



**DBD**

Diseño de  
Bases de Datos

# MongoDb



Universitat d'Alacant  
Universidad de Alicante

**lsi** Departamento  
de Lenguajes  
y Sistemas  
Informáticos

# INTRODUCCIÓN mongoDB

- **MongoDB** es una base de datos **noSQL** de tipo documental.
- **Su nombre viene de la palabra inglesa “humongous”** que significa enorme
- Almacena la información en documentos tipo BSON (JSON Binario).
- Carece de esquema predefinido.

# INTRODUCCIÓN mongoDB

- En mongoDB una **base de datos** es una **colección de documentos**.
- Estos documentos se componen de campos/características ... de distintos tipos de datos.

BASE DE DATOS {  
COLECCIÓN

{

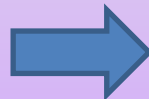
DOCUMENTOS

{

CARACTERÍSTICA: VALOR/ES

}

}}



## *SIMILITUD CON BD RELACIONAL*

COLECCIÓN → TABLA

DOCUMENTOS → FILAS

CARACTERÍSTICA/S → COLUMNAS

VALOR/ES → DATOS DE COLUMNAS

# INTRODUCCIÓN mongoDB

- {
- Nombre : “Departamento de Lenguajes y Sistemas Informáticos”,
- Empleados: 70,
- Asignaturas: [ “Diseño de Bases de Datos”, “Fundamentos de Bases de Datos”, “Programación”, “Gestión de la Información” ],
- Dirección: { calle: “Carretera de San Vicente”,
- número: 2,
- codigopostal: 03690
- },
- Teléfono: { centralUA: { número: 965903400,
- extensión: 3972 }
- directo: [965903466, 965943211]
- }
- }

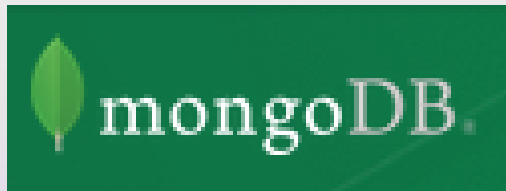
# INTRODUCCIÓN mongoDB

**MongoDB**, a través de JSON, puede utilizar los siguientes tipos de datos:

- String: guardados en UTF-8. Van siempre entre comillas dobles.
- Number: números. Al guardarse en BSON pueden ser de tipo byte, int32, int64 o double.
- Boolean: con valor true o false.
- Array: van entre corchetes [] y pueden contener de 1 a N elementos, que pueden ser de cualquiera de los otros tipos.
- Documentos: un documento en formato JSON puede contener otros documentos embebidos que incluyan más documentos o cualquiera de los tipos anteriormente descritos.
- Null.

# INTRODUCCIÓN mongoDB

- Para empezar a trabajar con mongoDB lo haremos en modo local.



**1.-** Iniciamos el servidor  
**mongod**



**2.-** Abrimos un terminal  
**mongo**

# INTRODUCCIÓN mongoDB

> **show dbs**

admin	0.000GB
comercio	0.000GB
config	0.000GB
local	0.000GB
profesores	0.000GB
practicasyDBD	0.000GB

Muestra las bases de datos creadas

> **use comercio**

switched to db comercio

Indica la base de datos a utilizar.

> **show collections**

clientes

Muestra las colecciones creadas.

> **db**

comercio

Muestra la base de datos en la que te encuentras.

- La inserción de datos se realiza a través de colecciones con las funciones **.insert()** o **.insertOne()**

La colección crea con el primer documento que le insertas, no hace falta crearla previamente (aunque se puede hacer).

**Un documento de la colección**

```
{  
  nombre: 'Pedro',  
  apellido1: 'León',  
  apellido2: 'Roldán'  
}
```

**JSON**

Javascript Object Notation



# INSERT

mongoDB

Nombre de la colección

operación

```
> db.clientes.insert({nombre: 'Pedro', apellido1: 'León', apellido2: 'Roldán'})  
WriteResult({ "nInserted" : 1 })
```

1 documento insertado

db. Indica que va a actuar en la base de datos donde nos encontremos en ese momento

# FIND

# mongoDB

Para ver la información almacenada en una colección se utiliza la función **.find()**

```
> db.clientes.find()  
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }  
>
```

