CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UNIDAD 8

BASES DE DATOS NOSQL: MONGODB

BASES DE DATOS 22/23
CFGS DAW

PARTE 1 DE 2. MONGODB: INSTALACIÓN, DDL Y DQL

Revisado por:
Abelardo Martínez y Pau Miñana
Autor:
Sergio Badal

ÍNDICE DE CONTENIDO

UD8.1. MONGODB: INSTALACIÓN, DDL Y DQL

1. BASES DE DATOS NO RELACIONALES (NOSQL)

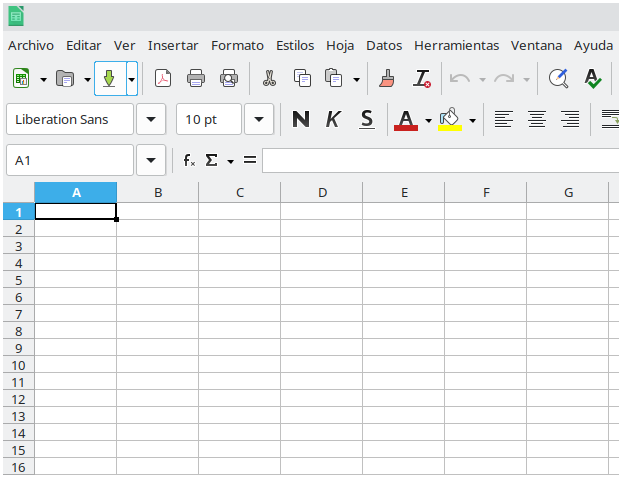
1.1 ¿POR QUÉ NO-SQL?

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?

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. Veamos un 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.

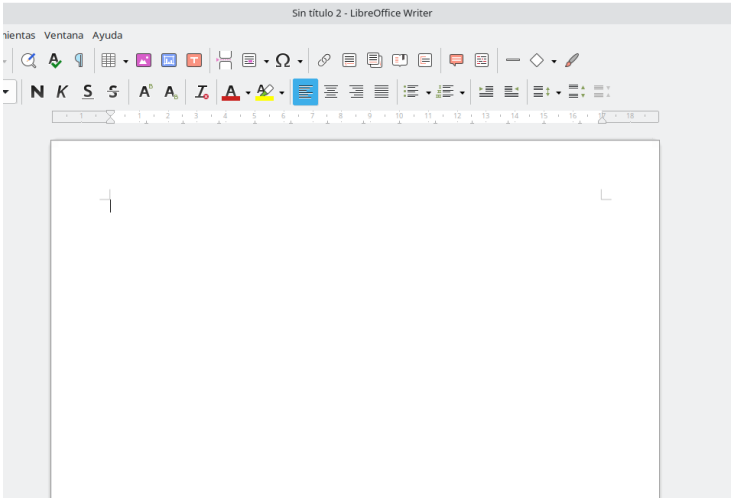
SQL se utiliza para ejecutar consultas, recuperar datos y editar datos actualizando, eliminando o creando nuevos registros.

Las BDR, encabezadas por las omnipresentes Oracle y MySQL, y seguidas de SQL Server y PostgreSQL, copan actualmente el 90-95% del mercado como puedes ver en el siguiente enlace:

