

DAW/DAM. UD 8. BASES DE DATOS NOSQL PARTE 1. MONGODB: DDL Y DQL

DAW/DAM. Bases de datos (BD)

UD 8. BASES DE DATOS NOSQL

Parte 1. MongoDB: DDL y DQL

Abelardo Martínez y Pau Miñana

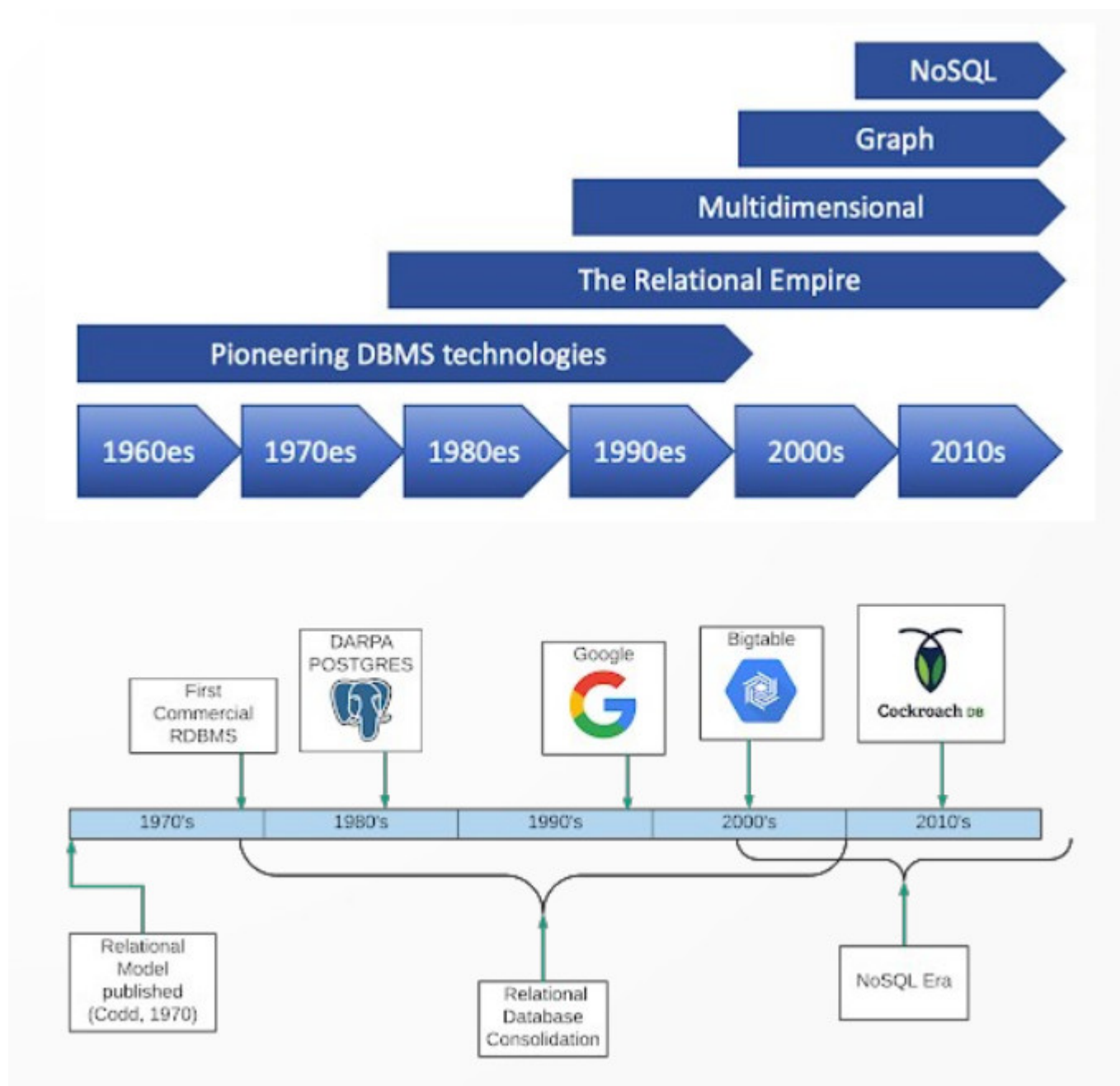
Basado y modificado de Sergio Badal (www.sergiobadal.com) y Raquel Torres.

Curso 2023-2024

1. BD no relacionales (NoSQL)

Las bases de datos han ido evolucionando a lo largo del tiempo, desde los viejos sistemas jerárquicos y en red, hasta los más sofisticados SGBD relacionales. No obstante, a partir de la primera década de los 2000, apareció un nuevo paradigma de bases de datos no relaciones. Con el tiempo, este tipo de bases de datos han ido ampliando su potencial y expandiéndose, de manera que representan una alternativa válida a las clásicas relacionales.

En esta gráfica podemos ver la evolución de las bases de datos:



1.1. ¿Por qué NO-SQL?

NoSQL - “Not Only SQL”

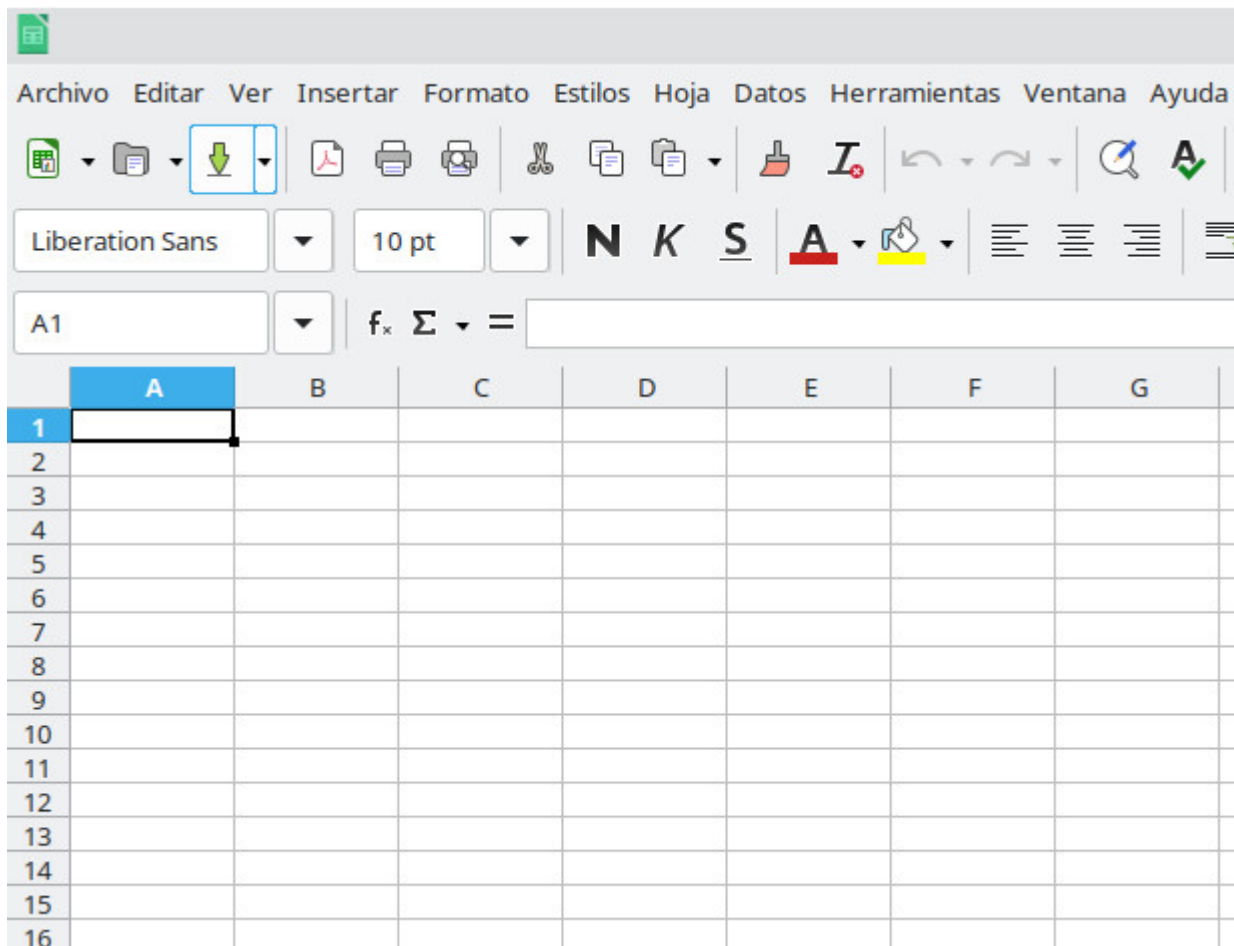
Es una nueva categoría de bases de datos no-relacionales y altamente distribuidas.

