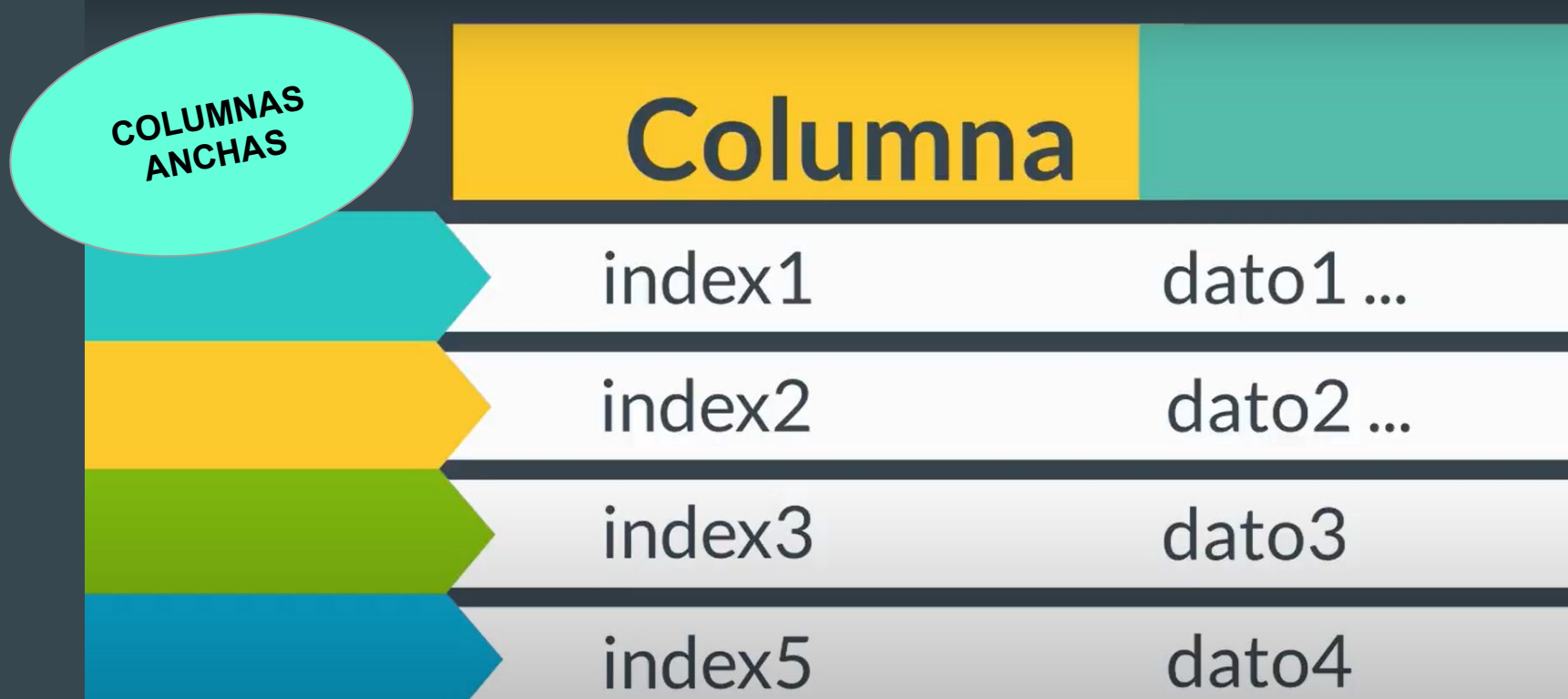
CLAVE- VALOR



DynamoDB
Core Components



Clasificación de las base de datos noSql



Columna	
index1	dato1 ...
index2	dato2 ...
index3	dato3
index5	dato4

Parecidos a los relacionales pero las columnas pueden variar de fila a fila en cada tabla y pueden contener lista de valores. Incorporan características multidimensionales

Clasificación de las base de datos noSql

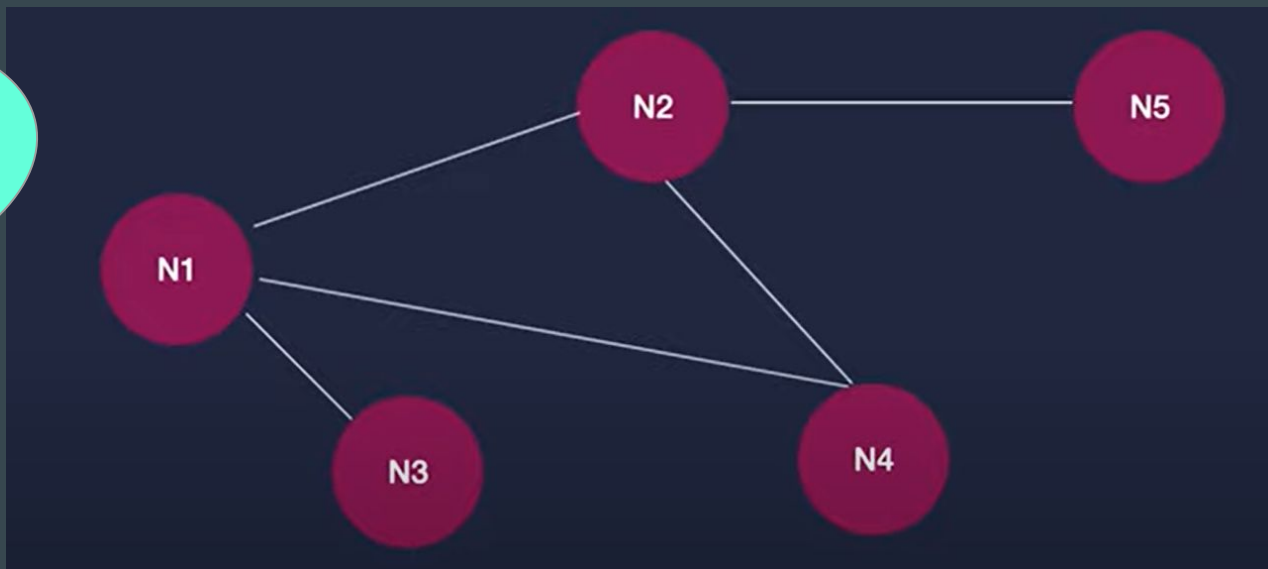
**COLUMNAS
ANCHAS**



cassandra

id	user	job
1	Name: Marcos Age: 28 City: NY	Title: CSAM Start_date: 2015
2	Name: Jaime Age: 34 City: London Married: false	Title: Doctor Start_date: 2012 Location: Paris

Clasificación de las base de datos noSql

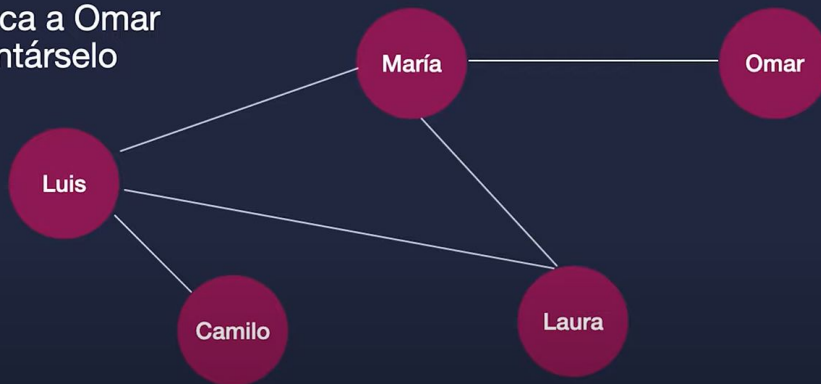


Las bases de datos de grafos también nos permiten aplicar la teoría de grafos a nuestros datos de manera eficiente, lo que nos permite descubrir conexiones de nuestros datos que de otra manera son difíciles de ver. Por ejemplo, rutas mínimas entre nodos o conjuntos disjuntos dentro de nuestros datos.

Diseñadas para almacenar y navegar a través de relaciones.