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

410 systems in ranking, March 2023

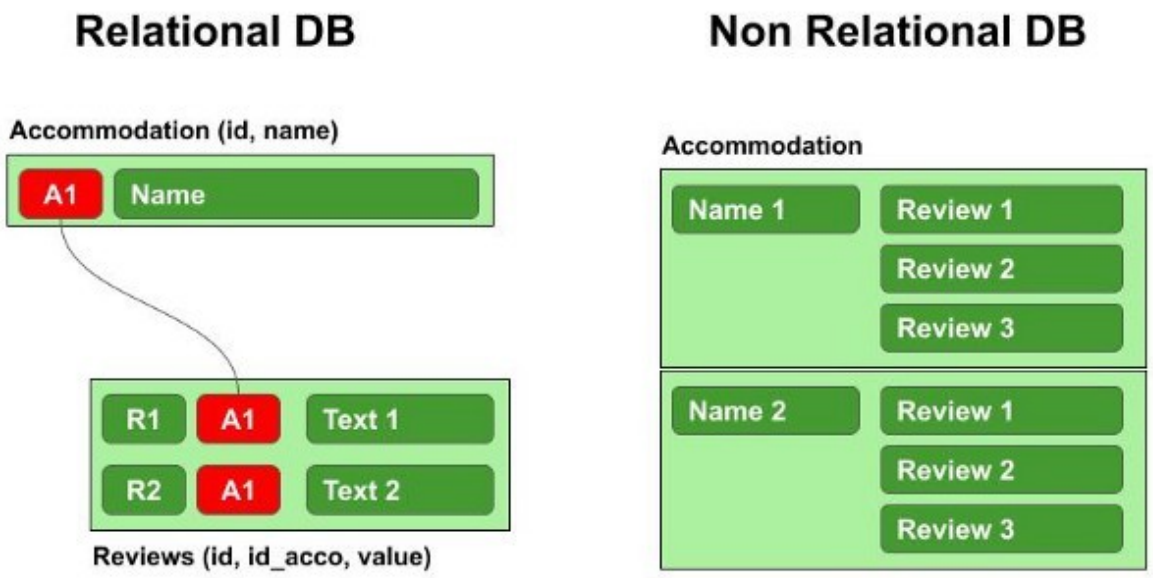
Rank Mar 2023	Rank Feb 2023	Rank Mar 2022	DBMS	Database Model	Score Mar 2023	Score Feb 2023	Score Mar 2022
1.	1.	1.	Oracle	Relational, Multi-model	1261.29	+13.77	+9.97
2.	2.	2.	MySQL	Relational, Multi-model	1182.79	-12.66	-15.45
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	922.01	-7.08	-11.77
4.	4.	4.	PostgreSQL	Relational, Multi-model	613.83	-2.67	-3.10
5.	5.	5.	MongoDB	Document, Multi-model	458.78	+6.02	-26.88
6.	6.	6.	Redis	Key-value, Multi-model	172.45	-1.39	-4.31
7.	7.	7.	IBM Db2	Relational, Multi-model	142.92	-0.04	-19.22
8.	8.	8.	Elasticsearch	Search engine, Multi-model	139.07	+0.47	-20.88
9.	9.	↑ 10.	SQLite	Relational	133.82	+1.15	+1.64
10.	10.	↓ 9.	Microsoft Access	Relational	132.06	+1.03	-3.37



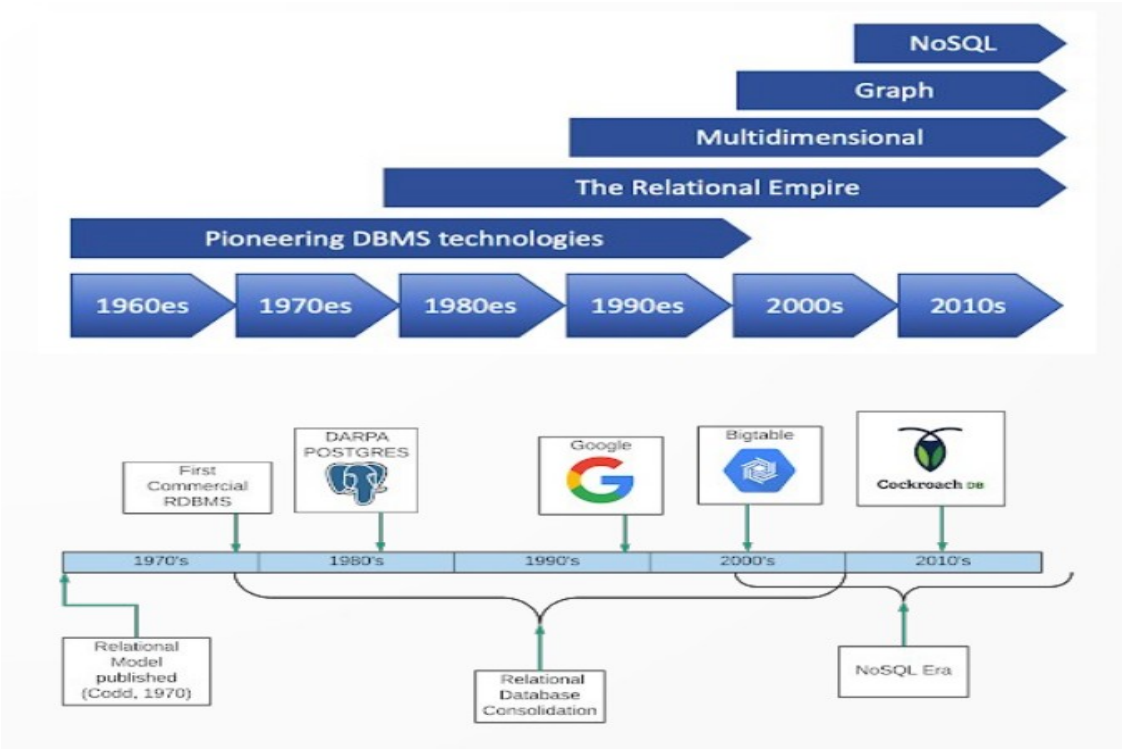
Volvamos al ejemplo inicial. Esta vez escogeremos el documento de procesador de texto. ¿Por qué? ¡Todo el espacio está en blanco! Los datos vienen en todas formas y tamaños... ¡libertad total!