**\_id**: Identificador único asignado por MongoDB  
si no se lo asignamos nosotros

Para ver la información almacenada en una colección en un formato más legible se utiliza **find().pretty()**

```
> db.clientes.find().pretty()
{
  "_id" : ObjectId("5c0275f8f87ed69a71545d14"),
  "nombre" : "Pedro",
  "apellido1" : "León",
  "apellido2" : "Roldán"
}
```

Para insertar varios clientes de golpe, se utiliza un **array** de clientes, `[ ]` o usando **`.insertMany()`** con el array

```
> db.clientes.insert([
  {nombre: 'Laura', apellido1: 'Rodríguez', apellido2: 'Sanz'},
  {nombre: 'Andrea', apellido1: 'Lara', apellido2: 'Sempere'},
  {nombre: 'Miguel', apellido1: 'Cobos', apellido2: 'Pascual'},
  {nombre: 'Manuel', apellido1: 'Beltrán', apellido2: 'Sanz'}
])
```

```
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 4,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
```

Si quisiésemos asignar el  
identificador(único)

```
> db.clientes.insert(
  {_id:1, nombre: 'Laura',
  apellido1: 'Rodríguez', apellido2:
  'Sanz'},
```

→ **4 inserciones**

Para insertar varios clientes de golpe, se utiliza un **array** de clientes, [ ] o usando **.insertMany()** con el array

```
> db.clientes.insertMany([  
  {nombre: 'Laura', apellido1: 'Rodríguez', apellido2: 'Sanz'},  
  {nombre: 'Andrea', apellido1: 'Lara', apellido2: 'Sempere'},  
  {nombre: 'Miguel', apellido1: 'Cobos', apellido2: 'Pascual'},  
  {nombre: 'Manuel', apellido1: 'Beltrán', apellido2: 'Sanz'}  
])
```

```
BulkWriteResult({  
  "writeErrors" : [ ],  
  "writeConcernErrors" : [ ],  
  "nInserted" : 4,  
  "nUpserted" : 0,  
  "nMatched" : 0,  
  "nModified" : 0,  
  "nRemoved" : 0,  
  "upserted" : [ ]  
})
```

Si quisiésemos asignar el  
identificador(único)

```
> db.clientes.insert(  
  {_id:1, nombre: 'Laura',  
   apellido1: 'Rodríguez', apellido2:  
   'Sanz'},
```

→ **4 inserciones**

# FIND

# mongoDB

## Vemos los clientes insertados:

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" :
"León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Laura", "apellido1" :
"Rodríguez", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" :
"Lara", "apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" :
"Cobos", "apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" :
"Beltrán", "apellido2" : "Sanz" }
>
```

# FIND

# mongoDB

Si queremos ver los clientes insertados en un formato más legible:

```
> db.clientes.find().pretty()
{
  "_id" : ObjectId("5c0275f8f87ed69a71545d14"),
  "nombre" : "Pedro",
  "apellido1" : "León",
  "apellido2" : "Roldán"
}
{
  "_id" : ObjectId("5c02c396735bd96b3e9ec858"),
  "nombre" : "Laura",
  "apellido1" : "Rodríguez",
  "apellido2" : "Sanz"
}
{
  "_id" : ObjectId("5c02c396735bd96b3e9ec859"),
  "nombre" : "Andrea",
  "apellido1" : "Lara",
  "apellido2" : "Sempere"
}
{
  "_id" : ObjectId("5c02c396735bd96b3e9ec85a"),
  "nombre" : "Miguel",
  "apellido1" : "Cobos",
  "apellido2" : "Pascual"
}
{
  "_id" : ObjectId("5c02c396735bd96b3e9ec85b"),
  "nombre" : "Manuel",
  "apellido1" : "Beltrán",
  "apellido2" : "Sanz"
}
```

# INSERT

# mongoDB

A diferencia de una base de datos relacional, donde en una tabla todas las filas tienen las mismas columnas, en mongoDB no es necesario que todos los clientes tengan los mismos datos. Se puede insertar un registro con un nuevo dato, por ejemplo sexo, que el resto de clientes no tenían:

```
> db.clientes.insert({nombre: 'Rosa', apellido1: 'Rodríguez', apellido2: 'Sanz',  
sexo:'mujer'});  
WriteResult({ "nInserted" : 1 })
```



# FIND

# mongoDB

## Y podemos ver el resultado con **.find()**

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León",
"apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Laura", "apellido1" : "Rodríguez",
"apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara",
"apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos",
"apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán",
"apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez",
"apellido2" : "Sanz", "sexo" : "mujer" }
>
```

# FIND

# mongoDB

Si queremos buscar los clientes que tengan un valor en una de las características

**find({característica:valor})**

*Vamos a buscar clientes que en la característica apellido2 tengan el valor Sanz*

```
> db.clientes.find({apellido2:'Sanz'})
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Laura", "apellido1" : "Rodríguez", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }
```

## ¿Qué ocurre si nos equivocamos al poner el nombre de una característica en la inserción?

*Insertamos unas características en la colección pero indicando apellido2 (en vez de apellido1)*

```
> db.clientes.insert({nombre: 'Sergio', apellido1: 'Valiente', apellido2: 'Sanz'});  
WriteResult({ "nInserted" : 1 })
```

*Al realizar la búsqueda por **apellido2** no aparecerá puesto que la hemos llamado **apellido2** (se creará una nueva característica: apellido2)*

```
> db.clientes.find({apellido2: 'Sanz'})  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Laura", "apellido1" : "Rodríguez", "apellido2" :  
"Sanz" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" :  
"Sanz" }  
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" :  
"Sanz", "sexo" : "mujer" }
```

Vamos a partir de que tenemos los siguientes datos y queremos realizar una búsqueda por identificador:


```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León",
"apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Laura", "apellido1" : "Rodríguez",
"apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara",
"apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos",
"apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán",
"apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez",
"apellido2" : "Sanz", "sexo" : "mujer" }
>
```

# FIND

# mongoDB

Buscamos por identificador, indicando la característica y el valor:

```
> db.clientes.find({_id:'5c0275f8f87ed69a71545d14'})  
>
```



Es necesario utilizar la función **ObjectId()**



```
> db.clientes.find({_id:ObjectId('5c0275f8f87ed69a71545d14')})  
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" :  
"León", "apellido2" : "Roldán" }
```

# UPDATE

# mongoDB

Cuando queremos modificar datos de alguna colección se utiliza la función **.update()**

*Vamos a cambiar el nombre de la característica **apellido2** por **apellido2***

```
> db.clientes.update(  
... {apellido2:'Sanz'},  Características que debe cumplir el documento a modificar  
... {  
... nombre:'Sergio',  
... apellido1:'Valiente',  
... apellido2:'Sanz'  Nuevo contenido del documento  
... }  
... )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

## Suponemos que tenemos los siguientes datos

*Vamos a cambiar el nombre de un cliente puesto que es incorrecto*

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Laura", "apellido1" : "Rodríguez", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }
>
```

## Para cambiar datos, utilizamos .update()

*Cambiamos el nombre de Laura por el de Lara*

Características que debe cumplir el documento a modificar

```
> db.clientes.update({apellido1:'Rodríguez', apellido2:'Sanz'},  
{nombre:'Lara', apellido1:'Rodríguez', apellido2:'Sanz', sexo:'mujer'})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Otra forma de hacerlo accediendo por el \_id

```
> db.clientes.update({_id: ObjectId('5c02c396735bd96b3e9ec858')},  
{nombre:'Lara', apellido1:'Rodríguez', apellido2:'Sanz', sexo:'mujer'})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```



# UPDATE

# mongoDB

## Y volvemos a ver los datos:

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }
>
```

Hay que tener en cuenta que update cambia el documento anterior por el que pasamos, por lo que si no incluimos las características completas, aunque no cambien, se perderán.

*Por ejemplo, añadimos un dato con un apellido erróneo*

```
> db.clientes.insert({nombre: 'Mario', apellido1: 'Valiente', apellido2: 'xxx'});  
WriteResult({ "nInserted" : 1 })
```

*Pero nos damos cuenta y lo cambiamos con .update()*

