

# Big Data aplicada

---

UT3 – ALMACENAMIENTO DE DATOS MASIVO



Daniel López  
[DANIEL.LOPEZ@IESELCAMINAS.ORG](mailto:DANIEL.LOPEZ@IESELCAMINAS.ORG)

## UT3 – Almacenamiento de datos masivos

### Índice

1.	Introducción .....	2
1.1.	Bases de datos SQL.....	2
1.2.	Bases de datos NoSQL.....	4
2.	Introducción a MongoDB .....	10
2.1.	Características de MongoDB .....	10
2.2.	Instalación de MongoDB en Linux.....	14
2.3.	Primeros pasos .....	17
2.4.	Administración de las BBDD .....	21
2.5.	Gestión de los datos .....	23
3.	El sistema de almacenamiento Cloud .....	37
3.1.	Amazon S3 Estándar (S3 Estándar) .....	37
3.2.	AWS Glacier .....	37
3.3.	AWS RedShift (datawarehousing) .....	39
3.4.	Athena AWS .....	41
3.5.	AWS DynamoDB .....	42
4.	Laboratorio DynamoDB.....	48
4.1.	Añadir elementos .....	50
4.2.	Capacidades de las tablas.....	53
4.3.	Operaciones sobre los datos .....	55
4.4.	DAX - Caché de tablas.....	57
4.5.	Tablas con Lambda y Cloudwatch .....	57
5.	Bibliografía .....	65

## 1. Introducción

Desde los años 80, la mayoría de los sistemas de información almacenan sus datos en gestores relacionales

- Aplicaciones de banca
- CRM (Customer Relationship Management)
- ERP (Enterprise Resource Planning)
- Sistemas sanitarios
- Sistemas académicos
- Etc.

Básicamente, estos recogen procesos operacionales que gestionan datos simples y estructurados (p.ej. transacciones bancarias, compras, etc)

Es indiscutible que la forma en la que generamos datos y los consumimos ha cambiado desde la entrada de las redes sociales y la visualización en streaming. Debido a la cantidad de datos que se recopilan y se consumen, las bases de datos relacionales tradicionales no son el método más eficiente para afrontar esta nueva era de datos.

NoSQL – “Not Only SQL” son bases de datos no-relacionales y altamente distribuidas.

Surgen por la necesidad de afrontar ciertos problemas de escalabilidad y rendimiento que las BBDD relacionales resultan inefficientes cuando tienen que enfrentarse a grandes volúmenes de datos.

Aportando:

- Simplicidad en los diseños.
- Escalado horizontal.
- Mayor control en la disponibilidad.

En el artículo enlazado en Aules llamado “Bases de datos NoSQL: Guía definitiva”, escrito por Sara Martín, explica muy bien qué son y por qué surgen.