Clasificación de las base de datos noSql

Para que Laura conozca a Omar
María tiene que presentárselo



GRAFOS



Clasificación de las base de datos noSql

DOCUMENTOS

***LAS BASES DE DATOS DE
DOCUMENTOS SUELEN ALMACENAR
DOCUMENTOS JSON, XML Y BSON.***

Base de datos ---> Colección

DOCUMENTOS → JSON DE JAVA SCRIPT



JavaScript Object Notation

Es un formato ligero de intercambio de datos, que resulta sencillo de leer y escribir para los programadores, y simple de interpretar y generar para las máquinas

Base de datos ---> Colección

DOCUMENTOS → JSON DE JAVA SCRIPT



Es un estándar basado en texto plano para el intercambio de datos, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas

Base de datos ---> Colección

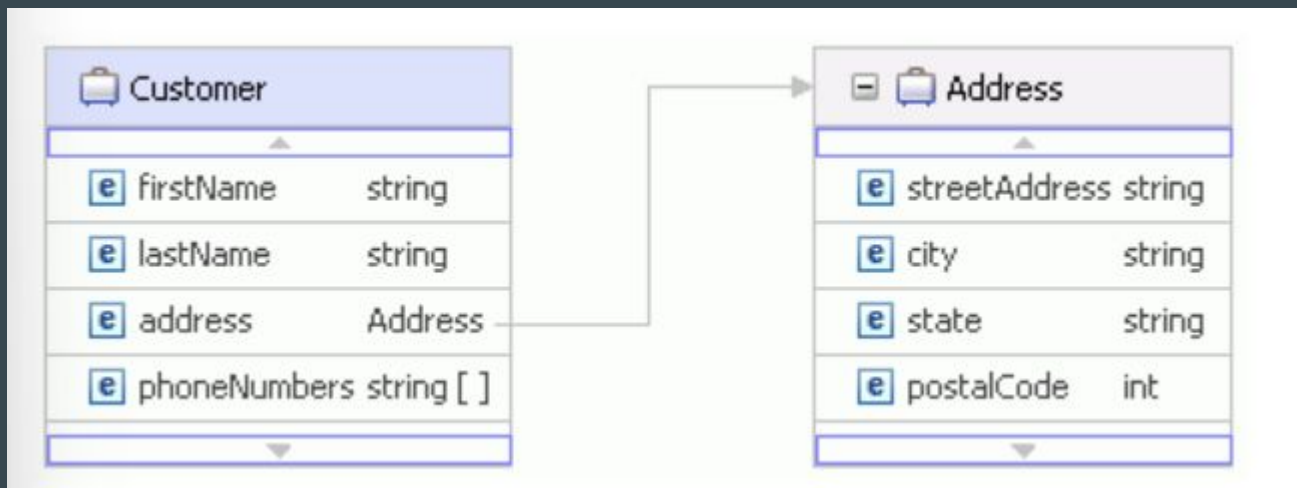
DOCUMENTOS → JSON DE JAVA SCRIPT

Una de las características de JSON, al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por este método no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor Python, pues cada uno tiene su propia librería para codificar y decodificar cadenas en este formato.

Base de datos ---> Colección

EJ. DE JSON (CARDINALIDAD UNICA)

Dado el siguiente objeto de negocio:



Base de datos ---> Colección

EJ. DE JSON (CARDINALIDAD UNICA)

Y dado los valores de las siguientes propiedades del objeto de negocio

Objeto de negocio	Propiedad	Valor
<div>Customer</div> <div>Address</div>	<div>firstName</div> <div>lastName</div> <div>streetAddress</div> <div>city</div> <div>state</div> <div>postalCode</div> <div>phoneNumbers[0]</div> <div>phoneNumbers[1]</div>	<div>John</div> <div>Smith</div> <div>21 2nd Street</div> <div>New York</div> <div>NY</div> <div>10121</div> <div>212-732-1234</div> <div>646-123-4567</div>

Base de datos ---> Colección

EJ. DE JSON (CARDINALIDAD ÚNICA)

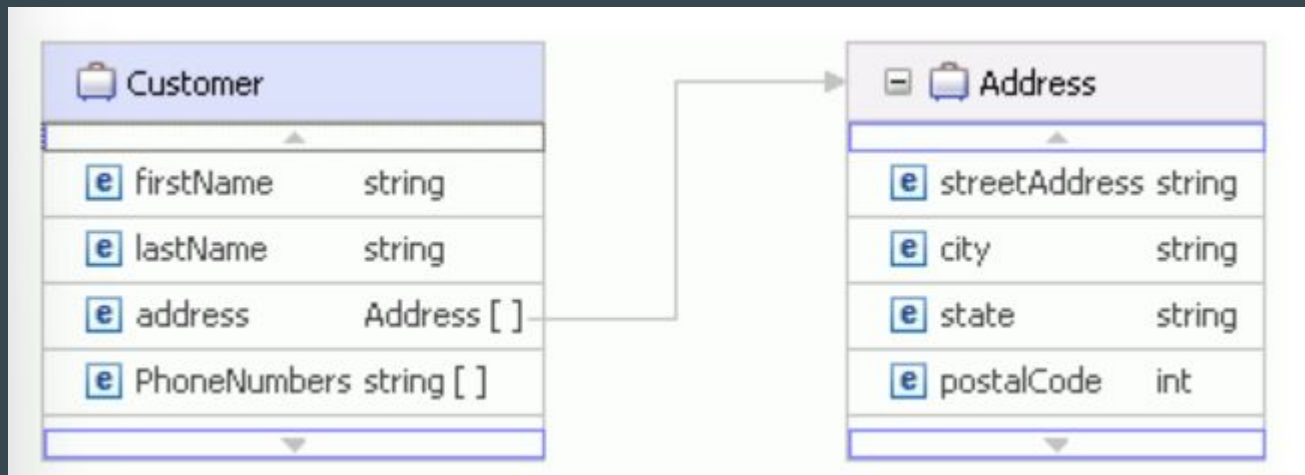
El formato Json sería

```
1      {
2          "firstName": "John",
3          "lastName": "Smith",
4          "address": {
5              "streetAddress": "21 2nd Street",
6              "city": "New York",
7              "state": "NY",
8              "postalCode": 10021
9          },
10         "phoneNumbers": [
11             "212-732-1234",
12             "646-123-4567"
13         ]
14     }
```

Base de datos ---> Colección

EJ. DE JSON (CARDINALIDAD MÚLTIPLE)

Dado el siguiente objeto de negocio:



Base de datos ---> Colección

EJ. DE JSON (CARDINALIDAD MÚLTIPLE)

Dado el
siguiente
objeto de
negocio:

Objeto de negocio	Propiedad	Valor
<pre>Customer Address[0] Address[1]</pre>	<pre>firstName lastName streetAddress city state postalCode streetAddress city state postalCode phoneNumbers[0] phoneNumbers[1]</pre>	<pre>John Smith 21 2nd Street New York NY 10121 577 Airport Blvd Burlingame CA 94010 212-732-1234 646-123-4567</pre>

Base de datos ---> Colección

EJ. DE JSON
(CARDINALIDAD
MÚLTIPLE)

El formato Json sería

```
15      {
16          "firstName": "John",
17          "lastName": "Smith",
18          "address": [{
19              "streetAddress": "21 2nd Street",
20              "city": "New York",
21              "state": "NY",
22              "postalCode": 10021
23          }, {
24              "streetAddress": "577 Airport Blvd",
25              "city": "Burlingame",
26              "state": "CA",
27              "postalCode": 94010
28          }],
29          "phoneNumbers": [
30              "212-732-1234",
31              "646-123-4567"
32          ]
33      }
```

Esquema tabla SQL

	COLUMNA	COLUMNA
FILA	REGISTRO	REGISTRO
FILA	REGISTRO	REGISTRO

Base de datos ---> COLECCIÓN DE DOCUMENTOS

	CAMPO	CAMPO
DOCUMENTO	DATOS DOC.	DATOS DOC.
DOCUMENTO	DATOS DOC.	DATOS DOC.

VENTAJAS

- **Rendimiento.** Las bases de datos NoSQL ofrecen un mayor rendimiento que las SQL (necesitan menos recursos de hardware).
- **Fiabilidad.** Las bases de datos relacionales SQL son más fiables que las NoSQL (si un proceso tiene algún error, no se lleva a cabo).
- **Disponibilidad.** En cuanto a la disponibilidad, ambas alternativas son igual de válidas.
- **Consistencia.** La consistencia de las bases de datos NoSQL es pobre, en cambio las SQL son bases de datos muy consistentes (la consistencia es la capacidad de garantizar la integridad de los datos).
- **Almacenamiento.** Las bases de datos SQL son indicadas cuando la cantidad de datos no son extremadamente grandes, mientras que las NoSQL son ideales para manejar grandes volúmenes de datos.
- **Escalabilidad.** Las bases de datos NoSQL son escalables por lo que se pueden aumentar su capacidad fácilmente, sin embargo, las SQL pueden ser escalables, pero con un costo económico más elevado. Las bases de datos No SQL utilizan un escalado horizontal (aumentar el número de servidores) mientras que las SQL utilizan un escalado vertical (aumentar los recursos de un servidor).

CUANDO USAR SQL

Cuando en un proyecto **el volumen de los datos no tendrá un gran crecimiento**, o este se realice de forma lenta, las bases de datos SQL.