```
> db.clientes.update( {apellido2:'xxx'}, { apellido2:'Oncina'} )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Al especificar solo la característica que queremos cambiar, se ha perdido el resto

```
>db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León",
"apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "apellido1" : "Rodríguez",
"apellido2" : "Sanz", "sexo" : "mujer" }
.....
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez",
"apellido2" : "Sanz", "sexo" : "mujer" }
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente",
"apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02d0e1735bd96b3e9ec85e"), "apellido2" : "Oncina" }
>
```



Se ha perdido toda la información previa que teníamos de este documento

**{nombre: 'Mario', apellido1: 'Valiente', apellido2: 'xxx'}**

No es necesario repetir todas las características cuando necesitamos realizar un cambio, el problema es que no estábamos utilizando bien la función update. Para cambiar determinadas características sin sustituir el resto se debe especificar **\$set** en la función .update()

# UPDATE

# mongoDB

```
> db.clientes.update(  
... {_id:ObjectId('5c02c396735bd96b3e9ec858')},  
... {  
... $set:{edad:30}  
... }  
... )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
>
```

```
> db.clientes.find()  
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "apellido1" :  
"Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer", "edad" : 30 }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }  
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02d0e1735bd96b3e9ec85e"), "apellido2" : "Oncina" }
```

## Podemos ver otro ejemplo:

*Insertamos un documento pero con una característica equivocada*

```
> db.clientes.insert({nombre: 'Pilar', apellido1: 'Valiente', apellido2: 'xxx'});  
WriteResult({ "nInserted" : 1 })
```

```
> db.clientes.find()  
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer", "edad" : 30 }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }  
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02d0e1735bd96b3e9ec85e"), "apellido2" : "Oncina" }  
{ "_id" : ObjectId("5c02e27a735bd96b3e9ec85f"), "nombre" : "Pilar", "apellido1" :  
"Valiente", "apellido2" : "xxx" }
```

Y modificamos la característica equivocada sin perder el resto de información:

```
> db.clientes.update(  
... {_id:ObjectId('5c02e27a735bd96b3e9ec85f')},  
... {  
... $set:{apellido2:'Ruiz',edad:30}  
... }  
... )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.clientes.find({_id:ObjectId('5c02e27a735bd96b3e9ec85f')})  
{ "_id" : ObjectId("5c02e27a735bd96b3e9ec85f"), "nombre" : "Pilar", "apellido1" :  
"Valiente", "apellido2" : "Ruiz", "edad" : 30 }
```

Otra acción que podemos realizar es incrementar el valor de una característica con **\$inc**

*Vamos a decrementar la edad de Lara en 2 años menos, pero realizamos la búsqueda por su apellido (Rodríguez)*

```
> db.clientes.find({apellido1:'Rodríguez'})
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara",
  "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer", "edad" : 30 }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa",
  "apellido1" : "Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }
>
```



## Así se utiliza \$inc

```
> db.clientes.update(  
... {apellido1:'Rodríguez'},  
... {$inc:{edad:-2}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.clientes.find({apellido1:'Rodríguez'}).pretty()  
{  
  "_id" : ObjectId("5c02c396735bd96b3e9ec858"),  
  "nombre" : "Lara",  
  "apellido1" : "Rodríguez",  
  "apellido2" : "Sanz",  
  "sexo" : "mujer",  
  "edad" : 28  
}  
{  
  "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"),  
  "nombre" : "Rosa",  
  "apellido1" : "Rodríguez",  
  "apellido2" : "Sanz",  
  "sexo" : "mujer"  
}
```

Otro ejemplo usando **\$inc** *(ahora con el mismo criterio que antes, vamos a incrementar la edad de en 7 años más)*

```
> db.clientes.update(  
... {apellido1:'Rodríguez'},  
... {  
... $inc:{edad:7}  
... }  
... )  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.clientes.find({apellido1:'Rodríguez'})  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "apellido1" :  
"Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer", "edad" : 35 }  
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" :  
"Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer" }  
>
```

- Vemos que a pesar de que hay varios registros donde el apellido1 es Rodríguez, solo ha cambiado el primer registro que encuentra.
- Este es el comportamiento por defecto de mongoDB para evitar errores, si queremos que afecte a todas las colecciones que cumplan el criterio se debe especificar un tercer parámetro **{multi:true}**