Piensa en algún familiar o conocido con pocos o nulos conocimientos de Informática, dale mucha información de muchas fuentes diferentes, un portátil, abre una hoja de cálculo y un documento de procesador de textos, y observa con cuál se queda. Puede ser un poco básico este ejemplo, pero es una buena manera de comprender la principal diferencia entre bases de datos relacionales (BDR o SQL) y no relacionales (BDNR o NoSQL).

¿Cuál crees que escogería?

a) Hoja de cálculo

Tal vez prefiera la hoja de cálculo. ¿Por qué? Porque encaja muy bien en filas y columnas.



Una base de datos relacional es, en gran medida, un conjunto de hojas de cálculo perfectamente conectadas y estandarizadas. Es aquella que almacena datos en tablas, la relación entre cada dato es clara y la búsqueda a través de esas relaciones es relativamente fácil.

La relación entre las tablas y los tipos de campos se denomina esquema y, para las bases de datos relacionales, el esquema debe estar claramente definido. Por ejemplo:



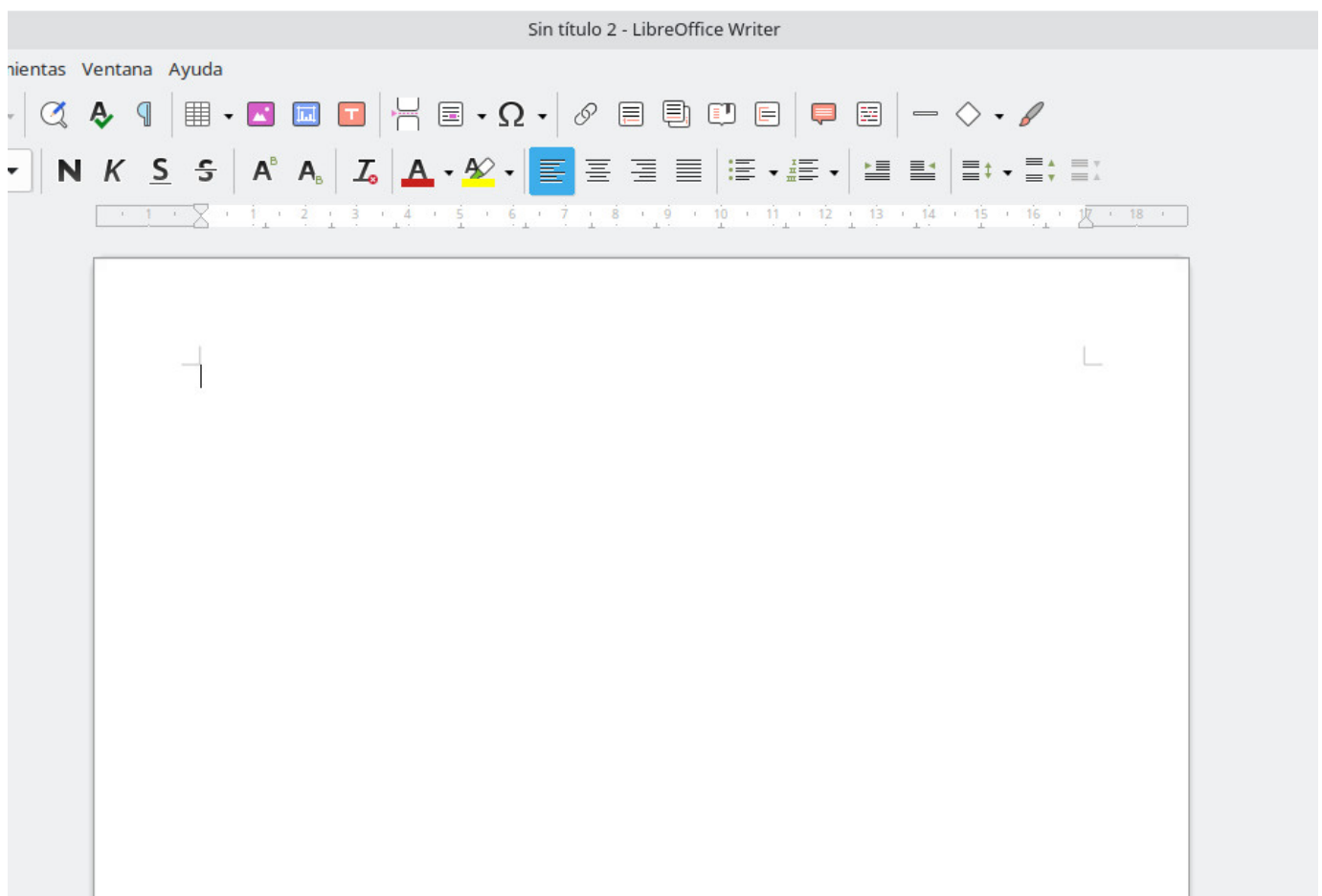
Las BDR también se denominan bases de datos SQL, donde SQL es realmente el lenguaje de consulta estructurado y el idioma que usamos para comunicarnos con las bases de datos relacionales. SQL se utiliza para ejecutar consultas, recuperar datos y editar datos actualizando, eliminando o creando nuevos registros.

Las BDR son las más utilizadas en la actualidad y las que hemos visto hasta ahora. En el siguiente enlace podemos ver la lista de las más usadas:

<https://db-engines.com/en/ranking>

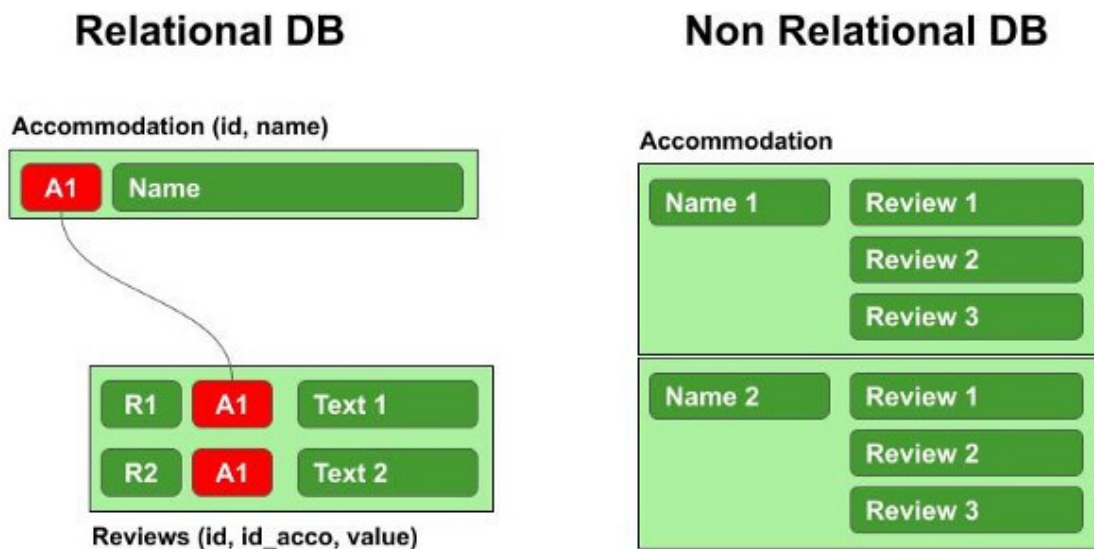
b) Procesador de textos

Volvamos al ejemplo inicial. Esta vez escogeremos el documento de procesador de texto. ¿Por qué? ¡Todo el espacio está en blanco! Los datos vienen en todas formas y tamaños... ¡libertad total!