En proyectos donde el **pico de usuarios que accedan a la base de datos esté previsto**, las bases de datos relacionales funcionan de manera óptima.

Si las **necesidades de procesamiento** de la base de datos requieren de un **único servidor**, se pueden utilizar bases de datos SQL.

CUANDO USAR NOSQL

Si el **crecimiento de la base de datos se realiza de forma rápida**, con grandes aumentos en poco tiempo, lo ideal es recurrir a bases de datos no relacionales NoSQL.

Si el acceso a la base de datos puede sufrir **picos altos y en múltiples ocasiones**, lo mejor es optar por No SQL.

Si las necesidades de procesamiento no se pueden prevenir, es mejor utilizar bases de datos NoSQL escalables (permiten expandirse).

MongoDB





SISTEMA DE BASE DE DATOS

NO-RELACIONAL (NOSQL)



MongoDB es una base de datos de código abierto.

Utiliza un modelo de base de datos basada en documentos



Proporciona la flexibilidad necesaria para acomodar estructuras de datos variables y en evolución, a diferencia del modelo más rígido visto en SQL



mongoDB

ORIENTADO A DOCUMENTOS

Se representa con...



Se almacena como...



*Es una representación binaria
de Json*

```

{"hello": "world"} → \x16\x00\x00\x00 // total document size
                     \x02 // 0x02 = type String
                     hello\x00 // field name
                     \x06\x00\x00\x00world\x00 // field value
                     \x00 // 0x00 = type E00 ('end of object')

```

```

{"BSON": ["awesome", 5.05, 1986]} → \x31\x00\x00\x00
                                   \x04BSON\x00
                                   \x26\x00\x00\x00
                                   \x02\x30\x00\x08\x00\x00\x00awesome\x00
                                   \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
                                   \x10\x32\x00\xc2\x07\x00\x00
                                   \x00
                                   \x00

```



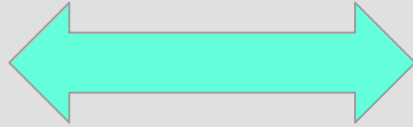
Se almacena como...



BSON { 01010100
11101001
10101110
01010100 }

Es una representación binaria
de Json

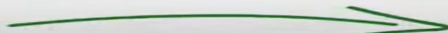
MongoDB utiliza BSON en lugar de JSON para almacenar datos debido a su rendimiento y eficiencia mejorados.



Aunque MongoDB utiliza BSON internamente, proporciona una interfaz JSON para interactuar con los datos almacenados

 **mongoDB**
ORIENTADO A DOCUMENTOS

Se almacena como...



Es una representación binaria de Json

BSON { }

Instalamos MongoDB

...

Instalar MongoDB

MongoDB Community Server

The Community version of our distributed database offers a flexible document data model along with support for ad-hoc queries, secondary indexing, and real-time aggregations to provide powerful ways to access and analyze your data.

Available Downloads

Version
5.0.12

Platform
Windows

Package
msi

1 - Bajamos MongoDB de la pagina MongoDB.com

<https://www.mongodb.com/try/download/community>

2 - Instalamos MongoDB (Seleccionar Complete)

3- Por defecto se instala MongoDBCompas (software para gestionar la bd)

4 - Se recomienda instalar studio 3T (antes robo 3t)

<https://robomongo.org/>

Download the latest
version of Robo 3T

A free 30-day trial of the full access edition of Studio 3T is included with Robo 3T.

Download Robo 3T + Studio 3T

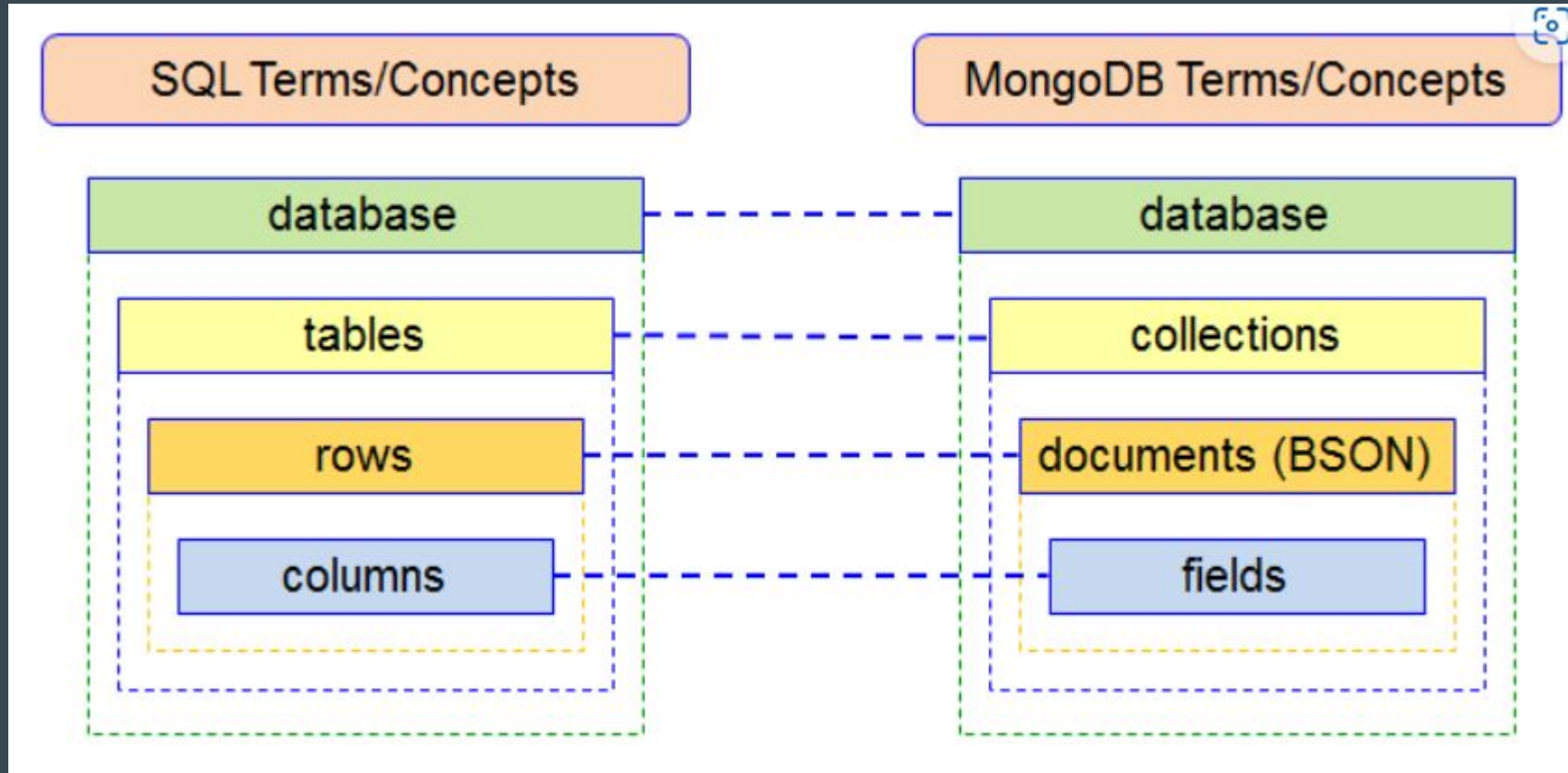
Download Studio 3T Only

Studio 3t



Studio 3T proporciona una **interfaz gráfica de usuario (GUI)** y un **entorno de desarrollo integrado (IDE)** para acceder y modificar las bases de datos MongoDB y sus documentos.