# UPDATE

# mongoDB

Así se utiliza **{multi:true}** cuando queremos actualizar valores de varios documentos que cumplan una condición (*actualizamos el valor de edad a 35 para todos los documentos cuyo apellido1 sea Rodríguez*)

```
> db.clientes.update(  
... {apellido1:'Rodríguez'},  
... {$set:{edad:35}},  
... {multi:true}  
... )  
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
```

```
> db.clientes.find({apellido1:'Rodríguez'})  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "apellido1" :  
"Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer", "edad" : 35 }  
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" :  
"Rodríguez", "apellido2" : "Sanz", "sexo" : "mujer", "edad" : 35 }
```

Hay otra opción interesante a través de la que se puede eliminar una característica de un documento: **\$unset**

*Vamos a borrar la característica sexo de Rosa*

```
> db.clientes.update(  
  {nombre:'Rosa',apellido1:'Rodríguez'},  
  {  
    $unset:{sexo:0}  
  }  
)  
WriteResult({ "nMatched" : 1,  
  "nUpserted" : 0, "nModified" : 1 })
```

```
> db.clientes.find({apellido1:'Rodríguez'}).pretty()  
{  
  "_id" : ObjectId("5c02c396735bd96b3e9ec858"),  
  "nombre" : "Lara",  
  "apellido1" : "Rodríguez",  
  "apellido2" : "Sanz",  
  "sexo" : "mujer",  
  "edad" : 35  
}  
{  
  "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"),  
  "nombre" : "Rosa",  
  "apellido1" : "Rodríguez",  
  "apellido2" : "Sanz"  
}
```

## Otro ejemplo de **\$unset** combinado con otras operaciones (\$inc)

*Vamos a borrar la característica apellido2 de Lara y además incrementaremos su edad en 3*

```
> db.clientes.update(
... {nombre:'Lara'},
... {
... $set:{apellido1:'Ramírez'},
... $unset:{apellido2:0},
... $inc:{edad:3}
... }
... )
```

```
> db.clientes.find({nombre:'Lara'}).pretty()
{
  "_id" : ObjectId("5c02c396735bd96b3e9ec858"),
  "nombre" : "Lara",
  "apellido1" : "Ramírez",
  "sexo" : "mujer",
  "edad" : 38
}
```

# UPDATE

mongoDB

Hay un parámetro extra ***{upsert:true}*** que permite insertar el documento si este no existe pero en el caso de que exista actualizará los datos sustituyendo las características actuales por las nuevas especificadas.

Si intentamos actualizar un documento que no existe con un update sin más, vemos que no afecta a ningún documento porque no lo encuentra, ya que no existe:

```
> db.cliente.update(  
... { nombre:'Samuel'},  
... {  
... nombre:'Samuel',  
... apellido1:'Carrasco',  
... apellido2:'Carratalá'  
... }  
... )  
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
```



# UPDATE

# mongoDB

Pero cambia de comportamiento con {upsert:true}

```
> db.cliente.update(  
... {nombre:'Samuel'},  
... {  
... nombre:'Samuel',  
... apellido1:'Carrasco',  
... apellido2:'Carratalá'  
... },  
... {upsert: true}  
... )  
WriteResult({  
  "nMatched" : 0,  
  "nUpserted" : 1,  
  "nModified" : 0,  
  "_id" : ObjectId("5c02f5c9b61a0cb0edb5781c")  
})
```

*Como detecta  
que el  
documento no  
existe, lo crea*

# UPDATE

# mongoDB

Si nos equivocamos al crear una característica podríamos borrarla y volverla a crear, pero es más sencillo y consume menos recursos utilizar el comando **\$rename**

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "edad" : 38, "apellido1" : "Ramírez", "genero" : "mujer" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }
...
```

```
> db.clientes.update( {nombre:"Lara"}, { $rename:{ "genero":"sexo", "apellido1":"apel1" } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

El resultado del update anterior sería este:

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "edad" : 38, "apel1" : "Ramírez", "sexo" :
"mujer" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" :
"Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" :
"Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" :
"Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz"
}
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente", "apellido2" : "Sanz"
}
{ "_id" : ObjectId("5c02d0e1735bd96b3e9ec85e"), "apellido2" : "Oncina" }
{ "_id" : ObjectId("5c02e27a735bd96b3e9ec85f"), "nombre" : "Pilar", "apellido1" : "Valiente", "apellido2" : "Ruiz",
"edad" : 30 }
```

Para eliminar datos de nuestra colección, utilizaremos la función **.remove()** que recibe como parámetro la consulta que se utilizará para filtrar los documentos que se borrarán.

Si no especificamos ninguna consulta, se eliminarán todos los datos de la colección (el comportamiento es muy similar al de una operación ***delete*** de una base de datos relacional donde si no especificamos un filtro con la sentencia ***where*** se borrarán todos los datos de la tabla).

# REMOVE

# mongoDB

Si tenemos los siguientes datos y queremos borrar la colección que solo tiene la característica *apellido2:"Oncina"*

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "edad" : 38, "apel1" : "Ramírez", "sexo" : "mujer" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02d0e1735bd96b3e9ec85e"), "apellido2" : "Oncina" }
{ "_id" : ObjectId("5c02e27a735bd96b3e9ec85f"), "nombre" : "Pilar", "apellido1" : "Valiente", "apellido2" : "Ruiz", "edad" : 30 }
```

# REMOVE

mongoDB

Utilizaremos **.remove()** indicando la condición especificada

```
> db.clientes.remove({apellido2:'Oncina'})  
WriteResult({ "nRemoved" : 1 })
```

```
> db.clientes.find()  
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "edad" : 38, "sexo" : "mujer", "apellido1" : "Ramírez" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }  
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente", "apellido2" : "Sanz" }  
{ "_id" : ObjectId("5c02e27a735bd96b3e9ec85f"), "nombre" : "Pilar", "apellido1" : "Valiente", "apellido2" : "Ruiz", "edad" : 30 }
```

# REMOVE

# mongoDB

Y podemos ver otro ejemplo donde el borrado afecta a varios documentos:

```
> db.clientes.find()
{ "_id" : ObjectId("5c0275f8f87ed69a71545d14"), "nombre" : "Pedro", "apellido1" : "León", "apellido2" : "Roldán" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec858"), "nombre" : "Lara", "edad" : 38, "sexo" : "mujer", "apellido1" : "Ramírez" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec859"), "nombre" : "Andrea", "apellido1" : "Lara", "apellido2" : "Sempere" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85a"), "nombre" : "Miguel", "apellido1" : "Cobos", "apellido2" : "Pascual" }
{ "_id" : ObjectId("5c02c396735bd96b3e9ec85b"), "nombre" : "Manuel", "apellido1" : "Beltrán", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02c7cd735bd96b3e9ec85c"), "nombre" : "Rosa", "apellido1" : "Rodríguez", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02cbd7735bd96b3e9ec85d"), "nombre" : "Sergio", "apellido1" : "Valiente", "apellido2" : "Sanz" }
{ "_id" : ObjectId("5c02e27a735bd96b3e9ec85f"), "nombre" : "Pilar", "apellido1" : "Valiente", "apellido2" : "Ruiz", "edad" : 30 }
```

```
> db.clientes.remove(
... {apellido2:'Sanz'}
... )
WriteResult({ "nRemoved" : 3 })
```

Hemos visto que

**`db.nombre_colección.find(criterios_búsqueda)`**

muestra toda la información almacenada de los documentos que responden a los criterios de búsqueda.

Pero vamos a ver distintas formas de indicarle los criterios según nuestra necesidad



# FIND

# mongodb

CRITERIOS BÚSQUEDA		EJEMPLO
Condición propiedad=valor	:valor	db.clientes.find({nombre:'Juan'})
Varias condiciones con AND	{cond1, cond2,...}	db.clientes.find({nombre:'Juan', edad:45})
Varias condiciones con OR	{ \$or: [...] }	db.clientes.find({ \$or:[ {nombre:'Andrea'}, {nombre:'Miguel'} ] })
Condición propiedad > valor	{\$gt: <i>valor</i> } greater than	db.clientes.find({edad :{\$gt: 20} })
Condición propiedad >= valor	{\$gte: <i>valor</i> } greater than   equal	db.clientes.find({edad :{\$gte: 20} })
Condición propiedad < valor	{\$lt: <i>valor</i> } lower than	db.clientes.find({edad :{\$lt: 30} })
Condición propiedad <= valor	{\$lte: <i>valor</i> } lower than   equal	db.clientes.find({edad :{\$lte: 30} })