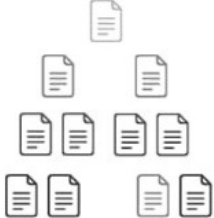
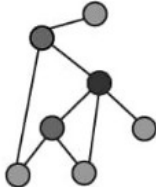
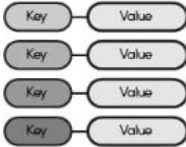

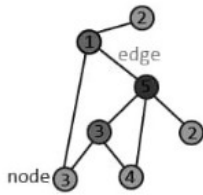
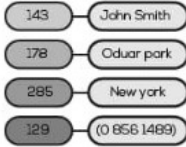
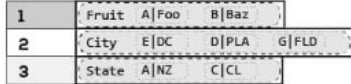




1.2. Estructura y tipos

Una base de datos no-relacional es cualquier base de datos que no utiliza el esquema tabular de filas y columnas como en las BDR. Más bien, su modelo de almacenamiento está optimizado para el tipo de datos que almacena y es muy frecuente almacenar todos los datos que tienen algo en común en el mismo archivo o en la misma unidad de información (el **documento** o **nodo**).



Los diferentes tipos de NoSQL son los siguientes:

NoSQL DATABASE TYPES

Document	Graph	Key-Value	Wide-Column
			
<pre>{ "user": { "id": "143", "name": "improgrammer", "city": "New York" } }</pre>			
			

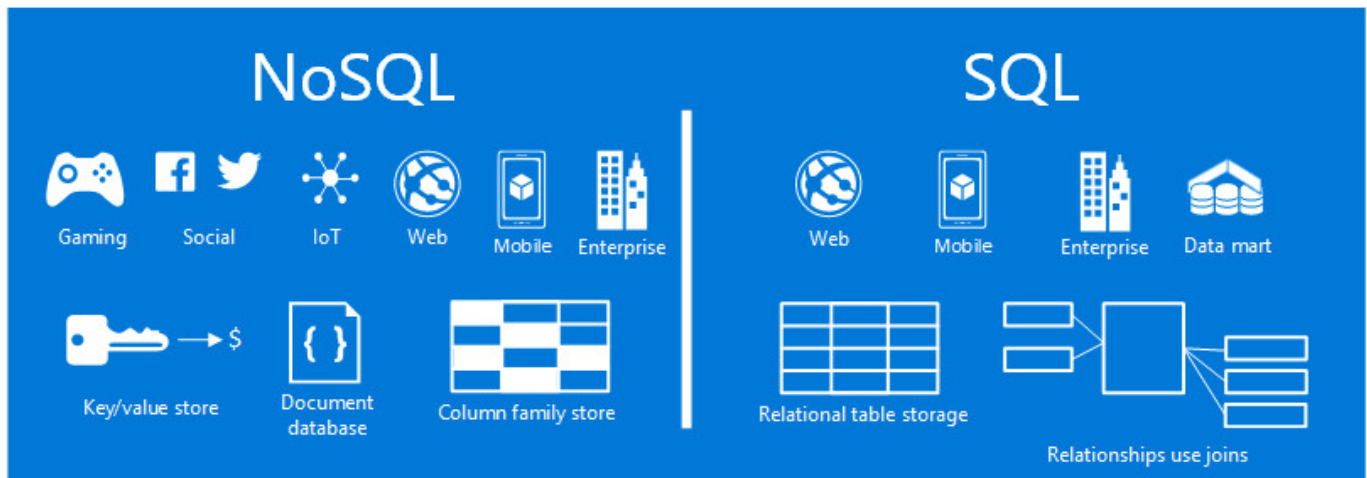
1.3. Diferencias y usos

En esta infografía se muestran las principales diferencias entre las bases de datos relacionales y las NoSQL:



Pese a que en principio puede parecer que son excluyentes, lo cierto es que **las bases de datos relacionales y las NoSQL se complementan bastante bien** y tienen su utilidad

mayor o menor, dependiendo del uso que les vayamos a dar. En este diagrama tenemos un resumen:



1.4. Teorema CAP

En Ingeniería del Software se usan mucho los triángulos para explicar conceptos complejos. En términos de desarrollo del software se recurre al **triángulo de hierro** para explicar que es utópico conseguir los tres lados del triángulo:



El triángulo de hierro representa la **imposibilidad de conseguir alcance, coste y tiempo sin dañar la calidad del software**, entendiendo el alcance como el conjunto de características del proyecto. Veamos como ejemplo el proyecto de creación de un *E-commerce*:

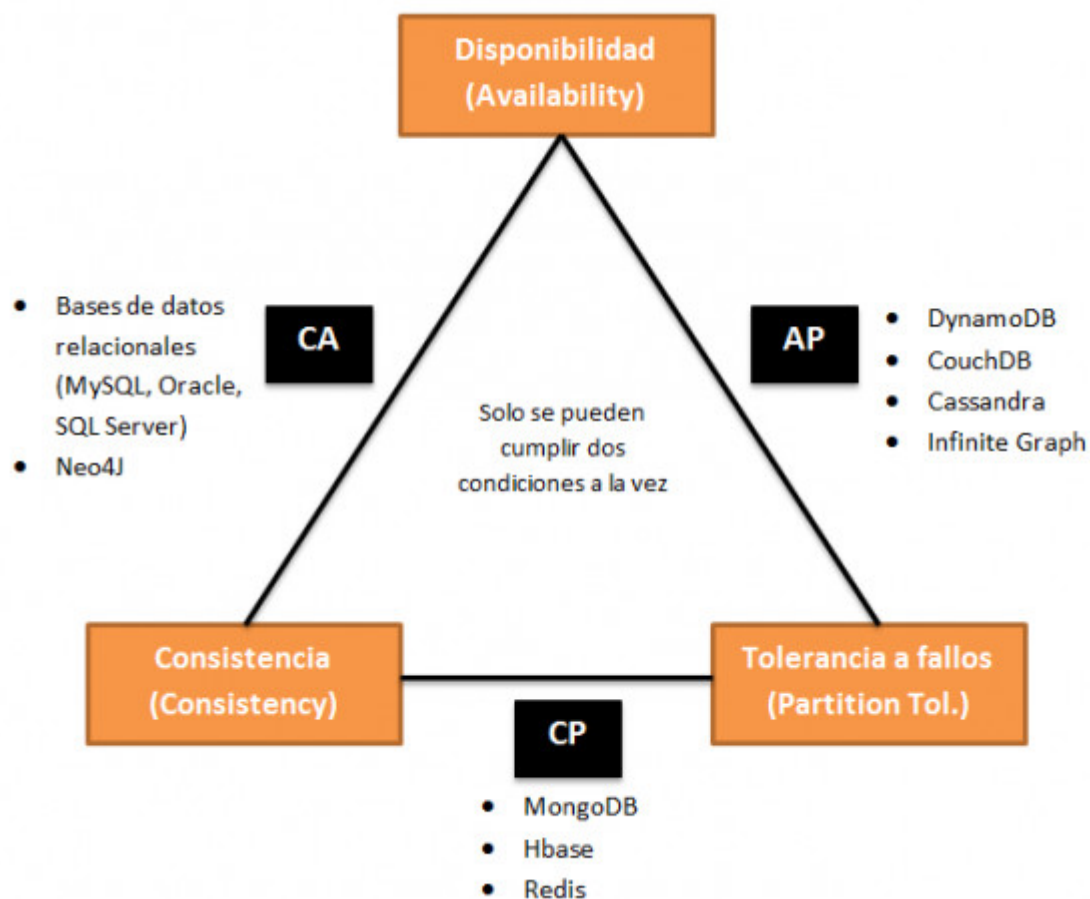
- **ALCANCE.** *E-commerce* para vender zapatos *online*, con esta lista de requisitos [...]
- **TIEMPO.** Entrega del proyecto completo en 6 meses desde la firma del contrato.
- **COSTE.** El presupuesto final será de 21.450 euros más IVA.