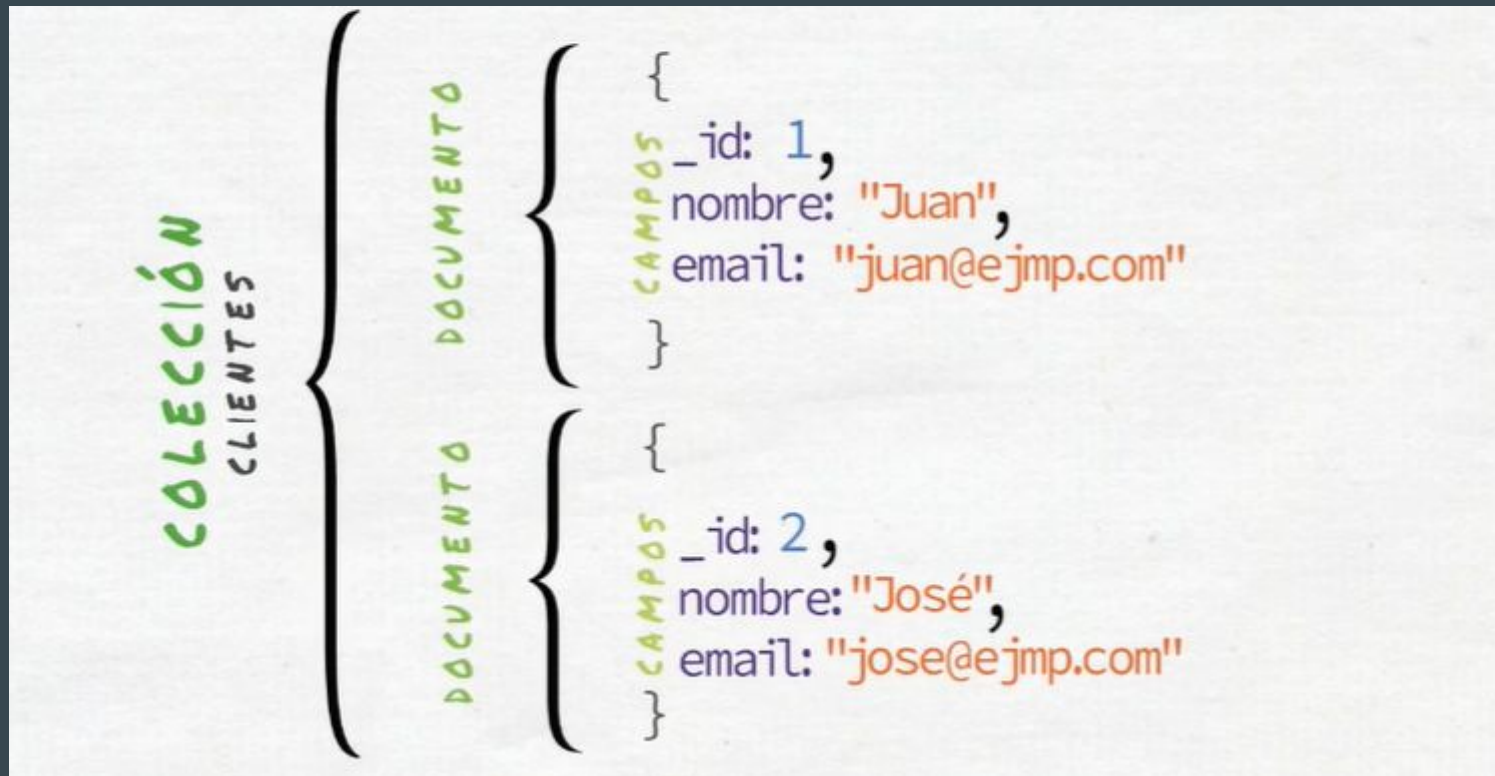
Recordar Relacional Vs NoRelacional



Forma de almacenar los datos en Mongo



Representación de los documentos de la colección “Clientes”



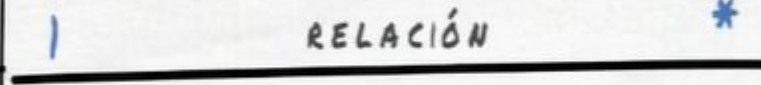
Supongamos que un cliente tiene más de un mail (mundo relacional)

TABLA: CLIENTES

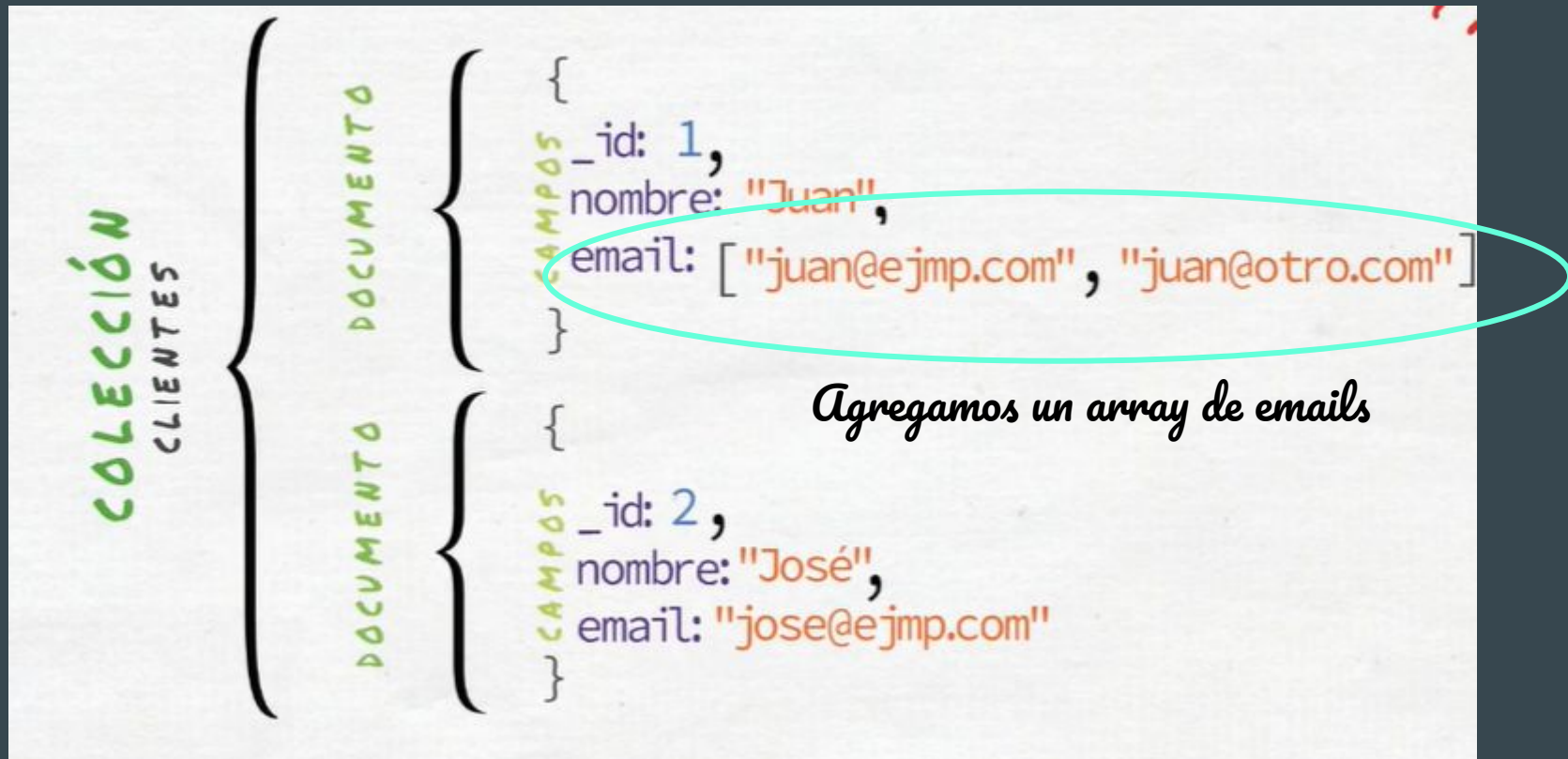
ID	NOMBRE
1	JUAN
2	JOSÉ
3	LUIS

TABLA: EMAILS

ID	ID_CLI	EMAIL
1	1	JUAN@ETMP.COM
2	1	JUAN@OTRO.COM
3	3	LUIS@ETMP.COM



Base de datos ---> Ejemplo del mundo no relacional



Tipos de Datos

Tipo	Alias
Doble	"doble"
Cuerda	"cadena"
Objeto	"objeto"
Arreglo	"matriz"
Datos binarios	"binData"
Indefinido	"indefinido"
ObjectId	"objectId"
Booleano	"bool"
Fecha	"fecha"
Nulo	"nulo"

Tipos de Datos

Nulo	"nulo"
Expresión regular	"regex"
DBPointer	"dbPointer"
JavaScript	"javascript"
Símbolo	"símbolo"
JavaScript (con ámbito)	"javascriptWithScope"
Entero de 32 bits	"int"
Timestamp	"marca de tiempo"
Entero de 64 bits	"largo"
Decimal128	"decimal"
Clave mínima	"minKey"
Tecla máxima	"maxKey"

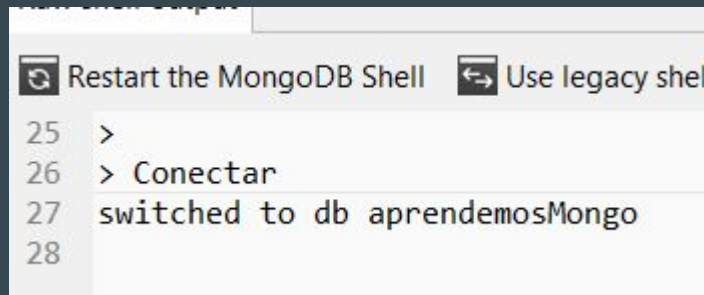
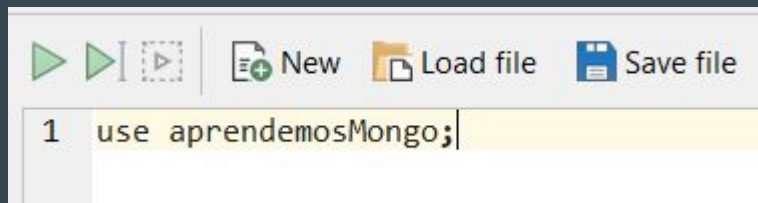
MongoDB CRUD Operations

...