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 común almacenar todos los datos que tienen algo en común en el mismo archivo o en la misma unidad de información (el documento o nodo).

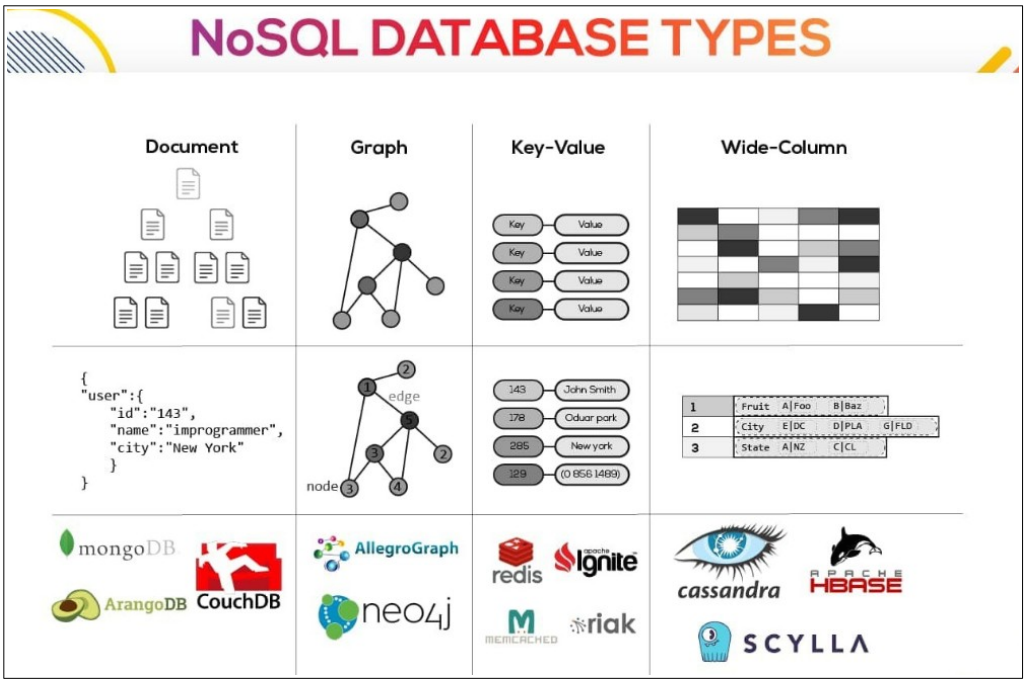


También puedes ver en esta gráfica la evolución de las bases de datos:

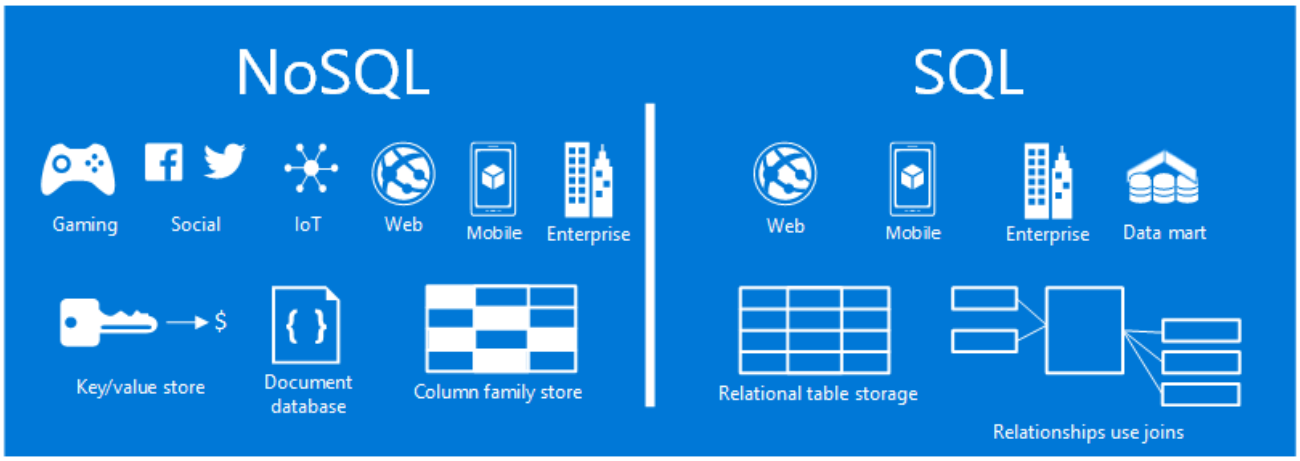


... una infografía con las principales diferencias...

... los diferentes tipos de NoSQL ...



... y sus diferentes usos:



Y unos enlaces recomendados por si quieres saber más:

¿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

https://www.genbeta.com/desarrollo/el-concepto-nosql-o-como-almacenar-tus-datos-en-una-base-de-datos-no-relacional

1.2 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).
- SACRIFICAMOS 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).

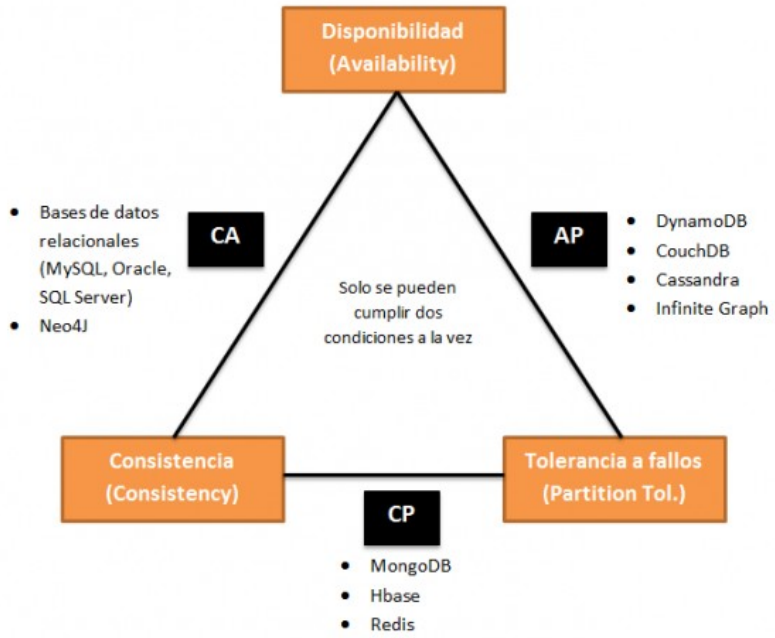
METODOLOGÍAS ÁGILES SCRUM, KANBAN 04 TRIÁNGULO DE HIERRO

https://www.youtube.com/watch?v=PdzW4G_hbsw

TRIÁNGULO DE HIERRO

https://proyectosagiles.org/triangulo-hierro/

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

TEOREMA CAP - PÍLDORAS DE CONOCIMIENTO

https://www.youtube.com/watch?v=Ydv-y_oH_CY

2. INTRODUCCIÓN A MONGODB

2.1 ¿QUÉ ES MONGODB?