La solución que ha elegido la Ingeniería del Software para este “problema” son las metodologías ágiles (como SCRUM) que sacrifican uno de los tres vértices del triángulo para no dañar la calidad del software. En este ejemplo, si usamos metodologías ágiles, le daremos al cliente a escoger entre:

- **SACRIFICAR EL ALCANCE.** Entregaremos el proyecto en “tiempo y coste”, es decir, en la fecha y con el coste acordado, pero no nos comprometemos a entregar todos los requisitos (habrá que negociarlos).
- **SACRIFICAR EL TIEMPO.** Entregaremos el proyecto en “coste y forma”, es decir, con el coste acordado y el listado de requisitos que firmamos, pero con una fecha más flexible (habrá que negociarla).
- **SACRIFICAR EL COSTE.** Entregaremos el proyecto en “tiempo y forma”, es decir, en el tiempo acordado y el listado de requisitos que firmamos, pero con un coste más flexible

(habrá que negociarlo).

En términos de diseño y gestión de bases de datos se recurre al **triángulo CAP** para explicar que es utópico conseguir los tres lados del triángulo:



- **CONSISTENCY (Consistencia)**. Los datos que vemos al consultar la base de datos están 100% actualizados.
- **AVAILABILITY (Disponibilidad)**. La base de datos siempre está disponible, sin fallos, ni cortes de conexión ni excesivos retardos en las peticiones.
- **PARTITION TOLERANCE (Tolerancia a fallos)**. El sistema funcionará perfectamente en sistemas distribuidos en los que una misma BD puede estar almacenada en varios servidores de manera particionada.

La solución que se ha optado para este “problema” son **las bases de datos no relacionales** (como MongoDB) que **sacrifican uno de los tres vértices del triángulo para ser más realistas**. En este ejemplo, si usamos bases de datos no relacionales, le daremos al cliente a escoger entre:

- SACRIFICAR LA TOLERANCIA: No podemos garantizar que el sistema funcione correctamente de manera distribuida (ej: MySQL).
- SACRIFICAR LA CONSISTENCIA: No podemos garantizar que el sistema devuelva los datos 100% actualizados (ej: Cassandra, usado en Facebook).
- SACRIFICAMOS LA DISPONIBILIDAD: No podemos garantizar que el sistema esté siempre disponible (ej: MongoDB, usado en Expedia) pero cuando lo esté, los datos serán 100% actualizados.

2. Introducción a MongoDB

Dentro de las bases de datos NoSQL, probablemente una de las más famosas sea **MongoDB**. Con un concepto muy diferente al de las BDR, se está convirtiendo en una interesante alternativa. Su nombre deriva de la palabra inglesa “humongous”, que en lenguaje coloquial significa inmenso/a, enorme.

Cuando uno se inicia en MongoDB se puede sentir perdido. No tenemos tablas, no tenemos registros y, lo que es más importante, no tenemos SQL. Aun así, MongoDB es una seria candidata para almacenar los datos de nuestras aplicaciones.

La lista de organizaciones que utiliza MongoDB es impresionante: Foursquare, LinkedIn, empresas de telecomunicaciones como Orange y Telefónica, Cisco, Bosch, plataformas de formación como Codecademy, eBay, Expedia, etc.

Vamos a explicar cómo funciona esta base de datos NoSQL, qué podemos hacer con ella y cómo podemos hacerlo.

2.1. ¿Qué es MongoDB?

Como hemos visto anteriormente, hay muchos tipos de BBDD no relacionales (NoSQL). **MongoDB es una base de datos orientada a documentos**. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en **DOCUMENTOS**, que son almacenados como **BSON (es una representación binaria de JSON)** y que, a su vez, se agrupan o clasifican en **COLECCIONES**.

Por tanto, los **DOCUMENTOS** de una misma **COLECCIÓN** de una base de datos no relacional (o NoSQL), podemos entenderlos como los **REGISTROS/FILAS** de una **TABLA** de una base de datos relacional (o SQL).



Una de las diferencias más importantes con respecto a las BDR, es que **no es necesario seguir un esquema (unas normas estrictas), pudiendo tener cada documento un esquema diferente**.

Ejemplo

Imaginemos que tenemos una **COLECCIÓN** a la que llamamos Personas. Un documento podría almacenarse de la siguiente manera:

```
{
  Nombre: "Luis",
  Estudios: "Administración y Dirección de Empresas",
  Amigos: 12
}
```

El documento anterior es un clásico documento JSON. Tiene strings, arrays, subdocumentos y números. En la misma **COLECCIÓN** podríamos guardar un documento como éste:

```
{
  Nombre: "Pedro",
  Apellidos: "Martínez Campo",
  Edad: 22,
  Aficiones: ["fútbol", "tenis", "ciclismo"],
  Amigos: [
    {
      Nombre: "María",
      Edad: 22
    },
    {
      Nombre: "Luis",
      Edad: 28
    }
  ]
}
```

El primer documento no sigue el mismo esquema que el segundo. Tiene menos campos, algún campo nuevo que no existe en el documento anterior e incluso un campo de distinto tipo.

Esto que es algo impensable en una BDR es algo totalmente válido en MongoDB.
¿Entiendes ahora el ejemplo de la hoja de cálculo y el procesador de texto?

2.2. ¿Cómo funciona MongoDB?

Como curiosidad, comentarte que **MongoDB está escrito en C++**, aunque las consultas se hacen pasando objetos JSON como parámetro (usando, generalmente, **JavaScript**). Es algo bastante lógico, dado que los propios documentos JSON se almacenan internamente como BSON (¡en binario!) y son codificados/descodificados usando C++.

Por ejemplo, la siguiente consulta buscará todos los clientes cuyo nombre sea Pedro:

```
db.clientes.find({Nombre:"Pedro"})
```

La consulta se ejecutará como un comando en JavaScript, que buscará entre todos los DOCUMENTOS JSON de la COLECCIÓN "clientes", descodificando cada documento de BSON a JSON en C++.

MongoDB viene de serie con una consola desde la que podemos ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones de MongoDB, podemos utilizar muchas de las funciones propias de JavaScript. En la consola también podemos definir variables, funciones o utilizar bucles.

Si no nos gusta JavaScript y queremos usar nuestro lenguaje de programación favorito, existen *drivers* para un gran número de ellos. Hay *drivers* oficiales para C#, Java, Node.js, PHP, Python, Ruby, C, C++, Perl o Scala. Aunque estos *drivers* están soportados por MongoDB, no todos están en el mismo estado de madurez.

2.3. ¿Dónde se puede utilizar MongoDB?

Aunque se suele decir que las bases de datos NoSQL tienen un ámbito de aplicación reducido, MongoDB se puede utilizar en muchos de los proyectos que desarrollamos en la actualidad. **Cualquier aplicación que necesite almacenar datos semi-estructurados (sin un esquema fijo) puede usar MongoDB.** Es el caso de las típicas aplicaciones CRUD (create, read, update, delete) o de muchos de los desarrollos web actuales.

Eso sí, aunque las colecciones de MongoDB no necesitan definir un esquema, es importante que diseñemos nuestra aplicación para seguir uno. Tendremos que pensar si necesitamos normalizar los datos, desnormalizarlos o utilizar una aproximación híbrida. Estas decisiones pueden afectar al rendimiento de nuestra aplicación. En definitiva, el esquema lo definen las consultas que vayamos a realizar con más frecuencia.

MongoDB es especialmente útil en entornos que requieran escalabilidad, entendida como la posibilidad de crecer de varios miles de registros a varios millones. Con sus opciones de replicación y *sharding* (partición de una misma colección o base de datos en varios equipos), que son muy sencillas de configurar, podemos conseguir un sistema que escale horizontalmente sin demasiados problemas.

Escalar horizontalmente consiste en replicar la información en varios servidores de similares características mientras que **escalar verticalmente** consiste en ampliar la capacidad de los servidores, sin adquirir nuevos.

2.4. ¿Dónde no se debe utilizar MongoDB?

En esta base de datos no existen las transacciones. Aunque nuestra aplicación puede utilizar alguna técnica para simular las transacciones, MongoDB no tiene esta capacidad. **Solo garantiza operaciones atómicas a nivel de documento.** Si las transacciones son algo indispensable en nuestro desarrollo, deberemos pensar en otro sistema.