Create
Read
Update
Delete

Crear base de datos

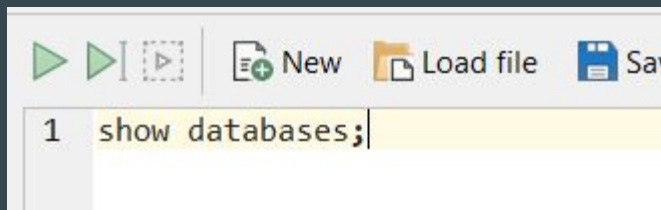
use **nombreBD**;



Tener en cuenta que no se va a ver en la lista de bd de datos hasta que no le pongamos algún documento dentro de ella

Ver base de datos

show dbs;



Raw shell output

30	biblioteca	80.00	KiB
31	blog	112.00	KiB
32	config	108.00	KiB
33	db1	40.00	KiB
34	local	104.00	KiB
35	mongo3	216.00	KiB
36	personas	144.00	KiB

Crear una colección

`db.createCollection(name, options);`

Options es opcional

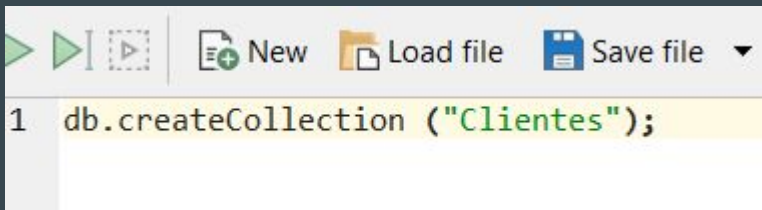
Si es verdadero, habilita una colección limitada. La colección limitada es una colección de tamaño fijo que sobrescribe automáticamente sus entradas más antiguas cuando alcanza su tamaño máximo



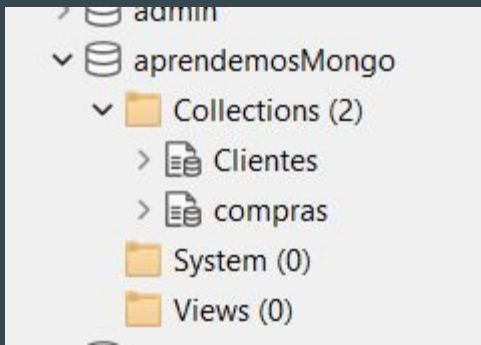
The screenshot shows the MongoDB Shell interface. The command `db.createCollection("compras", { capped : true, size : 6142800, max : 10000 })` is entered in the command line. A green arrow points from the `options` parameter to the text box above. Another green arrow points from the `size` parameter to the text box below. A third green arrow points from the `max` parameter to the text box to the right.

Especifica un tamaño máximo en bytes para una colección con límite.

Especifica el número máximo de documentos permitidos en la colección limitada.



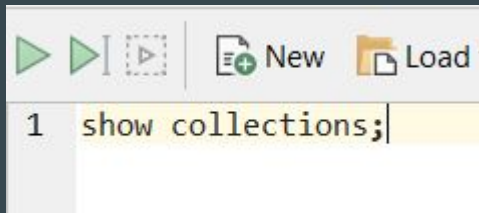
The screenshot shows the MongoDB Shell interface with the command `db.createCollection ("Clientes");` entered in the command line.



Ver las colecciones de una bd

Options es para especificar la configuración de la colección. Este parámetro es opcional

show collections;

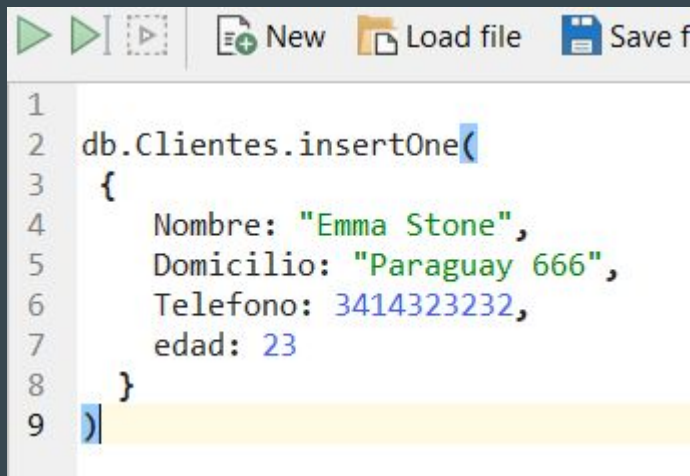


```
1 show collections;
```

```
51 { ok: 1 }
52 Clientes
53 compras
54
```

Insertar un documento en una colección

```
db.nombreColeccion.insertOne({ nombreCampo: 'valor',  
                                nombreCampo2: 'valor2'});
```



The screenshot shows a MongoDB shell window with a toolbar at the top containing icons for running, stepping through, and saving, along with buttons for 'New', 'Load file', and 'Save f'. The command being executed is as follows:

```
1  
2 db.Clientes.insertOne(  
3   {  
4     Nombre: "Emma Stone",  
5     Domicilio: "Paraguay 666",  
6     Telefono: 3414323232,  
7     edad: 23  
8   }  
9 )
```

Result > insertedId		
acknowledged	insertedId	
<input checked="" type="checkbox"/> true	<input checked="" type="checkbox"/> id 650c8643edefe...	

Insertar documentos en una colección

```
db.nombreColeccion.insertMany([ { nombreCampo: 'valor',  
nombreCampo2: 'valor2' }, { nombreCampo: 'valor', nombreCampo2:  
'valor2' } ] );
```

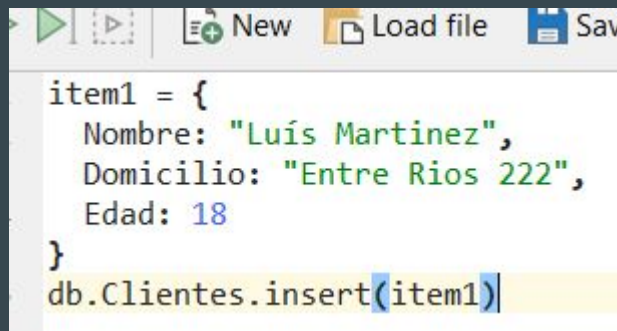
```
1  
2 db.Clientes.insertMany([  
3   {  
4     Nombre: "Juan Perez",  
5     Domicilio: "Rioja 2666",  
6     Telefono: 34132544375,  
7     edad: 25  
8   },  
9   {  
10    Nombre: "Analía Jerez",  
11    Domicilio: "San nicolas 456",  
12    Telefono: 2998765432,  
13    edad: 28  
14  }  
15 ] )
```

Result > insertedIds	
acknowledged	insertedIds
<input checked="" type="checkbox"/> true	<input checked="" type="checkbox"/> { 2 fields }


Insertar un elemento definido previamente

valor = { };

db.nombreColeccion.insertOne(valor);

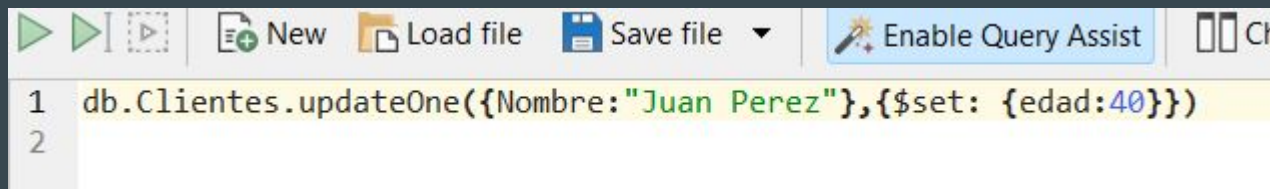


```
item1 = {  
  Nombre: "Luís Martínez",  
  Domicilio: "Entre Rios 222",  
  Edad: 18  
}  
db.Clientes.insert(item1)
```

Result > insertedIds	
acknowledged	insertedIds
<input checked="" type="checkbox"/> true	 { 1 fields }

