MONGO DB:

Definición:

- **MongoDB** es una **base de datos noSQL de tipo documental**.
- Almacena la información en documentos **tipo BSON** (JSON Binario).
- Su nombre viene de la palabra inglesa "humongous" que significa enorme
- Carece de esquema predefinido.
- En mongoDB **una base de datos es una colección de documentos.** Estos documentos se componen de campos/características … de distintos tipos de datos.

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

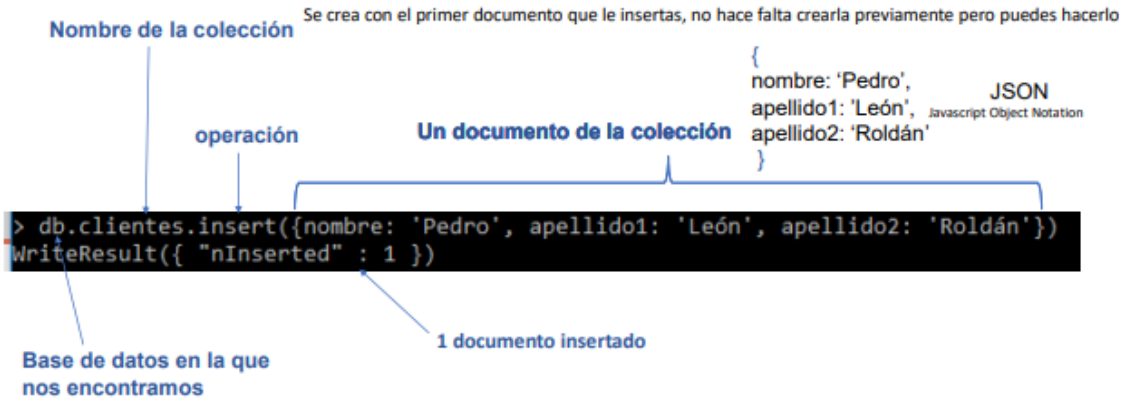
- **String:** guardados en UTF-8. Van siempre entre comillas dobles.
- **Number:** números. Al guardarse en BSON pueden ser de tipo byte, int32, int64 o double.
- **Boolean:** con valor true o false.
- **Array:** van entre corchetes [] y pueden contener de 1 a N elementos, que pueden ser de cualquiera 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.**

Comandos:

- **Iniciar servidor:** mongod
- **Abrir un terminal:** mongo
- **Mostrar base de datos creadas:** show dbs
- **Indicar la base de datos a utilizar:** use comercio
- **Muestra las colecciones creadas:** show collections
- **Muestra la base de datos en la que te encuentras:** db
- **Borrar una base de datos:**
  - Te sitúas en la base de datos: use nombre
  - Te aseguras si está en la base de datos correcta: bd
  - Ejecutar: bd.dropDataBase()

Insert:

La inserción de datos se realiza a través de colecciones.



Para insertar varios clientes de golpe, se utiliza un array de clientes ([]).

```
> 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" : [ ]
})
```

4 inserciones

Si quisiésemos asignar el identificador(único)

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

No es necesario que todos los clientes tengan los mismos campos.

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

```
> 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" }
>
```

Select:

Para ver la información almacenada en una colección

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

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

Para ver la información almacenada en una colección en un formato más legible

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

Ver clientes insertados: (También se puede poner **db.clientes.find().pretty()**)

```
> 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" }
>
```

Si queremos **buscar los clientes** que tengan un valor en una de las características.
(find({característica:valor}))

```
> 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" }
```

Si te equivocas al poner el nombre de una característica al insertar.

```
> db.clientes.insert({nombre: 'Sergio', apellido1: 'Valiente', apelido2: 'Sanz'});
WriteResult({ "nInserted" : 1 })
```

No aparece ya que apelido2 <> 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" }
```

Si quieres **buscar por identificador** hay que utilizar la función **Objectid().**

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

Update:

```
> db.clientes.update(
... {apelido2:'Sanz'},
... {
... nombre:'Sergio',
... apellido1:'Valiente',
... apellido2:'Sanz'
... }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