# FIND

# mongoDB

CRITERIOS BÚSQUEDA		EJEMPLO
Condición valor1<=propiedad >= valor2	<code>{ \$gte: valor1 }, { \$lte: valor2 }</code>	<code>db.clientes.find({ edad : { \$gte: 20, \$lte: 30 } })</code>
Condición encontrar una subcadena	<code>{ \$regex: 'expresión' }</code>	<code>bd.clientes.fnd({ nombre: { regex: 'a' } })</code>
Condición EXISTS	<code>{ propiedad: { \$exists: true } }</code>	<code>db.clientes.find({ edad: { \$not: { \$exists: true } } })</code>
Condición NOT	<code>{ propiedad: { \$not: { condición } } }</code>	<code>db.clientes.find({ edad: { \$not: { \$gte: 30 } } })</code> <code>db.clientes.find({ edad: { \$not: { \$exists: true } } })</code>

Además se pueden mostrar documentos según los criterios de búsqueda pero añadiendo ciertas propiedades a mostrar:

**`db.nombre_colección.find(criterios_búsqueda, propiedades a mostrar)`**

`db.clientes.find({nombre:{$regex:'a'}},{edad:1})` →

De cada documento que en el nombre contenga la subcadena a, mostrará únicamente el identificador y la edad.

`db.clientes.find({nombre:{$regex:'a'}},{edad:0})` →

En este caso mostrará todas las propiedades salvo la edad.

# FIND()

# mongodb

ASPECTOS SOBRE LA RESPUESTA		EJEMPLO
Mejor presentación	<code>pretty()</code>	<code>db.clientes.find().pretty()</code>
Ordenación ascendente por una propiedad	<code>sort( {propiedad:1})</code>	<code>db.clientes.find({nombre:\$regex:'a'}).sort({nombre:1})</code>
Ordenación descendente por una propiedad	<code>sort( {propiedad: -1})</code>	<code>db.clientes.find({nombre:\$regex:'a'}).sort({nombre:-1})</code>
Limitar el número de documentos mostrados	<code>limit</code>	<code>db.clientes.find({nombre:\$regex:'a'}, {edad:1}).sort({nombre:-1}).limit(1)</code>

Se pueden agrupar datos con la función **.aggregate()** y el funcionamiento sería similar a un group by en una base de datos relacional.

```
db.micoleccion.aggregate(  
  {$match: {condiciones}},  
  {$group: {  
    _id: "$groupbyfield",  
    nombre_resultado: {función_agregada}} } } )
```

- \$sum: suma o incrementa
- \$avg : calcula la media del grupo
- \$min: mínimo de los valores del grupo
- \$max: máximo del grupo
- \$first: obtiene el primer elemento del grupo
- \$last: obtiene el último elemento del grupo

¿Cómo podemos ver cuántos clientes hay de cada edad?

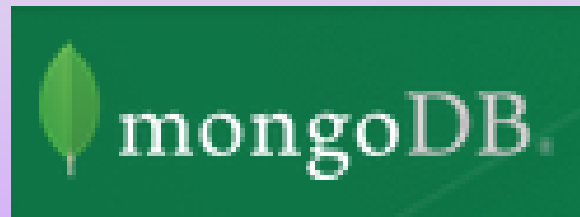
```
db.clientes.aggregate(  
  {$match:{}}, → Condiciones, en este caso no se especifican  
  {$group: {_id:"$edad", → Condiciones de agrupación  
    cuenta:{$sum:1} → Nombre resultado:  
  } → Función de agrupación  
});
```

Para borrar una base de datos se utiliza la función `.dropDatabase()`, pero antes hay que situarse en la base de datos que queremos borrar.

1. Nos situamos en la base de datos que queremos borrar  
**>use nombre\_bd**
2. Nos aseguramos de estar en la base de datos correcta  
**>db**
3. Borramos la base de datos  
**>db.dropDatabase()**

# MANUAL CONSULTA mongoDB

<https://docs.mongodb.com>







**DBD**

Diseño de  
Bases de Datos

**¿DUDAS,  
PREGUNTAS?**



Universitat d'Alacant  
Universidad de Alicante

**lsi** Departamento  
de Lenguajes  
y Sistemas  
Informáticos