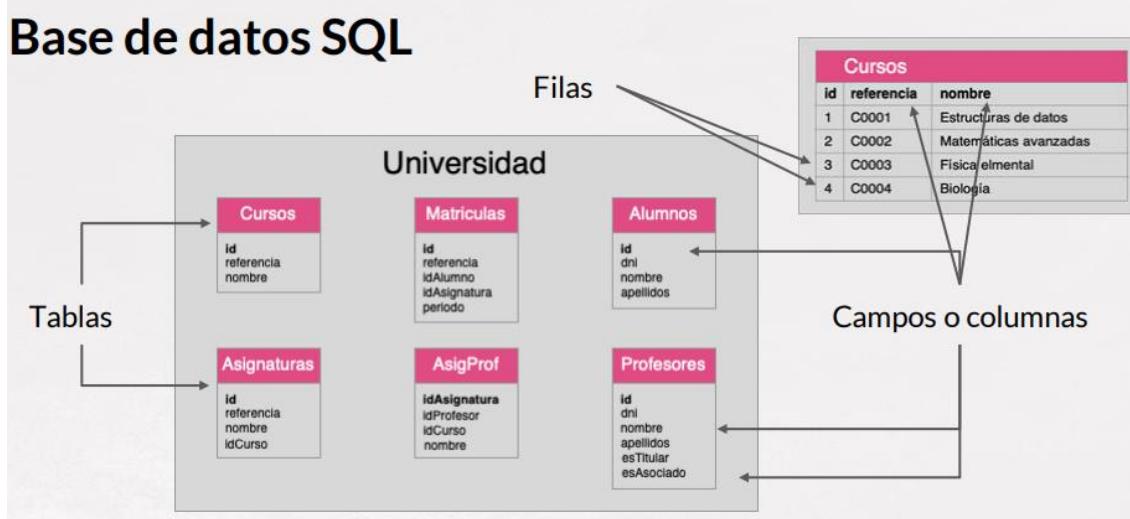
### 1.1. Bases de datos SQL

Antes de empezar, conviene recordar qué son las BBDD relacionales, estas BBDD son la forma de almacenar información más común antes de la llegada de otros tipos como NoSQL. Las características de estas BBDD son:

- La información se organiza en entidades denominadas tablas.
- Cada tabla está formada por filas o registros.
- La información de las tablas puede estar relacionada. Por eso se suelen denominar bases de datos relacionales.
- Usan SQL como lenguaje estructurado de consultas.
- Permiten el uso de transacciones (formas de almacenar la información) para almacenar la información aplicando los principios **ACID** (<https://es.wikipedia.org/wiki/ACID>):
  - Atomicidad
  - Consistencia
  - Aislamiento
  - Durabilidad

- La separación de la información en diversas entidades o tablas se hace aplicando un proceso denominado normalización. ([https://es.wikipedia.org/wiki/Normalizaci%C3%B3n\\_de\\_bases\\_de\\_datos](https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos))
- Siguen un esquema: conjunto de reglas que la información contenida en las tablas debe cumplir.
- Almacenan datos estructurados

Ejemplo: bbdd de universidad



Reglas a cumplir por la tabla Cursos

## El esquema para Cursos podría ser:

Cursos		
id	referencia	nombre
1	C0001	Estructuras de datos
2	C0002	Matemáticas avanzadas
3	C0003	Física elemental
4	C0004	Biología

Tiene 3 columnas:

- **id**:
  - int (entero)
  - debe ser positivo
  - no permite valores nulos
- **referencia**:
  - string (texto)
  - permite valores nulos
- **nombre**:
  - string (texto)
  - no admite valores nulos

reglas

El enfoque relacional no puede solucionar los problemas que derivan de los nuevos usos de internet. Pone solución en varios aspectos:

- No es capaz de almacenar información desestructurada
- Poca flexibilidad del esquema relacional

## Ventajas

- Ofrecen esquemas flexibles.
- Desarrollos más rápidos e iterativos.
- Ideales para datos semiestructurados y no estructurados
- Escalabilidad sencilla

- Alto rendimiento en consultas sobre datos que no implican relaciones jerárquicas
- Por tanto, son convenientes cuando:
- No se tiene un gran presupuesto
- Cuando las estructuras de datos son variables
- Para la captura y procesado de eventos

### Desventajas

- Cuando el modelo ACID encaja mejor.
- No todas las BBDD contemplan atomicidad e integridad de los datos
- Falta de compatibilidad entre instrucciones SQL
- Falta de estandarización
- En definitiva, cuando los datos deben ser consistentes sin dar posibilidad de error.

## 1.2. Bases de datos NoSQL

Pero, ¿por qué surgen este tipo de bbdd? Con la llegada de Internet, suponen una alternativa al modelo clásico de bases de datos SQL relacionales. Empresas como Google, Facebook, Amazon o Twitter comienzan a usarlas debido a problemas que el enfoque relacional no podía solucionar.