Modificar un valor de un documento

```
db.nombreColeccion.updateOne({campo:valor}, {$set:  
  {campo:nombre}});
```



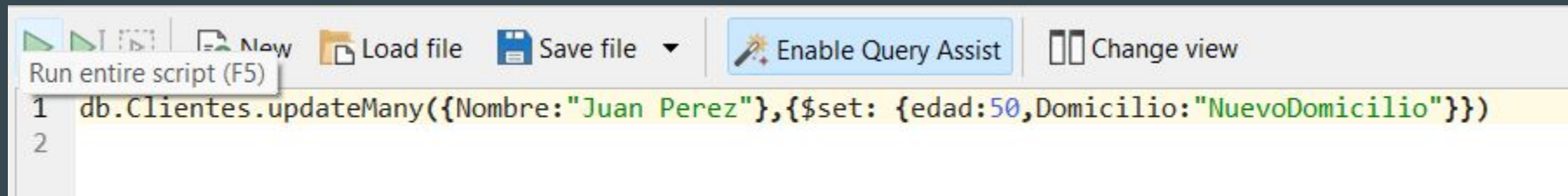
The screenshot shows a MongoDB command line interface with a toolbar at the top containing icons for running, stepping through, and saving queries, along with buttons for 'New', 'Load file', 'Save file', and 'Enable Query Assist'. The command entered is:

```
1 db.Clientes.updateOne({Nombre:"Juan Perez"},{$set: {edad:40}})  
2
```

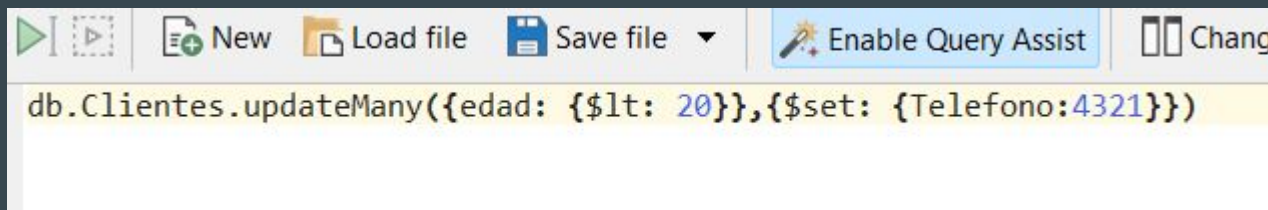
Result > insertedId					
acknowledged	insertedId	matchedCount	modifiedCount	upsertedCount	
<input checked="" type="checkbox"/> true	<input type="checkbox"/> null	<input type="text" value="123"/> 1.0	<input type="text" value="123"/> 1.0	<input type="text" value="123"/> 0.0	

Modificar varios valores

```
db.nombreColeccion.updateMany({elementoABuscar:valor},  
    {$set: {elementoACambiar: nuevo valor,  
            elementoACambiar2: nuevo valor2}})
```



A screenshot of a MongoDB Shell interface. The top toolbar includes icons for 'Run entire script (F5)', 'New', 'Load file', 'Save file', 'Enable Query Assist', and 'Change view'. The script editor shows two lines of code: `1 db.Clientes.updateMany({Nombre:"Juan Perez"},{$set: {edad:50,Domicilio:"NuevoDomicilio"}})` and `2`.



A screenshot of a MongoDB Shell interface. The top toolbar includes icons for 'Run', 'New', 'Load file', 'Save file', 'Enable Query Assist', and 'Change view'. The script editor shows a single line of code: `db.Clientes.updateMany({edad: {$lt: 20}},{$set: {Telefono:4321}})`.
















CON
CONDICION

Ver los documentos de una colección

`db.nombreColeccion.find();`







```
1 db.Clientes.find();
2
```

```
1 db.getCollection("Clientes").find({})
2
```

Clientes > Nombre				
_id	Nombre	Domicilio	Telefono	edad
 650c8643edefe4...	 Emma Stone	 Paraguay 666	 3414323232.0	 23
 650c99dfedefe4...	 Juan Perez	 Rioja 2666	 34132544375.0	 25
 650c99dfedefe4...	 Analía Jerez	 San nicolas 456	 2998765432.0	 28

Tres maneras de ver las colecciones

Raw shell output Find Query (line 1) ✕

← ← → → | 50 | Documents 1 to 3 |      

Cientes > Nombre
















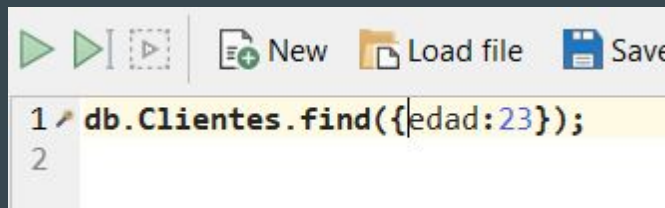
_id	Nombre	Domicilio	Telefono	edad
 650c8643edefe4...	 Emma Stone	 Paraguay 666	 3414323232.0	 23
 650c99dfedefe4...	 Juan Perez	 Rioja 2666	 34132544375.0	 25
 650c99dfedefe4...	 Analía Jerez	 San nicolas 456	 2998765432.0	 28

Table View
Tree View
Table View
JSON View

Ver los documentos con condición

```
db.nombreColeccion.find({campo:valor});
```



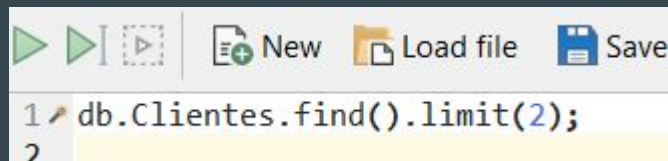
A screenshot of a code editor interface. At the top, there is a toolbar with icons for running (green play button), stepping through (green play button with a vertical line), and a dashed box icon. To the right of these icons are buttons labeled 'New' (with a document icon), 'Load file' (with a folder icon), and 'Save' (with a floppy disk icon). Below the toolbar, there is a text area containing a MongoDB query: `1 db.Clientes.find({edad:23});`. The line number '1' is on the left, and the line number '2' is on the left of the next line, which is empty.

Clientes > Nombre				
_id	Nombre	Domicilio	Telefono	edad
[id] 650c8643edefe4...	[img] Emma Stone	[img] Paraguay 666	[123] 3414323232.0	[332] 23

Ver los documentos con condición

Ver los x primeros documentos

```
db.nombreColeccion.find().limit(nro);
```



```
1 db.Clientes.find().limit(2);
2
```

Clientes > Nombre				
_id	Nombre	Domicilio	Telefono	edad
650c8643edefe4...	Emma Stone	Paraguay 666	3414323232.0	23
650c99dfedefe4...	Juan Perez	Rioja 2666	34132544375.0	25

Ver los documentos con condición >, <, >= o <=

```
db.nombreColeccion.find({campo: { $gt : valor}});
```

```
1 db.Clientes.find({edad:{$gt:24}});
```

Clientes > Nombre

_id	Nombre	Domicilio	Telefono	edad
[id] 650c99dfedefe4...	[icon] Juan Perez	[icon] Rioja 2666	[123] 34132544375.0	[332] 25
[id] 650c99dfedefe4...	[icon] Analía Jerez	[icon] San nicolas 456	[123] 2998765432.0	[332] 28

gt = greater than >

gte = greater than equal >=

lt = lower than <

lte = lower than equal <=

Listado de operadores relacionales

- \$eq - equal - igual
- \$lt - low than - menor que
- \$lte - low than equal - menor o igual que
- \$gt - greater than - mayor que
- \$gte - greater than equal - mayor o igual que
- \$ne - not equal - distinto
- \$in - in - dentro de
- \$nin - not in - no dentro de

Ver los documentos con condición usando AND

```
db.nombreColeccion.find({campo: valor, campo2: valor});
```

Ver los documentos con condición usando OR

```
db.nombreColeccion.find({$or:[{campo: valor}, {campo2: valor}]});
```

Ver los documentos con condición usando IN

```
db.nombreColeccion.find({campo: {$in:[ valor1,valor2]}});
```

Ver los documentos con condición usando ALL

```
db.nombreColeccion.find({campo: {$all:[ valor1,valor2]}});
```

Modificar un documento y agregar un dato en un arreglo

```
db.nombreColeccion.updateOne({campo:valor}, {$push:  
  {campo:nombre}});
```

Modificar un documento y eliminar un dato en un arreglo

```
db.nombreColeccion.updateOne({campo:valor},  
  {$pull: {campo:nombre}});
```

Modificar un documento(modifica “todo el documento”)