Tampoco existen los JOINS. Para consultar datos relacionados en dos o más colecciones, tenemos que hacer más de una consulta. En general, **si nuestros datos pueden ser estructurados en tablas, y necesitamos las relaciones, es mejor que optemos por una BDR clásica.**

Y para finalizar, están las **consultas de agregación**. MongoDB tiene un *framework* para realizar consultas de este tipo llamado Aggregation Framework, aunque también puede usar Map Reduce. Aún así, **estos métodos no llegan a la potencia de un sistema relacional.** Si vamos a necesitar explotar informes complejos, deberemos pensar en utilizar otro sistema, si bien esta es una brecha que MongoDB va recortando con cada versión y en poco tiempo esto podría dejar de ser un problema.

2.5. ¿Cómo se instala MongoDB?

La instalación de una instancia del servidor es muy sencilla; simplemente tenemos que bajar los binarios para nuestro sistema operativo. Hay versiones para Linux, Windows y MacOS.

- Cómo instalar MongoDB en Windows, Linux y Mac. <https://platzi.com/blog/como-instalar-mongodb-en-window-linux-y-mac/>
- Cómo Instalar MongoDB Community Edition en Ubuntu. <https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/>
- Cómo Instalar MongoDB paso a paso en Windows. <https://www.youtube.com/watch?v=gkCnXcxHC4o>

¿Cómo usar la consola?

Si ya tenemos el servidor lanzado en nuestra máquina, bastará con lanzar desde la consola el comando adecuado según el sistema operativo. Desde ese momento entraremos en la consola y podremos realizar consultas, de manera muy similar a como lo hacíamos con MySQL.

2.6. Comandos básicos

En Mongo, **NO ACABAREMOS LOS COMANDOS CON ;**

Podemos ver la ayuda con **help** [enter] y podemos salir de mongo con **exit** [enter]

3. Estructura de un documento

Así como la unidad mínima de información en una BDR era la celda, en MongoDB es el documento JSON, pudiendo ser éste todo lo complejo que queramos. Por ejemplo:

Archivo colores1.json	Archivo colores2.json	Archivo colores3.json
<pre>{ "arrayColores":[{ "nombreColor":"rojo", "valorHexadec":"#f00" }, { "nombreColor":"verde", "valorHexadec":"#0f0" }, { "nombreColor":"azul", "valorHexadec":"#00f" }, { "nombreColor":"cyan", "valorHexadec":"#0ff" }, { "nombreColor":"magenta", "valorHexadec":"#f0f" }, { "nombreColor":"amarillo", "valorHexadec":"#ff0" }, { "nombreColor":"negro", "valorHexadec":"#000" }] }</pre>	<pre>{ "arrayColores":[{ "rojo":"#f00", "verde":"#0f0", "azul":"#00f", "cyan":"#0ff", "magenta":"#f0f", "amarillo":"#ff0", "negro":"#000" }] }</pre>	<pre>{ "rojo":"#f00", "verde":"#0f0", "azul":"#00f", "cyan":"#0ff", "magenta":"#f0f", "amarillo":"#ff0", "negro":"#000" }</pre>

Para más información:

- JSON básico. <https://desarrolloweb.com/home/json>
- Aprende JSON en 30 minutos. <https://www.youtube.com/watch?v=qQjALhiEM3A>

4. Lenguaje de definición de datos (DDL)

Comandos para crear y manipular objetos (de ahora en adelante, asumimos un [enter] al final de cada comando):

Operación	Descripción	Comando
Mostrar todas	Mostrar las bases de datos	show databases show dbs
Conectar	Conectarnos a una base de datos concreta.	El comando “use” si no existe la bd, la crea: use mibasededatos
Mostrar actual	Podemos ver en qué base de datos estamos.	db
Borrar	Podemos borrar la BD a la que estamos conectados.	db.dropDatabase()
Mostrar colecciones	Podemos mostrar las colecciones creadas.	show collections
Crear colecciones	Se crea una colección.	db.createCollection (mcoleccion, {parametros})
Borrar colecciones	Se borra una colección.	db.mcoleccion.drop()
Copiar colecciones	Crear una colección en otra.	db.mcoleccion.aggregate()

4.1. Poblar la base de datos

Creamos una colección para poder ejecutar los ejemplos que vendrán después. Recuerda que MongoDB está programado en C++, almacena documentos JSON y, la forma más habitual de “conversar” con él es mediante JavaScript.

De momento, ejecuta el *script* (en javascript) aunque no entiendas algunos comandos. Son todos muy intuitivos. En el siguiente punto veremos cómo listar documentos (DQL) y, en la segunda parte de este tema, veremos cómo realizar las operaciones DML (insert, update, delete) sobre una BD en Mongo.

CONSEJO

Para ejecutar el código en tu consola, copia y pega este código en un editor de textos de Linux (Kwrite, Pluma, Gedit) o Windows (Bloc de notas) y quítale las tabulaciones de la izquierda antes de pegarlo en la consola. Funciona perfectamente con las tabulaciones, pero la salida queda más elegante sin ellas.

Creamos una colección con datos de prueba para los ejemplos:

```
// limpiar pantalla (buena praxis)
cls
// conectar con la BD y borrarla (la crea si no existe)
use pruebas
db.dropDatabase()
use pruebas
// crear una nueva colección
db.createCollection("estudiantes")
// crea varios documentos
var j_est1 = {nombre:"Sergio", apellidos:"Gracia Merinda", email:"sergio@gmail.com", edad:19}
var j_est2 = {nombre:"Pablo", apellidos:"Concar López", email:"graciamerinda@gmail.com"}
var j_est3 = {nombre:"Eva", apellidos:"Gredos Martínez", email:"gredoseva@gmail.com", edad:13}
var j_est4 = {nombre:"Miranda", apellidos:"Aranda Bada", email:"indo@arandabada.com", edad:49}
var j_est5 = {nombre:"Gabriel", apellidos:"Muro Menéndez", edad:12}
var j_est6 = {nombre:"Ana", apellidos:"Gracia Merinda", email:"ana@gmail.com", edad:19}
var j_est7 = {nombre:"Tania", apellidos:"Concar López", email:"tania@gmail.com"}
var j_est8 = {nombre:"Miguel", apellidos:"Gredos Martínez", email:"miguel@gmail.com", edad:39}
var j_est9 = {nombre:"Álvaro", apellidos:"Aranda Bada", email:"diana@genial.com", edad:49}
var j_est10 = {nombre:"Diana", apellidos:"Muro Menéndez", edad:29}
// inserta esos documentos en la colección
```



```
db.estudiantes.insertMany([j_est1, j_est2, j_est3, j_est4, j_est5, j_est6, j_est7, j_est8, j_est9,
```