- Son idóneas con los datos a almacenar no siguen una estructura fija.
- Libre de esquemas
- Proporcionan replicación a través de escalado horizontal (Arquitectura distribuida)
- Consistencia débil
- Estructuras de datos sencillas
- Sistema de consultas propio
- Siguen el modelo BASE (Basic Availability, Soft State, Eventual Consistency) en lugar de ACID (Atomicity, Consistency, Isolation, Durability)
  - No garantizan los principios ACID. ↗ relacionadas con el esquema de transacciones, donde se ve relacionadas varias tablas.
    - MongoDB desde la versión 4, si permite el uso de transacciones.

### Modelo BASE

- **Basic Availability:** El sistema garantiza disponibilidad, en términos del Teorema de CAP
- **Soft State:** El estado del sistema puede cambiar a lo largo del tiempo, incluso sin entrada. Esto es provocado por el modelo de consistencia eventual.
- **Eventual Consistency:** el sistema alcanzará un estado consistente con el tiempo, siempre y cuando no reciba entrada durante este tiempo.

BASE VS Relacional: <https://queue.acm.org/detail.cfm?id=1394128>

### 1.2.1. NoSQL vs. Bases de datos relacionales

Esta tabla ofrece una breve comparación entre las funcionalidades de NoSQL y las bases de datos relacionales:

Feature	NoSQL Databases	Relational Databases
Performance	High	Low
Reliability	Poor	Good
Availability	Good	Good
Consistency	Poor	Good
Data Storage	Optimized for huge data	Medium sized to large
Scalability	High	High (but more expensive)

### 1.2.2. Cargas de trabajo operacionales

El panorama de Big Data está dominado por dos clases de tecnología: sistemas que brindan **capacidades operativas** para cargas de trabajo interactivas en tiempo real donde los datos se capturan y almacenan principalmente; y sistemas que brindan **capacidades analíticas** para análisis retrospectivos y complejos que pueden afectar a la mayoría o la totalidad de los datos. Estas clases de tecnología son complementarias y con frecuencia se implementan juntas.

Las cargas de trabajo operativas y analíticas para Big Data presentan requisitos opuestos y los sistemas han evolucionado para abordar sus demandas particulares por separado y de formas muy diferentes. Cada uno ha impulsado la creación de nuevas arquitecturas tecnológicas. Los sistemas operativos, como las bases de datos NoSQL, se enfocan en **atender solicitudes altamente concurrentes** mientras exhiben una baja latencia para las respuestas que operan con criterios de acceso altamente selectivos. Los sistemas analíticos, por otro lado, tienden a **centrarse en un alto rendimiento**; las consultas pueden ser muy complejas y tocar la mayoría, si no todos, los datos del sistema en cualquier momento. Ambos sistemas tienden a operar en muchos servidores que operan en un clúster, administrando decenas o cientos de terabytes de datos en miles de millones de registros.

### 1.2.3. Cargas de trabajo operativas

Para las cargas de trabajo operativas de Big Data, los sistemas de Big Data NoSQL, como las bases de datos de documentos, han surgido para abordar un amplio conjunto de aplicaciones, y otras arquitecturas, como almacenes de valores clave, almacenes de familias de columnas y bases de datos de gráficos, están optimizados para aplicaciones más específicas. Las tecnologías NoSQL, que se desarrollaron para abordar las deficiencias de las bases de datos relacionales en el entorno informático moderno, **son más rápidas y escalan mucho más rápida y económico que las bases de datos relacionales**.

Fundamentalmente, los sistemas de Big Data NoSQL están **diseñados para aprovechar las nuevas arquitecturas de computación en la nube** que han surgido durante la última década para permitir que los cálculos masivos se ejecuten de manera económica y eficiente. Esto hace que las cargas de trabajo operativas de Big Data sean mucho más fáciles de administrar y más económicas y rápidas de implementar.

Además de las interacciones del usuario con los datos, la mayoría de los sistemas operativos necesitan proporcionar cierto grado de inteligencia en tiempo real sobre los datos activos en el

sistema. Por ejemplo, en un juego multiusuario o una aplicación financiera, los agregados de las actividades del usuario o el rendimiento del instrumento se muestran a los usuarios para informar sus próximas acciones. Algunos sistemas NoSQL pueden proporcionar información sobre patrones y tendencias basados en datos en tiempo real con una codificación mínima y sin la necesidad de científicos de datos e infraestructura adicional.