**db.nombreColeccion.replaceOne({campo:valor},
{campo:nombre, campo:nombre, etc});**

```
db.temas.replaceOne({nombre:"comandos"},{docente:"juan",años:7|})
```

```
{  
  "acknowledged" : true,  
  "matchedCount" : 1.0,  
  "modifiedCount" : 1.0  
}
```

Se cambia todo el documento cuyo campo coincide con la primer parte del replace y le agrega los campos docente y años

Borrar un documento

db.nombreColeccion.deleteOne({campo:valor});

db.nombreColeccion.deleteMany({campo:valor});

```
> db.temas.find()
{ "_id" : ObjectId("630b982ae082e2860585c805"), "nombre" : "comandos", "duracion" : 10 }
{ "_id" : ObjectId("631584a3a7136f0c8d0b920a"), "nombre" : "inserciones", "duracion" : 20 }
> db.temas.remove({duracion:20})
WriteResult({ "nRemoved" : 1 })
> db.temas.find()
{ "_id" : ObjectId("630b982ae082e2860585c805"), "nombre" : "comandos", "duracion" : 10 }
\
```

Incrementar un valor dentro de un documento

```
db.nombreColeccion.updateOne({campo:valor},  
    {$inc: {campo: +valor}});
```

```
{  
  "_id" : ObjectId("632c4a0706fe4be888b1c50b"),  
  "nombre" : "Daniel",  
  "creditos" : 10.0  
}
```



```
db.estudiantes.updateOne({nombre:"Daniel"},{$inc:{creditos:+5}})
```

```
{  
  "_id" : ObjectId("632c4a0706fe4be888b1c50b"),  
  "nombre" : "Daniel",  
  "creditos" : 15.0  
}
```



Ordenar documentos

Por defecto  Se ordena cómo se crea

```
db.nombreColeccion.find().sort({campo:+1});
```

```
db.nombreColeccion.find().sort({campo:-1});
```

```
db.estudiantes.find().sort({creditos:+1})
```

Comodines

`db.Amigos.find({Nombre:/a/},{Nombre:1})` *Contienen a*

`db.Amigos.find({Nombre:/sa$/},{Nombre:1})` *Terminan con sa*

`db.Amigos.find({Nombre:/^M/},{Nombre:1})` *Terminan con M*

Relaciones Manuales

`db.Amigos.find({Nombre:/a/},{Nombre:1})` *Contienen a*

`db.Amigos.find({Nombre:/sa$/},{Nombre:1})` *Terminan con sa*

`db.Amigos.find({Nombre:/^M/},{Nombre:1})` *Empiezan con M*

MongoDB Relaciones entre Colecciones

...

COMBINAR DOS COLECCIONES

Tenemos diferentes enfoques para combinar dos colecciones en una colección utilizando MongoDB.

- 1 - Usar la instrucción **\$lookup** para unir dos colecciones
- 2 - Usar el operador **\$unwind** para aplanar una matriz antes de adjuntarla a los documentos resultantes
- 4 - Usar el filtro **\$project** en las consultas de agregación para unir dos colecciones

COMBINAR DOS COLECCIONES - \$lookup

Creamos la colección usuario

```
db.usuario.insertMany(  
  [  
    {  
      fullname: 'Juan Perez',  
      age: 30,  
      gender: 'Masculino',  
      nationality: 'Argentino'  
    },  
    {  
      fullname: 'Maria Gomez',  
      age: 45,  
      sex: 'Femenino',  
      nationality: 'Española'  
    }  
  ]  
)
```

COMBINAR DOS COLECCIONES - \$lookup

Creamos la colección dirección

```
db.direccion.insertMany([
  {
    fullname: 'Juan Perez',
    block_number: 22,
    street: 'Mitre',
    city: 'Rosario'
  },
  {
    fullname: 'Maria Gomez',
    block_number: 30,
    street: 'San juan',
    city: 'Barcelona'
  }
])
```

COMBINAR DOS COLECCIONES - \$lookup

Usamos \$lookup para unir las dos colecciones en la misma

```
db.usuario.aggregate([
  { $lookup:
    {
      from: 'direccion',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'Domicilio'
    }
  }
]).pretty();
```


COMBINAR DOS COLECCIONES - \$lookup

Usamos \$lookup para unir las dos colecciones en la misma. Resultado

```
{
  "_id" : ObjectId("633c4c34e47880b9ab98f18b"),
  "fullname" : "Maria Gomez",
  "age" : 45.0,
  "sex" : "Femenino",
  "nationality" : "Española",
  "Domicilio" : [
    {
      "_id" : ObjectId("633c4c93e47880b9ab98f18d"),
      "fullname" : "Maria Gomez",
      "block_number" : 30.0,
      "street" : "San juan",
      "city" : "Barcelona"
    }
  ]
}
```

COMBINAR DOS COLECCIONES - \$lookup

La función **\$lookup** acepta cuatro campos.

- ❖ Primero está el **campo from**, donde especificamos la colección que se supone que debe unirse con la otra colección.
- ❖ El segundo es el **campo localField**. Es uno de los atributos (campo) de los documentos de entrada de la colección especificados en el campo from.
- ❖ De manera similar, el tercer campo llamado **foreignField**, realiza la coincidencia de igualdad en foreignField con localField de los documentos de las colecciones.
- ❖ Por último, escribimos el nombre de la nueva matriz para el cuarto campo, **as**

COMBINAR DOS COLECCIONES - \$lookup

```
db.userInformation.aggregate([
  { $lookup:
    {
      from: 'userAddress',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'address' }
    }
  ]).pretty();
```

name of new array that would be created in **userInformation**

COMBINAR DOS COLECCIONES - \$unwind

Usamos \$unwind para aplanar la matriz antes de adjuntarla al documento resultante

```
db.usuario.aggregate([
  { $lookup:
    {
      from: 'direccion',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'domicilio'
    }
  },
  {
    $unwind: '$domicilio'
  }
]).pretty();
```

COMBINAR DOS COLECCIONES - \$unwind

Usamos \$unwind para aplanar la matriz antes de adjuntarla al documento resultante

	_id	fullname	age	gender	nationality	domicilio	sex
1	 ObjectId("6...)	 Juan Perez	 30.0	 Masculino	 Argentino	 { 5 fields }	
2	 ObjectId("6...)	 Juan Perez	 30.0	 Masculino	 Argentino	 { 5 fields }	
3	 ObjectId("6...)	 Maria Gomez	 45.0		 Española	 { 5 fields }	 Femenino

La diferencia fundamental para el operador \$unwind es que transforma una matriz con un solo elemento en el objeto aplanado,

COMBINAR DOS COLECCIONES - \$project

Usamos \$project para unir dos colecciones y **ver solamente algunos campos**

EJEMPLO: Supongamos que no queremos unir toda la colección denominada direccion con usuario, sólo queremos unir los campos city y street.

En este caso usaremos la instrucción \$addfields para mostrar ciertos campos en el resultado.

COMBINAR DOS COLECCIONES - \$project

```
db.usuario.aggregate([
  { $lookup:
    {
      from: 'direccion',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'domicilio'
    }
  },
  {
    $unwind: '$domicilio'
  },
  {
    $addFields: {
      street: '$domicilio.street',
      city: '$domicilio.city'
    }
  }
]).pretty();
```

COMBINAR DOS COLECCIONES - \$project

```
{  
  "id" : ObjectId("628bc4a45c544feccff5a566"),  
  "fullname" : "Juan Perez",  
  "age" : 22,  
  "gender" : "Masculino",  
  "nationality" : "Argentino",  
  "domicilio" : {  
    "id" : ObjectId("628bc4ae5c544feccff5a568"),  
    "fullname" : "Juan Perez",  
    "block_number" : 22,  
    "street" : "Mitre",  
    "city" : "Rosario"  
  },  
  "street" : "Mitre",  
  "city" : "Rosario"  
}
```


COMBINAR DOS COLECCIONES - \$project

EJEMPLO: El resultado nos trae street y city, pero también nos devuelve el objeto “domicilio” que no queremos ver.

Para solucionar esto, usaremos el filtro **\$project**, de forma tal que podamos especificar que campos queremos ver en el resultado.