Presta mucha atención a la salida que provocan esas órdenes en la consola de MongoDB. Fíjate en esos ids tan raros. Son las “claves primarias” que crea MongoDB de manera automática. Es lo más parecido que verás a las PRIMARY KEY de las BDR.

```
test> use pruebas
switched to db pruebas
pruebas> db.dropDatabase()
{ ok: 1, dropped: 'pruebas' }
pruebas> use pruebas
already on db pruebas
pruebas> // crear una nueva colección

pruebas> db.createCollection("estudiantes")
{ ok: 1 }
pruebas> // crea varios documentos

pruebas> var j_est1 = {nombre:"Sergio", apellidos:"Gracia Merinda", email:"sergio@gmail.com", edad:18}
pruebas> var j_est2 = {nombre:"Pablo", apellidos:"Concar López", email:"graciamerinda@gmail.com"}
pruebas> var j_est3 = {nombre:"Eva", apellidos:"Gredos Martínez", email:"gredoseva@gmail.com", edad:15}
pruebas> var j_est4 = {nombre:"Miranda", apellidos:"Aranda Bada", email:"mendo@arandabada.com", edad:20}
pruebas> var j_est5 = {nombre:"Gabriel", apellidos:"Muro Menéndez", edad:12}
pruebas> var j_est6 = {nombre:"Ana", apellidos:"Gracia Merinda", email:"ana@gmail.com", edad:19}
pruebas> var j_est7 = {nombre:"Tania", apellidos:"Concar López", email:"tania@gmail.com"}
pruebas> var j_est8 = {nombre:"Miguel", apellidos:"Gredos Martínez", email:"miguel@gmail.com", edad:17}
pruebas> var j_est9 = {nombre:"Álvaro", apellidos:"Aranda Bada", email:"diana@genial.com", edad:49}
pruebas> var j_est10 = {nombre:"Diana", apellidos:"Muro Menéndez", edad:29}

pruebas> // inserta esos documentos en la colección

pruebas> db.estudiantes.insertMany([j_est1, j_est2, j_est3, j_est4, j_est5, j_est6, j_est7, j_est8,
```

```
acknowledged: true,  
insertedIds: {  
  '0': ObjectId('65fdc088e399d41cc2c00e7e'),  
  '1': ObjectId('65fdc088e399d41cc2c00e7f'),  
  '2': ObjectId('65fdc088e399d41cc2c00e80'),  
  '3': ObjectId('65fdc088e399d41cc2c00e81'),  
  '4': ObjectId('65fdc088e399d41cc2c00e82'),  
  '5': ObjectId('65fdc088e399d41cc2c00e83'),  
  '6': ObjectId('65fdc088e399d41cc2c00e84'),  
  '7': ObjectId('65fdc088e399d41cc2c00e85'),  
  '8': ObjectId('65fdc088e399d41cc2c00e86'),  
  '9': ObjectId('65fdc088e399d41cc2c00e87')  
}  
}
```

Ahora que hemos poblado nuestra primera base de datos NoSQL (en MongoDB), ya podemos comenzar a “dialogar” con ella.

5. Lenguaje de consulta de datos (DQL)

Para leer documentos de una colección usamos el comando **find** (equivalente al SQL SELECT). Sintaxis:

```
db.<colección>.find(<filtro>, <proyección>).<operaciones>
```

```
<filtro> = {condición}
```

```
<proyección> = {campo1:1/0, ... , campoN:1/0}
```

```
<operaciones> = .sort(condición) / .limit(num) / .count()
```

siendo:

- **filtro**. Opcional. Filtra qué documentos mostrar, usando operadores.
- **proyección**. Opcional. Filtra qué campos mostrar.
- **operaciones**. Opcional. Aplica una operación al resultado (como por ejemplo ordenar).

5.1. MongoDB. Trabajando con filtros

Operadores genéricos (con símbolo dólar)

Operador	Descripción
\$exists:true/false	El campo debe existir (o no)
\$and[,,,] \$or[,,,] \$not[,,,]	Acumulación de condiciones
\$in[,,,] \$nin[,,,]	El campo debe estar (o no) en ese conjunto de valores
\$gt:val \$gte:val	Mayor que Mayor o igual que
\$lt:val \$lte:val	Menor que Menor o igual que
\$ne:val	No igual que (distinto de)

5.1.1. Ejemplos

Presta mucha atención a la salida que provocan estas órdenes en la consola de MongoDB. Fíjate en esos ids tan raros que siguen apareciendo. Pronto veremos cómo ocultarlos.

a) Find sin operadores

```
// Muestra todos los documentos que tienen como nombre "Sergio"
var j_valor1 = "Sergio"
var j_filtro1 = {nombre:j_valor1}
db.estudiantes.find(j_filtro1).pretty()
```

Y la salida:

```
pruebas> var j_valor1 = "Sergio"

pruebas> var j_filtro1 = {nombre:j_valor1}

pruebas> db.estudiantes.find(j_filtro1).pretty()
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7e'),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda',
    email: 'sergio@gmail.com',
    edad: 19
  }
]
```

El operador “**pretty**” se utiliza para que los datos devueltos se vean de forma más clara y se muestren en diferentes líneas. En la última versión de MongoDB ya no hace falta ponerlo, puesto que se aplica por defecto, tanto en entorno gráfico (Compass) como en consola.

b) Find con \$exists

```
// Muestra todos los documentos que NO tienen un campo llamado email
var j_valor1 = {$exists:false}
var j_filtro1 = {email:j_valor1}
db.estudiantes.find(j_filtro1)
```

Y la salida:

```
pruebas> var j_valor1 = {$exists:false}

pruebas> var j_filtro1 = {email:j_valor1}

pruebas> db.estudiantes.find(j_filtro1)
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e82'),
    nombre: 'Gabriel',
    apellidos: 'Muro Menéndez',
    edad: 12
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e87'),
    nombre: 'Diana',
    apellidos: 'Muro Menéndez',
    edad: 29
  }
]
```

c) Find con \$or

```
// Muestra todos los documentos que NO tienen un campo llamado email o tienen de nombre "Sergio"
var j_valor1 = {$exists:false}
var j_filtro1 = {email:j_valor1}
var j_valor2 = "Sergio"
var j_filtro2 = {nombre:j_valor2}
db.estudiantes.find({$or:[j_filtro1, j_filtro2]})
```

Y la salida:

```
pruebas> var j_valor1 = {$exists:false}

pruebas> var j_filtro1 = {email:j_valor1}

pruebas> var j_valor2 = "Sergio"

pruebas> var j_filtro2 = {nombre:j_valor2}

pruebas> db.estudiantes.find({$or:[j_filtro1, j_filtro2]})
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7e'),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda',
    email: 'sergio@gmail.com',
    edad: 19
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e82'),
    nombre: 'Gabriel',
    apellidos: 'Muro Menéndez',
    edad: 12
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e87'),
    nombre: 'Diana',
    apellidos: 'Muro Menéndez',
    edad: 29
  }
]
```

5.2. MongoDB. Trabajando con proyecciones

Nos permite ocultar o mostrar ciertos campos de cada documento. Utiliza el formato {campo: valor, campo:valor}, de manera que si el valor es 1 se incluirá ese campo.

Por defecto, se muestran todos los campos no asociados expresamente al valor 0 excepto si hay algún campo a 1. En este caso, NO puede haber campos a 0 (salvo el `_id`). Es decir, si indico que dos, tres o cuatro campos concretos deben aparecer, solo aparecerán esos dos, tres o cuatro.

5.2.1. Ejemplos

a) Mostrar solo los nombres (sin el _id)

```
// Hacemos una proyección, con el filtro vacío indicando que solo queremos el campo nombre
// Al indicar un campo a 1, ya se eliminan todos los demás (apellidos y email, en este caso)
// ¡NO se elimina el _id que tanto nos molestaba! Hay que indicarlo expresamente
// ¡CUIDADO! HAY QUE INDICAR SIEMPRE EL FILTRO SI INDICAMOS UNA PROYECCIÓN
var j_campos_proyeccion = {nombre:1, _id:0}
db.estudiantes.find({}, j_campos_proyeccion)
```

Y la salida:

```
pruebas> var j_campos_proyeccion = {nombre:1, _id:0}

pruebas> db.estudiantes.find({}, j_campos_proyeccion)
[
  { nombre: 'Sergio' },
  { nombre: 'Pablo' },
  { nombre: 'Eva' },
  { nombre: 'Miranda' },
  { nombre: 'Gabriel' },
  { nombre: 'Ana' },
  { nombre: 'Tania' },
  { nombre: 'Miguel' },
  { nombre: 'Álvaro' },
  { nombre: 'Diana' }
]
```

b) Mostrar solo nombre y apellidos (con el _id)

```
// Hago una proyección, con el filtro vacío indicando que solo quiero esos dos campos
// Al indicar un campo a 1, ya se eliminan todos los demás (el email, en este caso)
// ¡NO se elimina el _id que tanto nos molestaba! Hay que indicarlo expresamente
// ¡CUIDADO! HAY QUE INDICAR SIEMPRE EL FILTRO SI INDICAMOS UNA PROYECCIÓN
```

```
var j_campos_proyeccion = {nombre:1, apellidos:1}
db.estudiantes.find({}, j_campos_proyeccion)
```