Fuentes: https://www.genbeta.com/desarrollo/una-introduccion-a-mongodb

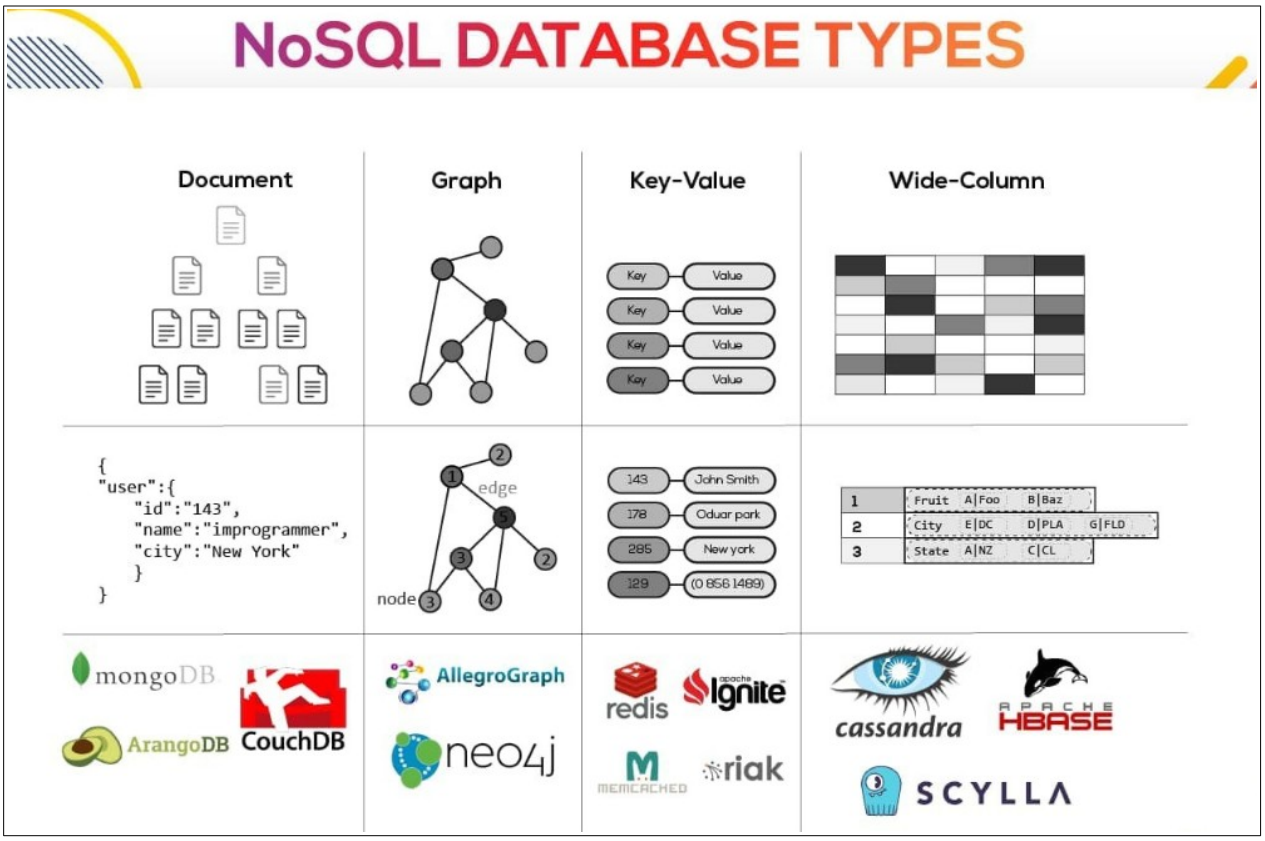
https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

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.

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

La lista de organizaciones que utiliza MongoDB es impresionante. Desde Foursquare y LinkedIn o empresas de telecomunicaciones como **Orange** y **Telefónica**. Empresas como Cisco, Bosch o plataformas de formación como Codecademy. Otras son eBay, Expedia.



2.2 ¿QUÉ ES MONGODB?

Como vemos en la imagen anterior, hay muchos tipos de BBDD no relacionales (NoSQL).

MongoDB es una base de datos **orientada a documentos**, como puede ser CouchDB o ArangoDB**.** 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.

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?

La última versión estable de MongoDB es la 6 (2022).

Puedes ver todas aquí : https://www.mongodb.com/evolved

2.3 ¿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 (no te asustes) :-) ...

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

... buscará todos los clientes cuyo nombre sea 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.

2.4 ¿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 tu información en varios servidores de similares características mientras que escalar verticalmente consiste en ampliar la capacidad de tus servidores, sin adquirir nuevos.

2.5 ¿DÓNDE NO SE DEBE USAR 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. 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. Eso sí, esta es una brecha que MongoDB va recortando con cada versión. En poco tiempo esto podría dejar de ser un problema.

2.6 ¿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 Windows, Linux y MacOs.