→ Características que debe cumplir el documento a modificar

Hemos cambiado el nombre de la característica apelido2 por apellido2

→ Nuevo contenido del documento

Ejemplo más claro:

```
> 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" }
>
```

Lara

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

O bien

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

CUIDADO, si **no quieres repetir la información** que no se modifica se puede poner el **identificador** y **$set(datos).**

```
> 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" }
```

Igual que $set:{datos} existe también:

- **$Inc:{datos}** = Solo suma o resta un valor de un campo.
- **$unset:{datos}** = Elimina un campo.

```
> 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
}
```

Con Update también se pueden insertar datos a la base de datos añadiendo **{upsert: true}:**

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

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


También puedes renombrar los campos creados con $rename:

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

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

```
> 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 }
```

Delete:

Para eliminar datos de la base de datos con la función **remove**.

```
> 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 }
```

```
> 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 }
```

Criterios y Aspectos:

db.nombre_colección.find(criterios_búsqueda) → **muestra toda la información almacenada de los documentos que responden a los criterios de búsqueda.**

Criterios de 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: *valor*} greater tan	db.clientes.find({edad :{$gt: 20} })
Condición propiedad >= valor	{$gte: *valor*} greater than | equal	db.clientes.find({edad :{$gte: 20} })
Condición propiedad < valor	{$lt: *valor*} lower than	db.clientes.find({edad :{$lt: 30} })
Condición propiedad <= valor	{$lte: *valor*} lower tan | equal	db.clientes.find({edad :{$lte: 30} })

Criterios de búsqueda		Ejemplo
Condición valor1<=propiedad >= valor2	*{$gte: valor1},{$lte:valor2}*	db.clientes.find({edad :{$gte: 20, $lte:30} })
Condición encontrar una subcadena	{$regex: *'expresión'*}	bd.clientes.find({ nombre: {regex: 'a'} })
Condición EXISTS	{propiedad:{$exists:true}}	db.clientes.find({edad:{$exists:true}})
Condición NOT	{propiedad:{$not:{condición}}}	db.clientes.find({edad:{$not:{$gte:30}}}) db.clientes.find({edad:{$not:{$exists:true}}})

db.nombre_colección.find(criterios_búsqueda, propiedades a mostrar)
→ muestra las propiedades que se indiquen de los documentos que responden a los criterios de búsqueda. (0 es no mostrar)

db.clientes.find({nombre:{$regex:'a'}},{edad:1}) ⟶ De cada documento muestra únicamente el identificador y la edad.

db.clientes.find({nombre:{$regex:'a'}},{edad:0}) ⟶ De cada documento muestra todas las propiedades salvo la edad.

Aspectos sobre la respuesta		Ejemplo
Mejor presentación	pretty()	db.clientes.find().pretty()
Ordenación ascendente por una propiedad	sort({propiedad:1})	db.clientes.find({nombre:{$regex:'a'}}).sort({nombre:1})
Ordenación descendente por una propiedad	sort({propiedad: -1})	db.clientes.find({nombre:{$regex:'a'}}).sort({nombre:-1})
limitar el número de documentos mostrados	limit	db.clientes.find({nombre:{$regex:'a'}},{edad:1}).sort({nombre:-1}).limit(1)

Agregados:

Agregación (group by)

```
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

Ejemplo: ¿Cuántos clientes hay de cada edad?