#### 1.2.4. Cargas de trabajo analíticas

Las cargas de trabajo analíticas de Big Data, por otro lado, tienden a ser abordadas por los sistemas de base de datos **MPP(Massing Parallel Processing) y MapReduce**. Estas tecnologías también son una reacción a las limitaciones de las bases de datos relacionales tradicionales y su falta de capacidad para escalar más allá de los recursos de un solo servidor. Además, **MapReduce proporciona un nuevo método de análisis de datos que es complementario a las capacidades proporcionadas por SQL**.

A medida que las aplicaciones ganan terreno y sus usuarios generan volúmenes cada vez mayores de datos, hay una serie de cargas de trabajo analíticas retrospectivas que brindan valor real a la empresa. Donde estas cargas de trabajo involucran algoritmos que son más sofisticados que la simple agregación, MapReduce ha surgido como la primera opción para el análisis de Big Data. Algunos sistemas NoSQL proporcionan la funcionalidad nativa de MapReduce que permite realizar análisis de los datos operativos en el lugar. Durante parte del curso, nos centraremos en la relación entre los sistemas **NoSQL** con sistemas analíticos como **Hadoop para MapReduce**.

#### 1.2.5. Teorema de CAP

“En un sistema distribuido de almacenamiento de datos no podemos garantizar consistencia, disponibilidad y tolerancia al particionado.”

- **Consistencia (C)**: cualquier lectura recibe como respuesta el dato más reciente, nunca un dato obsoleto.
- **Disponibilidad (A)**: Toda petición obtiene una respuesta no errónea en un tiempo razonable, aunque ésta no sea el dato más reciente.
- **Partition Tolerance (P)**: el sistema sigue funcionando, aunque exista un problema de comunicación o fallos parciales.

En caso de que ocurra una partición hay que elegir entre consistencia o disponibilidad.

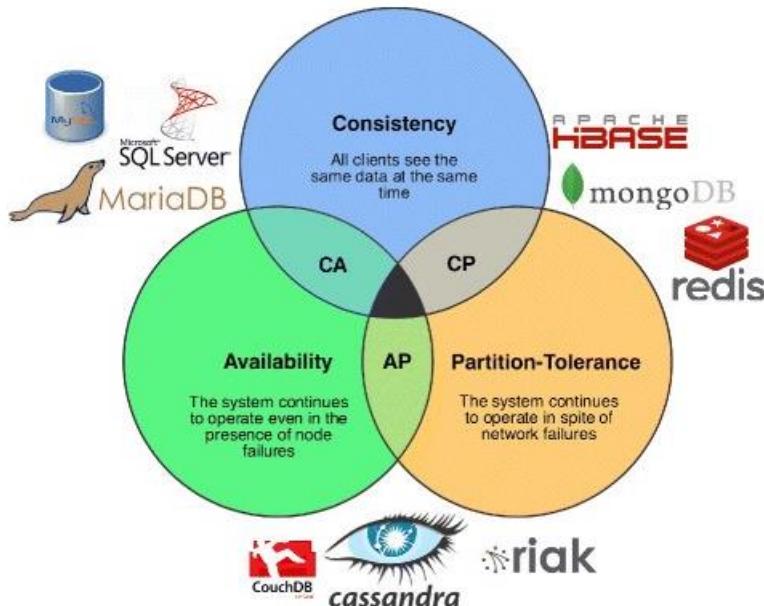
- **Elegimos consistencia**: el sistema devolverá un error o un tiempo de espera largo
- **Elegimos disponibilidad**: el sistema siempre procesará consultas e intentará devolver la información más reciente, aunque no garantice que esté actualizada debido a la partición de la red.