Cómo Instalar MongoDB paso a paso. Texto.

https://platzi.com/blog/como-instalar-mongodb-en-window-linux-y-mac/

Cómo Instalar MongoDB en Ubuntu. Texto.

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

Cómo Instalar MongoDB paso a paso en Windows. Video.

https://www.youtube.com/watch?v=gkCnXcxHC4o

2.7 ¿CÓMO PUEDO 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:

```
mise04@pc-mise04:~$ mongosh
Current Mongosh Log ID: 642189c75cdc2770285a2a81
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.0
Using MongoDB:          6.0.5
Using Mongosh:          1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-03-27T12:41:22.660+02:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See ht
   2023-03-27T12:41:23.707+02:00: Access control is not enabled for the database. Read and write access to data and configura
   2023-03-27T12:41:23.707+02:00: vm.max_map_count is too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

test>
```

Desde ese momento entraremos en la consola y podremos realizar consultas, de manera muy similar a como lo hacíamos con MySQL.

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 este todo lo complejo que queramos:

Archivo colores1.json

```
{
   "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"
      }
   ]
}
```

Archivo colores2.json

```
{
   "arrayColores":[{
         "rojo":"#f00",
         "verde":"#0f0",
         "azul":"#00f",
         "cyan":"#0ff",
         "magenta":"#f0f",
         "amarillo":"#ff0",
         "negro":"#000"
      }
   ]
}
```

Archivo colores3.json

```
{
   "rojo":"#f00",
   "verde":"#0f0",
   "azul":"#00f",
   "cyan":"#0ff",
   "magenta":"#f0f",
   "amarillo":"#ff0",
   "negro":"#000"
}
```

JSON BÁSICO. Texto.

https://desarrolloweb.com/home/json

APRENDE JSON EN 30 MINUTOS. Video.

https://www.youtube.com/watch?v=qQjALhIEM3A

4. TUTORIAL COMPLETO

Tutorial gratuito de 30 vídeos

https://www.youtube.com/watch?v=nlOWsnO-d7Q&list=PLXXiznRYETLcJE_4U9qN2pysZOSYyL4Mh

5. COMANDOS BÁSICOS

5.1 AYUDA Y SALIR

En Mongo, NO ACABAREMOS LOS COMANDOS CON ;

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

5.2 DATA MANIPULATION LANGUAGE (DDL)

Mostrar todas: Mostrar las bases de datos con **show databases [enter] o show dbs [enter]**

Conectar: Conectarnos a una base de datos concreta con **use mibasededatos [enter]**

- El comando “use” si no existe la bd, la crea.

Mostrar actual: Podemos ver en qué base de datos estamos con **db [enter]**

Borrar: Podemos borrar la BD a la que estamos conectados con **db.dropDatabase() [enter]**

Mostrar colecciones: Podemos mostrar colecciones creadas con **show collections [enter]**

Crear colecciones: **db.createCollection(micoleccion, {parametros}) [enter]**

Borrar colecciones: Se borra una colección con **db.micoleccion.drop() [enter]**

De ahora en adelante, asumimos un [enter] al final de cada comando.

Copiar colecciones: Crear una colección en otra con el comando **aggregate()**

Ejemplo:

```
db.micoleccion.aggregate([{$out: "copia_micoleccion"}])
```

5.3 DATA QUERY LANGUAGE (DQL)

5.3.1 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, pega antes el 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,
j_est10 ])
```

Presta mucha atención a la captura de la la siguiente página. En ella puedes ver 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>

pruebas>       db.createCollection("estudiantes")
{ ok: 1 }
pruebas>

pruebas>       var j_est1 = {nombre:"Sergio", apellidos:"Gracia Merinda", email:"sergio@gmail.com", edad:19}

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:13}

pruebas>       var j_est4 = {nombre:"Miranda", apellidos:"Aranda Bada", email:"indo@arandabada.com", edad:49}

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:39}

pruebas>       var j_est9 = {nombre:"Alvaro", apellidos:"Aranda Bada", email:"diana@genial.com", edad:49}

pruebas>       var j_est10 = {nombre:"Diana", apellidos:"Muro Menéndez", edad:29}

pruebas>
pruebas>       db.estudiantes.insertMany([j_est1, j_est2, j_est3, j_est4, j_est5, j_est6, j_est7, j_est8, j_est9, j_est10 ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("641c11e688ede466d22dfcd3"),
    '1': ObjectId("641c11e688ede466d22dfcd4"),
    '2': ObjectId("641c11e688ede466d22dfcd5"),
    '3': ObjectId("641c11e688ede466d22dfcd6"),
    '4': ObjectId("641c11e688ede466d22dfcd7"),
    '5': ObjectId("641c11e688ede466d22dfcd8"),
    '6': ObjectId("641c11e688ede466d22dfcd9"),
    '7': ObjectId("641c11e688ede466d22dfcda"),
    '8': ObjectId("641c11e688ede466d22dfcdb"),
    '9': ObjectId("641c11e688ede466d22dfcdc")
  }
}
```