Y la salida:

```
pruebas> var j_campos_proyeccion = {nombre:1, apellidos:1}

pruebas> db.estudiantes.find({}, j_campos_proyeccion)
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7e'),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7f'),
    nombre: 'Pablo',
    apellidos: 'Concar López'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e80'),
    nombre: 'Eva',
    apellidos: 'Gredos Martínez'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e81'),
    nombre: 'Miranda',
    apellidos: 'Aranda Bada'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e82'),
    nombre: 'Gabriel',
    apellidos: 'Muro Menéndez'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e83'),
    nombre: 'Ana',
    apellidos: 'Gracia Merinda'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e84'),
    nombre: 'Tania',
    apellidos: 'Concar López'
  }
]
```

```

},
{
  _id: ObjectId('65fdc088e399d41cc2c00e85'),
  nombre: 'Miguel',
  apellidos: 'Gredos Martínez'
},
{
  _id: ObjectId('65fdc088e399d41cc2c00e86'),
  nombre: 'Álvaro',
  apellidos: 'Aranda Bada'
},
{
  _id: ObjectId('65fdc088e399d41cc2c00e87'),
  nombre: 'Diana',
  apellidos: 'Muro Menéndez'
}
]

```

c) Mostrar todos los campos salvo el _id

```

// En este caso, debemos forzar a que aparezca el _id
// Recuerda que el _id es el único campo que debemos marcar a 0 si no lo quiero ver
// ¡CUIDADO! HAY QUE INDICAR SIEMPRE EL FILTRO SI INDICAMOS UNA PROYECCIÓN
var j_campos_proyeccion = {_id:0}
db.estudiantes.find({}, j_campos_proyeccion)

```

Y la salida:

```

pruebas> var j_campos_proyeccion = {_id:0}

pruebas> db.estudiantes.find({}, j_campos_proyeccion)
[
  {
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda',
    email: 'sergio@gmail.com',
    edad: 19
  },
  {
    nombre: 'Pablo',

```

```

    apellidos: 'Concar López',
    email: 'graciamerinda@gmail.com'
  },
  {
    nombre: 'Eva',
    apellidos: 'Gredos Martínez',
    email: 'gredoseva@gmail.com',
    edad: 13
  },
  {
    nombre: 'Miranda',
    apellidos: 'Aranda Bada',
    email: 'indo@arandabada.com',
    edad: 49
  },
  { nombre: 'Gabriel', apellidos: 'Muro Menéndez', edad: 12 },
  {
    nombre: 'Ana',
    apellidos: 'Gracia Merinda',
    email: 'ana@gmail.com',
    edad: 19
  },
  {
    nombre: 'Tania',
    apellidos: 'Concar López',
    email: 'tania@gmail.com'
  },
  {
    nombre: 'Miguel',
    apellidos: 'Gredos Martínez',
    email: 'miguel@gmail.com',
    edad: 39
  },
  {
    nombre: 'Álvaro',
    apellidos: 'Aranda Bada',
    email: 'diana@genial.com',
    edad: 49
  },
  { nombre: 'Diana', apellidos: 'Muro Menéndez', edad: 29 }
]

```

5.3. MongoDB. Trabajando con operaciones

Existen muchas operaciones y casi cualquier combinación entre ellas es válida. Nosotros veremos solo éstas: `sort`, `limit`, `countDocuments` y `count`.

```
<operaciones>  =  .sort(campos) / .limit(num) / .countDocuments() /  
.countDocuments(condición) / .count()
```

- Para **sort**, indicamos por qué campos se ordena y en qué **orden (1/-1)**.
- Para **limit**, solo un número.
- Para **countDocuments**, podemos pasar o no una condición
- Para **count**, solo podemos usarlo con `find`

La operación "count" sin usar find está obsoleta desde MongoDB 4.0. Para más información:

<https://stackoverflow.com/questions/65282889/mongodb-count-versus-countdocuments>

5.3.1. Ejemplos

a) Mostrar todos los estudiantes ordenados por apellidos y nombre, ascendentes los dos

```
// Añadimos la operación sort sobre la colección estudiantes, tras hacer find
// Indicamos los campos por los que queremos ordenar, y el sentido de ambos
var j_campos_sort = {apellidos:1, nombre:1}
db.estudiantes.find().sort(j_campos_sort)
```

Y la salida:

```
pruebas> var j_campos_sort = {apellidos:1, nombre:1}

pruebas> db.estudiantes.find().sort(j_campos_sort)
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e81'),
    nombre: 'Miranda',
    apellidos: 'Aranda Bada',
    email: 'indo@arandabada.com',
    edad: 49
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e86'),
    nombre: 'Álvaro',
    apellidos: 'Aranda Bada',
    email: 'diana@genial.com',
    edad: 49
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7f'),
    nombre: 'Pablo',
    apellidos: 'Concar López',
    email: 'graciamerinda@gmail.com'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e84'),
```

```
    nombre: 'Tania',
    apellidos: 'Concar López',
    email: 'tania@gmail.com'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e83'),
    nombre: 'Ana',
    apellidos: 'Gracia Merinda',
    email: 'ana@gmail.com',
    edad: 19
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7e'),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda',
    email: 'sergio@gmail.com',
    edad: 19
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e80'),
    nombre: 'Eva',
    apellidos: 'Gredos Martínez',
    email: 'gredoseva@gmail.com',
    edad: 13
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e85'),
    nombre: 'Miguel',
    apellidos: 'Gredos Martínez',
    email: 'miguel@gmail.com',
    edad: 39
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e87'),
    nombre: 'Diana',
    apellidos: 'Muro Menéndez',
    edad: 29
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e82'),
    nombre: 'Gabriel',
    apellidos: 'Muro Menéndez',
    edad: 12
  }
]
```


b) Mostrar el primer estudiante, ordenado por apellidos y nombre, ascendentes los dos

```
// Añadimos la operación sort sobre la colección estudiantes, tras hacer find
// Indicamos los campos por los que queremos ordenar, y el sentido de ambos
var j_campos_sort = {apellidos:1, nombre:1}
db.estudiantes.find().sort(j_campos_sort).limit(1)
```

Y la salida:

```
pruebas> var j_campos_sort = {apellidos:1, nombre:1}

pruebas> db.estudiantes.find().sort(j_campos_sort).limit(1)
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e81'),
    nombre: 'Miranda',
    apellidos: 'Aranda Bada',
    email: 'indo@arandabada.com',
    edad: 49
  }
]
```

c) Contar los estudiantes mayores de edad

```
// Añadimos la operación countDocuments sobre la colección estudiantes, sin usar find
var j_valor = {$gte:18}
var j_condicion = {edad:j_valor}
db.estudiantes.countDocuments(j_condicion)
```

Y la salida:

```
pruebas> var j_valor = {$gte:18}

pruebas> var j_condicion = {edad:j_valor}
```

```
pruebas> db.estudiantes.countDocuments(j_condicion)
6
```

d) Contar los estudiantes que tienen email

```
// Añadimos la operación countDocuments sobre la colección estudiantes, sin usar find
var j_valor = {$exists:true}
var j_condicion = {email:j_valor}
db.estudiantes.countDocuments(j_condicion)
```

Y la salida:

```
pruebas> var j_valor = {$exists:true}

pruebas> var j_condicion = {email:j_valor}

pruebas> db.estudiantes.countDocuments(j_condicion)
8
```

e) Contar los estudiantes que hay en total

```
// Añadimos la operación countDocuments sobre la colección estudiantes, sin usar find
db.estudiantes.countDocuments()
```

Y la salida:

```
pruebas> db.estudiantes.countDocuments()
10
```

5.4. Consultas combinadas

Ahora que ya conoces cómo usar filtros, proyecciones y operaciones, vamos a realizar consultas en las que tenemos que combinarlos todos.