```
db.clientes.aggregate({$match:{}},{$group:{_id:"$edad",cuenta:{$sum:1}}})
```

NOSQL:

NOSQL:

El **término NoSQL** (aunque ya se había utilizado antes) toma impulso con la aparición de la web 2.0, con la llegada de Facebook, Twitter o YouTube, donde cualquier usuario podía subir contenido, provocando así un crecimiento exponencial de los datos.

Las bases de datos relacionales no sirven para manejar este volumen de datos. Los sistemas tienen que **escalar** para poder ser más grandes y más rápidos.

Escalado:

- **Vertical:** Es caro, tiene un límite y las regulaciones de ciertos países no te permiten guardar los datos fuera del país, por lo que a veces no puedes hacerlo.
- **Horizontal:** hay que distribuir, **replicación y sharding.**

En el escalado Horizontal es **complicado en BD Relacionales.**

Replicación (todos los nodos deben guardar la misma información)

Distribuir datos en BD relacionales es complicado, hay que mantener las propiedades ACID: hay que hacer un commit de 2 fases, hay que esperar a que todos los nodos de la red contesten. Unas transacciones se quedan bloqueadas esperando a que otras terminen.

Sharding (los nodos tienen datos distintos)

También es difícil llevarlo a la práctica en las bases de datos relacionales

BD Relacionales:

Las BD Relacionales **cumplen** las **propiedades ACID**.

Es un conjunto de órdenes que se ejecutan sobre una base de datos constituyendo una unidad lógica de trabajo.

Si la transacción termina bien, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si en una transacción se producen errores, debe cancelarse completamente todas las acciones llevadas a cabo por la transacción.

Propiedades de las transacciones

- **Atomicidad:** una transacción es indivisible. O se efectúan todas las operaciones o debe quedar la BD como si no se hubiese efectuado ninguna.
- **Consistencia:** después de ejecutarse una transacción, la BD debe quedar en un estado correcto (sin necesidad de que sea correcto entre operaciones de una transacción).
- **Aislamiento (Isolation):** el comportamiento de una transacción no se ve afectado porque otras transacciones se estén ejecutando a la vez.
- **Durabilidad:** los efectos de una transacción son permanentes tras su grabación.

Otros Sistemas de Bases de Datos:

Estas bases de datos tienen varias características en común, que las diferencian de las bases de datos relacionales: -

- **Arquitectura distribuida:** pensadas para trabajar en múltiples servidores que se comunican entre sí.
- **Pueden manejar gran cantidad de datos.**
- **No genera cuellos de botella:** los sistemas relacionales necesitan transcribir cada sentencia para poder ser ejecutada, y la ejecución de muchas sentencias complejas puede ralentizar el sistema.
- **Flexibilidad de esquema:** a diferencia de las relacionales donde el esquema de la base de datos es fijo, estático, en estas bases de datos existe el esquema no es fijo, es un esquema dinámico.
- **No ACID** → eventualmente consistentes (**BASE**)

BASE:

BASE (Basically Available Soft-state Eventual Consistency)

NoSQL no garantiza las propiedades ACID, permite la Consistencia Eventual.