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

5.3.2 Leer de la base de datos (básico)

Para leer documentos de una colección usamos el comando **find.**

Find (SQL SELECT)

```
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).

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.

MongoDB. Trabajando con filtros

OPERADORES genéricos (CON SÍMBOLO DÓLAR)

```
$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 (mayor que) $gte:val (mayor o igual que)
$lt:val (menor que) $lte:val (menor o igual que) $ne:val (no igual que)
```

Ejemplos:

- `db.estudiantes.find({nombre:"Sergio"}).pretty()`
- `db.estudiantes.find({$and:[{nombre:"Sergio"}, {email:{$exists:true}}]}).pretty()`

Ejemplos solo con filtros

```
// limpiar pantalla (buena praxis)
cls
// 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()

// 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).pretty()

// 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]}).pretty()
```

Presta mucha atención a la captura de la la siguiente página. En ella puedes ver la salida que provocan esas órdenes en la consola de MongoDB. Fíjate en esos ids tan raros que siguen apareciendo. Pronto veremos cómo ocultarlos.

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

pruebas>       var j_filtro1    = {email:j_valor1}

pruebas>       db.estudiantes.find(j_filtro1).pretty()
[
  {
    _id: ObjectId("641c11e688ede466d22dfcd7"),
    nombre: 'Gabriel',
    apellidos: 'Muro Menéndez',
    edad: 12
  },
  {
    _id: ObjectId("641c11e688ede466d22dfcdc"),
    nombre: 'Diana',
    apellidos: 'Muro Menéndez',
    edad: 29
  }
]
```

```
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]}).pretty()
[
  {
    _id: ObjectId("641c11e688ede466d22dfcd3"),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda',
    email: 'sergio@gmail.com',
    edad: 19
  },
  {
    _id: ObjectId("641c11e688ede466d22dfcd7"),
    nombre: 'Gabriel',
    apellidos: 'Muro Menéndez',
    edad: 12
  },
  {
    _id: ObjectId("641c11e688ede466d22dfcdc"),
    nombre: 'Diana',
    apellidos: 'Muro Menéndez',
    edad: 29
  }
]
```

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.

Ejemplos:

- `db.estudiantes.find({}, {nombre:1, apellidos:1}).pretty()`
- `db.estudiantes.find({}, {_id:0, apellidos:1}).pretty()`

Ejemplos solo con proyecciones

```
// limpiar pantalla (buena praxis)
cls
// a) Mostrar solo los nombres (sin el _ID)
// Hago una proyección, con el filtro vacío indicando que solo quiero 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).pretty()

// 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!
// 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).pretty()

// 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 debo 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).pretty()
```

Presta mucha atención a las siguientes capturas. En ellas puedes ver la salida que provocan esas órdenes en la consola de MongoDB.

Fíjate que ya no aparecen los ids cuando marcamos el campo _id:0.

En los apartados b y c, incluimos captura de solo los dos primeros resultados. Luego veremos cómo recortar esa salida a únicamente X resultados.

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

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

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

pruebas>       db.estudiantes.find({}, j_campos_proyeccion).pretty()
[
  {
    _id: ObjectId("641c11e688ede466d22dfcd3"),
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda'
  },
  {
    _id: ObjectId("641c11e688ede466d22dfcd4"),
    nombre: 'Pablo',
    apellidos: 'Concar López'
  },
```