5.4.1. Ejemplos

a) Mostrar solo el nombre de los estudiantes que NO tienen email, ordenados por nombre de manera descendente. Oculta el campo _id

```
var j_valor          = {$exists:false}
var j_condicion      = {email:j_valor}
var j_campos_proyeccion = {nombre:1, _id:0}
var j_campos_sort    = {nombre:-1}
db.estudiantes.find(j_condicion, j_campos_proyeccion).sort(j_campos_sort)
```

Y la salida:

```
pruebas> var j_valor = {$exists:false}

pruebas> var j_condicion = {email:j_valor}

pruebas> var j_campos_proyeccion = {nombre:1, _id:0}

pruebas> var j_campos_sort = {nombre:-1}

pruebas> db.estudiantes.find(j_condicion, j_campos_proyeccion).sort(j_campos_sort)
[ { nombre: 'Gabriel' }, { nombre: 'Diana' } ]
```

b) Mostrar nombre, apellidos y edad de los tres primeros estudiantes que tienen el campo edad ordenados por edad descendente. Oculta el campo _id

```
var j_valor          = {$exists:true}
var j_condicion      = {edad:j_valor}
var j_campos_proyeccion = {nombre:1, apellidos:1, edad:1, _id:0}
var j_campos_sort    = {edad:-1}
db.estudiantes.find(j_condicion, j_campos_proyeccion).sort(j_campos_sort).limit(3)
```

Y la salida:

```
pruebas> var j_valor          = {$exists:true}

pruebas> var j_condicion      = {edad:j_valor}

pruebas> var j_campos_proyeccion = {nombre:1, apellidos:1, edad:1, _id:0}

pruebas> var j_campos_sort    = {edad:-1}

pruebas> db.estudiantes.find(j_condicion, j_campos_proyeccion).sort(j_campos_sort).limit(3)
[
  { nombre: 'Álvaro', apellidos: 'Aranda Bada', edad: 49 },
  { nombre: 'Miranda', apellidos: 'Aranda Bada', edad: 49 },
  { nombre: 'Miguel', apellidos: 'Gredos Martínez', edad: 39 }
]
```

c) Contar los estudiantes mayores de edad (que tienen el campo edad)

```
// Sin usar find
var j_valor1      = {$exists:true}
var j_condicion1  = {edad:j_valor1}
var j_valor2      = {$gte:18}
var j_condicion2  = {edad:j_valor2}
var j_condicionFinal = {$and:[j_condicion1, j_condicion2]}
db.estudiantes.countDocuments(j_condicionFinal)

// Alternativamente usando find
var j_valor1      = {$exists:true}
var j_condicion1  = {edad:j_valor1}
var j_valor2      = {$gte:18}
var j_condicion2  = {edad:j_valor2}
var j_condicionFinal = {$and:[j_condicion1, j_condicion2]}
db.estudiantes.find(j_condicionFinal).count()
```

Y la salida:

```
pruebas> var j_valor1          = {$exists:true}
```

```

pruebas> var j_condicion1      = {edad:j_valor1}

pruebas> var j_valor2          = {$gte:18}

pruebas> var j_condicion2      = {edad:j_valor2}

pruebas> var j_condicionFinal  = {$and:[j_condicion1, j_condicion2]}

pruebas> db.estudiantes.countDocuments(j_condicionFinal)
6

```

```

pruebas> var j_valor1          = {$exists:true}

pruebas> var j_condicion1      = {edad:j_valor1}

pruebas> var j_valor2          = {$gte:18}

pruebas> var j_condicion2      = {edad:j_valor2}

pruebas> var j_condicionFinal  = {$and:[j_condicion1, j_condicion2]}

pruebas> db.estudiantes.find(j_condicionFinal).count()
6

```

d) Contar los estudiantes que tienen el campo email

```

// Usando find
var j_valor1      = {$exists:true}
var j_condicion1  = {email:j_valor1}
db.estudiantes.find(j_condicion1).count()

// Sin usar find
var j_valor1      = {$exists:true}
var j_condicion1  = {email:j_valor1}
db.estudiantes.countDocuments(j_condicion1)

```

Y la salida:

```

pruebas> var j_valor1      = {$exists:true}

pruebas> var j_condicion1 = {email:j_valor1}

pruebas> db.estudiantes.find(j_condicion1).count()
8

```

```

pruebas> var j_valor1      = {$exists:true}

pruebas> var j_condicion1 = {email:j_valor1}

pruebas> db.estudiantes.countDocuments(j_condicion1)
8

```

e) Mostrar los tres primeros estudiantes que tengan TODOS los campos (nombre, apellidos, email y edad) ordenados por apellidos y nombre, ocultando el campo _id

```

var j_valor          = {$exists:true}
var j_condicion1     = {nombre:j_valor}
var j_condicion2     = {apellidos:j_valor}
var j_condicion3     = {email:j_valor}
var j_condicion4     = {edad:j_valor}
var j_campos_proyeccion = {_id:0}
var j_campos_sort    = {apellidos:1, nombre:1}
var j_condicionFinal = {$and:[j_condicion1, j_condicion2, j_condicion3, j_condicion4]}
db.estudiantes.find(j_condicionFinal, j_campos_proyeccion).sort(j_campos_sort).limit(3)

```

Y la salida:

```

pruebas> var j_valor          = {$exists:true}

pruebas> var j_condicion1     = {nombre:j_valor}

pruebas> var j_condicion2     = {apellidos:j_valor}

pruebas> var j_condicion3     = {email:j_valor}

```

```

pruebas> var j_condicion4      = {edad:j_valor}

pruebas> var j_campos_proyeccion = {_id:0}

pruebas> var j_campos_sort      = {apellidos:1, nombre:1}

pruebas> var j_condicionFinal   = {$and:[j_condicion1, j_condicion2, j_condicion3, j_condicion4]}

pruebas> db.estudiantes.find(j_condicionFinal, j_campos_proyeccion).sort(j_campos_sort).limit(3)
[
  {
    nombre: 'Miranda',
    apellidos: 'Aranda Bada',
    email: 'indo@arandabada.com',
    edad: 49
  },
  {
    nombre: 'Álvaro',
    apellidos: 'Aranda Bada',
    email: 'diana@genial.com',
    edad: 49
  },
  {
    nombre: 'Ana',
    apellidos: 'Gracia Merinda',
    email: 'ana@gmail.com',
    edad: 19
  }
]

```

f) Mostrar nombre y apellidos de los dos primeros estudiantes. Esta vez, sí queremos el _id

```

var j_campos_proyeccion = {nombre: 1, apellidos: 1}
db.estudiantes.find({}, j_campos_proyeccion).limit(2)

```

Y la salida:

```

pruebas> var j_campos_proyeccion = {nombre: 1, apellidos: 1}

```



```
pruebas> db.estudiantes.find({}, j_campos_proyeccion).limit(2)
[
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7e'),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda'
  },
  {
    _id: ObjectId('65fdc088e399d41cc2c00e7f'),
    nombre: 'Pablo',
    apellidos: 'Concar López'
  }
]
```

6. Bibliografía

- ¿Qué es SQL y NoSQL? [Platzi]. <https://www.youtube.com/watch?v=CuAYLX6reXE>
- NO SQL: como se modelan las bbdd no relacionales? [HolaMundo]. <https://www.youtube.com/watch?v=Zdlude8l8w4>
- El concepto NoSQL, o cómo almacenar tus datos en una base de datos no relacional. <https://www.genbeta.com/desarrollo/el-concepto-nosql-o-como-almacenar-tus-datos-en-una-base-de-datos-no-relacional>
- Metodologías ágiles Scrum, Kanban 04 Triángulo de hierro. https://www.youtube.com/watch?v=PdzW4G_hbsw
- Proyectos ágiles. Triángulo de hierro. <https://proyectosagiles.org/triangulo-hierro/>
- Teorema CAP. Píldoras de conocimiento. https://www.youtube.com/watch?v=Ydv-y_oH_CY
- Una introducción a MongoDB. <https://www.genbeta.com/desarrollo/una-introduccion-a-mongodb>
- MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no). <https://www.genbeta.com/desarrollo/mongodb-que-es-como-functiona-y-cuando-podemos-usarlo-o-no>
- Tutorial gratuito de 30 vídeos. https://www.youtube.com/watch?v=nlOWsnO-d7Q&list=PLXXiznRYETLcJE_4U9qN2pysZOSYyL4Mh



Obra publicada con [Licencia Creative Commons Reconocimiento Compartir igual 4.0](https://creativecommons.org/licenses/by-sa/4.0/)