En un sistema relacional el commit de una transacción EXIGE que este cambio se replique de forma inmediata en todos los nodos (sistemas transaccionales). Esto no es así en los sistemas NoSQL, esto se conoce también como BASE (coherencia eventual flexible básicamente disponible).

El principio **BASE facilita enormemente el escalado horizontal**.

El principio **BASE permite** que las **operaciones** sean **mucho más rápidas** que los sistemas que garantizan las propiedades ACID.

¿Voy a usar una base de datos NoSQL si mis datos pueden no ser consistentes?

Depende, si es un **sistema bancario o** si es una **red social**.

Tipos de Bases de Datos NOSQL

- **Clave-Valor**

Almacena cualquier tipo de objeto (número, cadena de caracteres, array, json,…) y accede a estos objetos por un identificador.

Clave	Valor
23567	346
23568	{depart1:"Lenguajes y Sistemas, depart2:"Física Aplicada",depart3:"Ciencias"}
23590	45.22
23591	"El Quijote"
23596	"Vaya usted a saber"

- Muy rápida la búsqueda por clave ya que se crea un índice de tipo hash para la clave.

- La búsqueda por valores de la base de datos no la puede realizar. No hay un patrón en este contenido. Hay que ir recuperando todos los contenidos y entonces ir analizándolos.

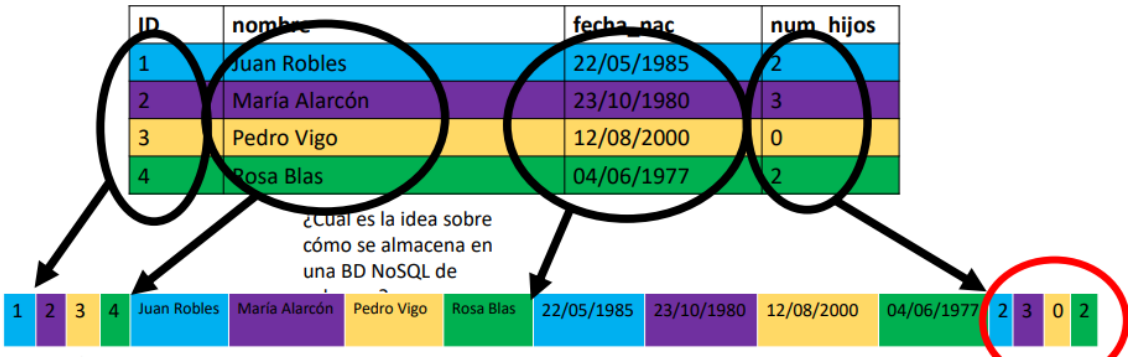
Ejemplo: DynamoDB (Amazon)

- **Columnas**

Es como si pusiéramos lo que serían las filas en columnas.

- Se almacena por filas.

- Cada lectura recupera una fila completa



- ¿Dónde viene bien esta estructura? En BD analíticas

- Desventajas: se ralentizan las operaciones de borrado y modificación

Ejemplo de BD columna: BigTable (Google)

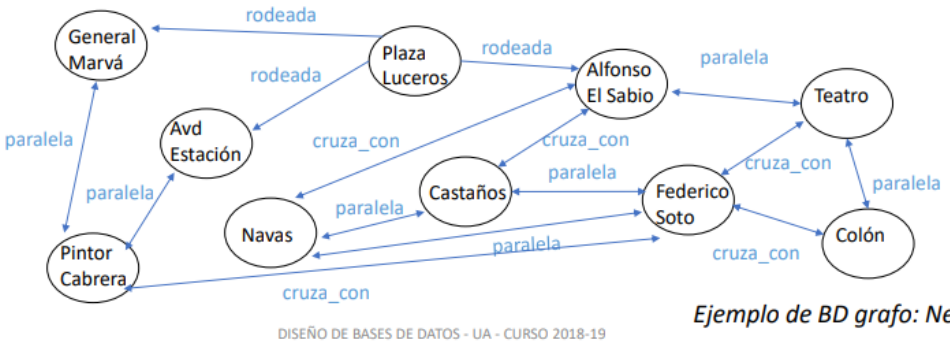
- **Grafos**

La información se representa como nodos de un grado y sus relaciones son las aristas del mismo.

Este tipo de bases de datos es muy útil para información muy interconectada entre sí y donde esta conexión es tan importante como la información en sí.

Son muy muy eficientes para ir pasando de un nodo a otro

Se usan en casos muy específicos: redes sociales, tráfico aéreo, distribución eléctrica…



Ejemplo de BD grafo: Neo4j

- **Documentales**

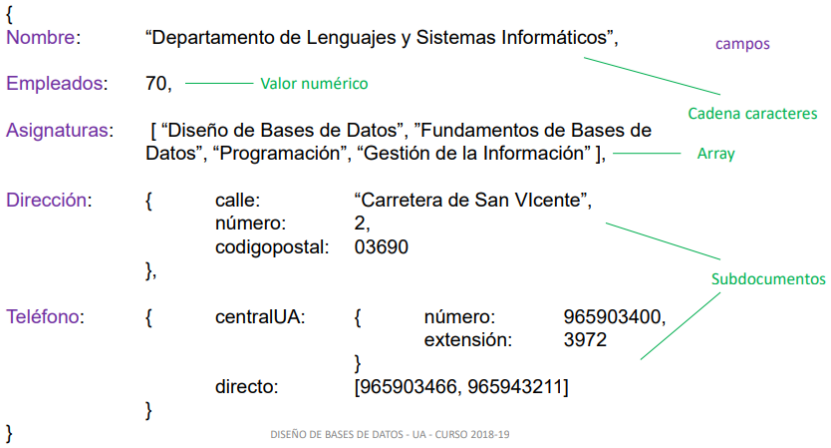
NO se refiere a una BD que guarda pdf's o documentos Word, …

Se refiere a una BD que guarda estructuras que a su vez pueden tener muchos niveles de información.

Para cada entrada no tenemos las típicas columnas de estructura fija, cada entrada tendrá su propia estructura.

Cada una de estas estructuras que contiene información desestructurada se llaman documento.

Muchas bases de datos documentales representan estas estructuras como objetos JSON.



Ya que de los sistemas de bases de datos noSQL, el **más utilizado es MongoDB**, será el que veamos un poco más en profundidad.

Seguridad en BD:

Objetivos

- **Confidencialidad**: sólo los usuarios autorizados pueden tener información a la información que les corresponde y con los permisos que les corresponde.
- **Integridad:** asegurar que lo que los usuarios tratan de hacer es correcto y evitar la pérdida accidental de la consistencia.
- **Disponibilidad:** asegurarse que la información estará disponible cuando sea necesaria.

¿Por qué protegernos?

- **Importancia estratégica de la información:** La información es un bien muy valioso. El 40% de las compañías que pierden completamente su sistema informático desaparecen.