```
pruebas>       var j_campos_proyeccion    = {_id:0}

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

MongoDB. Trabajando con operaciones

```
Existen muchas operaciones y casi cualquier combinación entre ellas es válida. Nosotros veremos
solo estas: sort, limit y count
<operaciones> = .sort(campos) / .limit(num) / .count() / .count(condicion)
Para sort, indicamos por qué campos se ordena y en qué orden (1/-1)
Para limit, solo un número
Para count, podemos pasar una condición si lo usamos sin find

Ejemplos:
•       db.estudiantes.find().sort({apellidos:1}).pretty()
•       db.estudiantes.find().count()
•       db.estudiantes.find().sort({apellidos:-1}).limit(1).pretty()
•       db.estudiantes.find().count()
•       db.estudiantes.count({edad:{$gt:18}})

// Como el campo “edad” no existe en ningún documento, el resultado de la última orden es 0
```

Ejemplos solo con operaciones

```
// limpiar pantalla (buena praxis)
cls
// a) Mostrar todos los estudiantes ordenados por apellidos y nombre, ascendentes los dos
// Añado la operación sort sobre la colección estudiantes, tras hacer find
// Indico los campos por los que quiero ordenar, y el sentido de ambos
var j_campos_sort   = {apellidos:1, nombre:1}
db.estudiantes.find().sort(j_campos_sort).pretty()

// b) Mostrar el primer estudiante, ordenado por apellidos y nombre, ascendentes los dos
// Añado la operación sort sobre la colección estudiantes, tras hacer find
// Indico los campos por los que quiero ordenar, y el sentido de ambos
var j_campos_sort   = {apellidos:1, nombre:1}
db.estudiantes.find().sort(j_campos_sort).limit(1).pretty()

// c) Contar los estudiantes mayores de edad
// Añado la operación count sobre la colección estudiantes, sin usar find
var j_valor = {$gte:18}
var j_condicion    = {edad:j_valor}
db.estudiantes.count(j_condicion)

// d) Contar los estudiantes que tienen email
// Añado la operación count sobre la colección estudiantes, sin usar find
var j_valor = {$exists:true}
var j_condicion    = {email:j_valor}
db.estudiantes.count(j_condicion)

// e) Contar los estudiantes que hay en total
// Añado la operación count sobre la colección estudiantes, usando find
db.estudiantes.find().count()
```

Presta mucha atención a las siguientes capturas. En ellas puedes ver la salida que provocan esas órdenes en la consola de MongoDB.

Fíjate que vuelven a aparecen los ids, ya que no estamos usando proyecciones (de momento).

En el apartado a) incluimos captura de solo los dos primeros resultados. En el siguiente apartado ya vemos cómo limitar esa salida.

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

pruebas>       db.estudiantes.find().sort(j_campos_sort).pretty()
[
  {
    _id: ObjectId("641c11e688ede466d22dfcd6"),
    nombre: 'Miranda',
    apellidos: 'Aranda Bada',
    email: 'indo@arandabada.com',
    edad: 49
  },
  {
    _id: ObjectId("641c11e688ede466d22dfcdb"),
    nombre: 'Alvaro',
    apellidos: 'Aranda Bada',
    email: 'diana@genial.com',
    edad: 49
  },
```

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

pruebas>       db.estudiantes.find().sort(j_campos_sort).limit(1).pretty()
[
  {
    _id: ObjectId("641c11e688ede466d22dfcd6"),
    nombre: 'Miranda',
    apellidos: 'Aranda Bada',
    email: 'indo@arandabada.com',
    edad: 49
  }
]
```

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

pruebas>       var j_condicion    = {edad:j_valor}

pruebas>       db.estudiantes.count(j_condicion)
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
6
```

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

pruebas>       var j_condicion    = {email:j_valor}

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

```
pruebas>       db.estudiantes.find().count()
10
```

5.3.3 Leer de la base de datos (avanzado)

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

Ahora la cosa se complica un poco...

Ejemplos avanzados

```
// limpiar pantalla (buena praxis)
cls
// 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).pretty()
```

```
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).pretty()
[ { 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).pretty()
```

```
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).pretty()
[
  { 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)
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.count(j_condicionFinal)

// ALTERNATIVAMENTE ...
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()
```

```
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.count(j_condicionFinal)
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()

// e) Contar los estudiantes que tienen el campo email (sin usar find)
var j_valor1                = {$exists:true}
var j_condicion1            = {email:j_valor1}
db.estudiantes.count(j_condicion1)

// f) 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).pretty()
```

```
pruebas>       db.estudiantes.find(j_condicionFinal, j_campos_proyeccion).sort(j_campos_sort).limit(3).pretty()
[
  {
    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
  }
]
```

```
// g) 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).pretty()
```