COMBINAR DOS COLECCIONES - \$project

```
db.usuario.aggregate([
  { $lookup:
    {
      from: 'userAddress',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'domicilio'
    }
  },
  {
    $unwind: '$domicilio'
  },
  {
    $addFields: {
      street: '$domicilio.street',
      city: '$domicilio.city'
    }
  },
  {
```

```
    {
      $project: {
        fullname: 1,
        age: 1,
        gender: 1,
        street: 1,
        city: 1
      }
    }
  ]).pretty();
```

COMBINAR DOS COLECCIONES - \$project

```
{
  "_id" : ObjectId("628bc4a45c544feccff5a566"),
  "fullname" : "Maria Gomez",
  "age" : 45,
  "gender" : "Femenino",
  "street" : "San Juan",
  "city" : "Barcelona"
}
{
  "_id" : ObjectId("628bc4a45c544feccff5a567"),
  "fullname" : "Juan Perez",
  "age" : 22,
  "street" : "Mitre",
  "city" : "Rosario"
}
```

\$match :Filtra los documentos para pasar solo los documentos que coinciden con las condiciones especificadas

```
db.noticias.aggregate([
  {
    $match: { nombre: "Juan" }
  }
])
```

\$count :Cuenta la cantidad de documentos de una colección

```
db.usuarios.aggregate([
  {
    $count: "cantidad de documentos"
  }
])
{
  "cantidad de documentos" : 3
}
```

\$group : Agrupa valores según ciertos filtros

```
db.noticias.aggregate( [  
  {  
    $match: { nombre: "Juan" }  
  },  
  {  
    $group: { _id: "$nombre", cantidadPalabras: { $sum: "$palabras" } }  
  }  
] )|
```

```
{  
  "_id" : "Juan",  
  "cantidadPalabras" : 220.0  
}
```

\$sort : Ordena los datos obtenidos. Como parámetro recibe un objeto con los diferentes campos sobre los cuales se ordenará, y un orden. El orden puede ser 1 si es ascendente o -1 si es descendente.

Se ordenan los resultados

```
db.usuario.aggregate([
  { $lookup:
    {
      from: 'direccion',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'Domicilio'
    }
  },
  { $project: {
    fullname: 1,
    age: 1
  } },
  { $sort: { age: 1 } }
]);
```

\$skip: Nos permite saltarnos los n primeros documentos que se obtengan en la agregación, donde n es un entero que pasamos como parámetro.

```
db.usuario.aggregate([
  { $lookup:
    {
      from: 'direccion',
      localField: 'fullname',
      foreignField: 'fullname',
      as: 'Domicilio'
    }
  },
  { $project: {
    fullname: 1,
    age: 1
  } },
  { $skip: 2 }
]);
```

*Muestra a partir
del documento nro. 3*

Eliminar una colección

`db.nombreColeccion.drop();`

```
> show collections
nuevaColeccion
prueba
segundaPrueba
temas
> db.nuevaColeccion.drop()
true
> show collections
prueba
segundaPrueba
temas
```

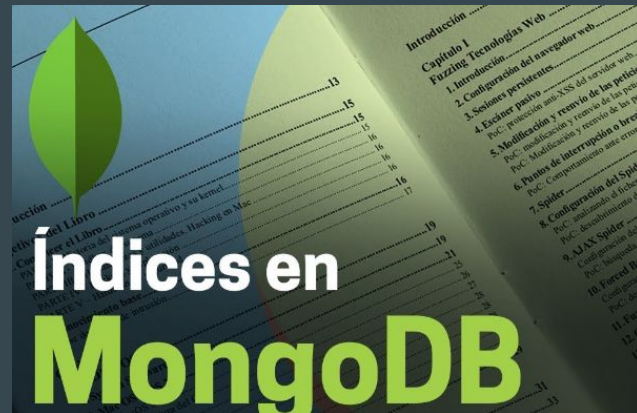

Eliminar base de datos

```
db.dropDatabase();
```

```
> show dbs
admin          0.000GB
borrando       0.000GB
config         0.000GB
cursoMongo2    0.000GB
local          0.000GB
prueba         0.000GB
> db.dropDatabase()
{ "ok" : 1 }
> show dbs
admin          0.000GB
config         0.000GB
cursoMongo2    0.000GB
local          0.000GB
prueba         0.000GB
```

ÍNDICES EN MONGODB

Los índices hacen que la consulta de datos sea más eficiente. Sin índices, MongoDB realiza una exploración de la colección que lee todos los documentos de la colección para determinar si los datos coinciden con las condiciones especificadas en la consulta. Los índices limitan el número de documentos que MongoDB lee y con los índices adecuados se puede mejorar el rendimiento.



ÍNDICES EN MONGODB

QUé son los índices???

Los índices son estructuras de datos especiales que almacenan una pequeña parte de la conjunto de datos de la colección en un formato fácil de recorrer.

El índice almacena el valor de un campo específico o un conjunto de campos, ordenados por el valor del campo. El orden de las entradas de índice admite Coincidencias de igualdad eficientes y operaciones de consulta basadas en intervalos.

ÍNDICES EN MONGODB

```
1 db.getCollection("noticias").find({nombre:"Frank"})  
2
```



Count Documents



00:00:00.029

```
db.usuarios.createIndex({nombre:1})  
db.getCollection("noticias").find({nombre:"Frank"})
```



Count Documents



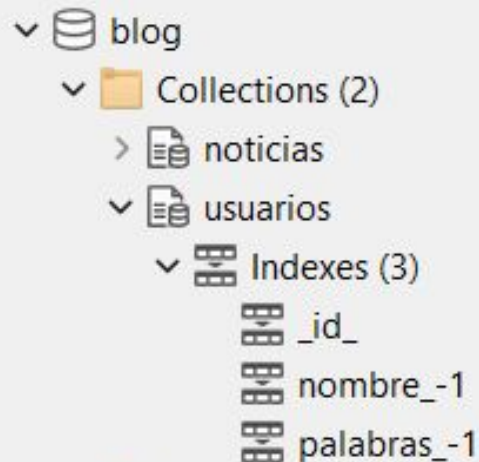
00:00:00.002

ÍNDICES EN MONGODB

Crear índices

```
db.NOMBRE_DE_LA_COLECCIÓN.createIndex({CLAVE:1},{name:
“nombreClave”})
```

```
db.usuarios.createIndex( { "nombre":-1 } )
```

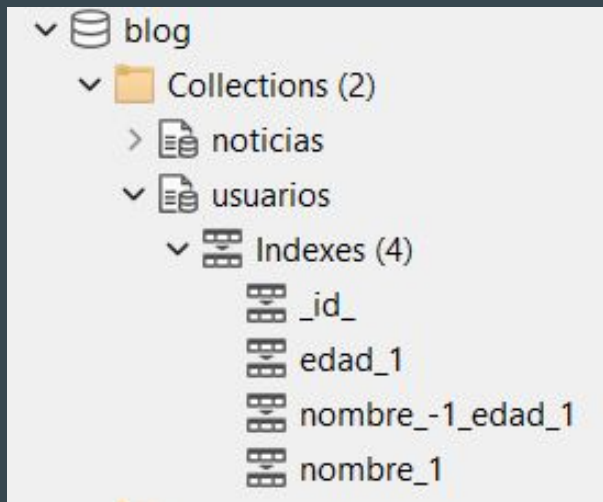


ÍNDICES EN MONGODB

Crear índices múltiples

```
db.NOMBRE_DE_LA_COLECCIÓN.createIndex({CLAVE:1, CLAVE2:1})
```

```
db.usuarios.createIndex(  
{ "nombre":-1, "edad:1 } )
```

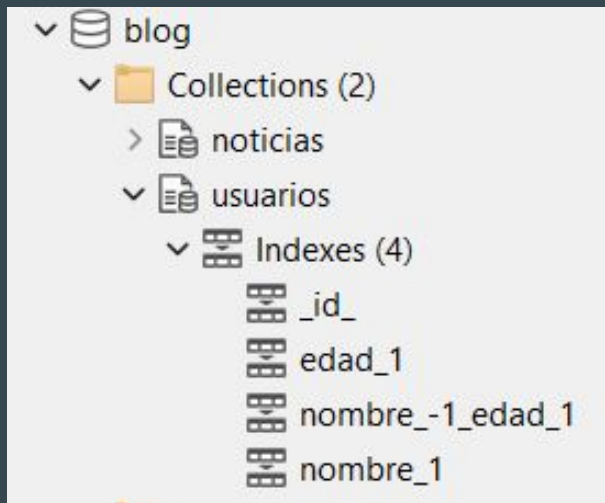


ÍNDICES EN MONGODB

Crear índices único

```
db.NOMBRE_DE_LA_COLECCIÓN.createIndex({CLAVE:1},{unique:true})
```

```
db.usuarios.createIndex(  
  { "dni":1 } ,{unique:true})
```



ÍNDICES EN MONGODB

Borrar un índices

```
db.NOMBRE_DE_LA_COLECCIÓN.dropIndex({"nombre"})
```

```
db.usuarios.dropIndex( { "dni" })
```


ÍNDICES EN MONGODB

Crear índices

```
db.NOMBRE_DE_LA_COLECCIÓN.createIndex ( {CLAVE:1} )
```

```
db.usuarios.createIndex( { "nombre":-1 } )
```