- **Evitar robos y/o sabotajes.**
- **Un robo o pérdida** tiene un coste de imagen elevado.
- Y además … **LEY DE PROTECCIÓN DE DATOS**.

¿De quién protegernos?

- **De agentes externos:**
  - Control de acceso a la BD
- **De agentes internos: usuarios autorizados**. Se debe controlar:
  - qué usuarios tienen acceso a qué datos
  - qué operaciones pueden realizar sobre dichos datos
  - que no se permitan operaciones que vulneren la integridad de los datos
- **De catástrofes y fallos:**
  - Política de copias de seguridad

¿Cómo protegernos?

- **De agentes externos:**
  - **A nivel de red:** Sólo acceden al servidor los ordenadores autorizados.
  - **A nivel de SO:** Sólo los usuarios con privilegios acceden a los ficheros de la BD.
  - **A nivel de BD:** Política de contraseñas.

- **De usuarios conocidos:**

  - **Controlar acceso a datos.**
    - **Seguridad por niveles:**
      - Asigna a cada usuario y cada objeto (tabla, fila, columna, vista…) se le asigna un nivel.
      - Control de acceso:
        - ✓ un sujeto S puede leer el objeto O si nivel(S) ≥ nivel(O).
        - ✓ un sujeto S puede escribir el objeto O si nivel(S) = nivel(O).

    - **Seguridad por privilegios.**
      - Privilegio: derecho a ejecutar un tipo de SQL o de acceder a objetos de otros usuarios
      - Tipos de privilegios:
        - ✓ de sistema: permite ejecutar cierto tipo de acciones como crear tablas, índices, usuarios, procedimientos…
        - ✓ de objetos: permite ejecutar ciertas acciones sobre objetos específicos (hacer un insert en una tabla, etc.) Se permite poner privilegios de objetos incluso a nivel de campos (un usuario puede hacer un update sobre unos campos, pero no sobre otros).

  - **La mayoría de las BD permiten la gestión de privilegios por ROLES para facilitar la administración.**
    - Roles: Agrupaciones de privilegios.
    - A un usuario se le puede otorgar un privilegio concreto o todos los incluidos en un role (mejora administración).

Sin ROLES: Usuarios con mismos privilegios hay que definir los permisos para cada uno Con roles: Usuarios con mismos privilegios tienen mismo role.

Con ROLES: Usuarios con mismos privilegios tienen mismo role.

  - **Para asegurar integridad en datos.**
    - **Constraints:** son propiedades de la base de datos que se deben satisfacer en cualquier momento. Si la constraint está activa es porque se cumple la restricción que define.
      - Tratamiento de valores nulos.
      - Valores por defecto.
      - Integridad de clave primaria.
      - Claves alternativas.
      - Integridad referencial.
      - Restricciones de integridad estáticas (check).

    - **Disparadores:** NO GARANTIZAN la integridad ya que sólo actúan cuando están activos y al activarse NO garantizan que cumpla aquello que controlan.

    - **Control de transacciones, principio ACID.**

- **Control de Transacciones:**

Una **TRANSACCIÓN** es una unidad lógica de trabajo. Sin conservar, por fuerza, la consistencia en los puntos intermedios.

- **ATOMICIDAD:** todas las operaciones que la componen son tomadas como u todo. O todas las acciones de la transacción se realizan de forma válida o ninguna de ellas se considera correcta.
- **CONSISTENCIA:** cualquier transacción llevará a la base de datos desde un estado válido a otro también válido, La integridad debe quedar garantizada.
- **DURABILIDAD:** una vez que se ha validado la transacción, no se debe perder.
- **AISLAMIENTO:** una transacción en ejecución no puede revelar sus resultados a otras transacciones concurrentes antes de finalizar.

- **Control de concurrencia.**
  - El control ACID lo hace el propio SGBD.
  - Hay que llevar especial cuidado en lenguajes como php, .net, jsp., etc, ya que por defecto llevan "commit implícito (después de cada instrucción se hace un commit)", es decir NO hay control de transacciones a no ser que el programador lo especifique. El programador solo debe definir el inicio y el final de una transacción.
  - Cualquier error o fallo en alguna de las instrucciones produce un rollback automático.

- **Catástrofes y fallos.**

  - Se debe **garantizar** la **Recuperación ante fallos** de los **usuarios** (por ejemplo, borrado accidental de información) y ante fallos del **sistema** (por ejemplo, rotura de un disco). Pare ello se dispone de los mecanismos de copias de seguridad.

  - Se debe de **garantizar** la **recuperación** ante **grandes catástrofes**, para ello se dispone de los centros de respaldo.

- **De errores y fallos: Copias de seguridad**

  - Las **copias de seguridad** permiten **recuperar la base de datos**, pero sólo la información que había en la misma hasta el momento de hacer la copia.

  - Los SGBD emplean **LOG DE TRANSACCIONES** (en ORACLE modo ARCHIVELOG) para garantizar la recuperación total.

- **Caso de ORACLE: uso de los REDO LOG en MODO ARCHIVER**.

  - En ORACLE por defecto la BD se crea en modo NOARCHIVELOG (con CREATE DATABASE).

  - Si activamos el modo ARCHIVELOG se irán archivando los ficheros redo log conforme se llenan (cada vez que ocurre un "log switch").

  - Si disponemos de ARCHIVE LOG, en caso de caída, podemos recuperar la BD hasta el instante mismo de la caída aplicando los REDO LOGS archivados a la última copia de seguridad (realizada con RMAN)

¿Algo más? Sí auditar.

Auditar, **registrar de forma automática** los:

- **Accesos a la BD**
- **Accesos y operaciones sobre los objetos de la BD.**

El tener procesos de auditoría en marcha pueden afectar al rendimiento de la BD

¿Cómo auditar? Depende del SGBD, en ORACLE:

- **Con disparadores**
- **Con utilidad AUDIT**

Utilidad AUDIT de ORACLE.

- Hay que configurarla
- Tiene muchas opciones y configuraciones posibles
- La información de la auditoría puede quedar en tablas propias de la BD o en ficheros del S.O.

Varios niveles de auditoría

- **De sentencias.** Seleccionando un tipo concreto de las mismas, que afectan a una determinada clase de objetos de base de datos.
- **De privilegios.** Auditoría de privilegios de sistema.
- **De esquema.** Sentencias específicas sobre objetos de un esquema concreto. Afectan a TODOS los usuarios de la BD.
- **De grano fino ("fine grained").** Acceso a datos concretos y cambios en los mismos a nivel columna. Se usa el paquete DBMS_FGA y sus procedimientos asociados, generándose apuntes en el "audit trail" de grano fino.