Teorema de CAP: [https://es.wikipedia.org/wiki/Teorema\\_CAP](https://es.wikipedia.org/wiki/Teorema_CAP)

Esto solo se hará si hay un fallo en la red, por lo que podemos distinguir 3 tipo de aplicaciones en función de las dos opciones que elijan:

- **Tipo AP**: garantizan disponibilidad y tolerancia al particionado, pero no que los datos sean igual en todos los nodos. Ejemplos:
  - Cassandra y DynamoDB
- **Tipo CP**: no garantizan el 100% de disponibilidad al cliente, pero el sistema se enfoca en aplicar los cambios de forma consistente, garantizando que los datos están actualizados, aunque se pierda la comunicación. Ejemplos:

- MongoDB y redis
- **Tipo CA:** garantizan que el sistema siempre estará disponible y con los datos siempre serán iguales en sus nodos, pero no tienen tolerancia a particionado.
  - Mayoría de BBDD relacionales.



### 1.2.6. Tipos de BBDD NoSQL

Dentro de este tipo de bases de datos no relacionales podemos encontrar los siguientes diferentes tipos:

- **Documental:** Una clave única para cada registro que normalmente suele ser un documento con una estructura simple como JSON o XML.
  - A este tipo corresponde MongoDB
  - Se apoya en la utilización de documentos para almacenar información.

```
{
  "_id" : ObjectId("54f612b6029b47909a90ce8d"),
  "title" : "A Tale of Two Cities",
  "text" : "It was the best of times, it was the worst
  "authorship" : "Charles Dickens"
}
```

MongoDB	SQL
Base de Datos	Base de Datos
Colecciones	Tablas
Documentos	Filas (registros)
Campos	Columnas

- Los documentos se agrupan en colecciones.
- Modelado flexible, recomendado para aplicaciones web, móviles o rrss que varían constantemente.
- Escritura rápida y mayor rendimiento.

- **Clave-Valor:** En este tipo de modelos cada elemento tiene asociada una clave única.

- Mejoran el rendimiento de ciertas apps, actuando en forma de índices.
- Tiene asociada una clave única lo que permite un acceso muy rápido.
- Su objetivo es la escalabilidad y la disponibilidad.
- Operaciones básicas get, put, delete.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

- Recomendado allí donde es necesario un acceso muy rápido en un volumen inmenso de datos.
- Sesiones
- E-shopping
- No existe un estándar para el manejo de datos.
- Un único método de acceso.

- ▶ **Es distribuida**
- ▶ **Escala linealmente**
- ▶ **No sigue un patrón maestro servidor. Es P2P**

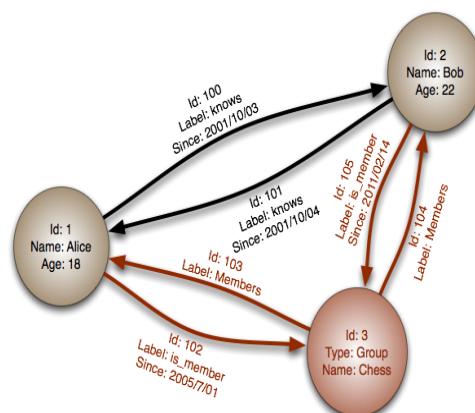


- ▶ **Escalabilidad horizontal**



- **BBDD de grafos:** aquellas cuyas relaciones pueden representarse mediante grafos.

- Representan la información con grafos:
- Nodos: propiedades de los datos.
- Aristas: relaciones entre los objetos.
- Utilizan algoritmos especiales para realizar las búsquedas:
- Búsqueda en profundidad: el siguiente nodo más profundo.
- Búsqueda en anchura: va moviéndose entre los niveles.
- Resultados en tiempo real.
- Estructuras flexibles y ágiles
- Difícil de escalar
- Ejemplo: Neo4j



- **BBDD columnar:** los datos de cada entrada están dispuestos uno debajo del otro.
- Cada entrada genera una columna
- Los datos están dispuestos uno debajo del otro
- Gira la BBDD orientada a filas:

ID	BBDD	TIPO
1	MongoDB	Documental
2	Cassandra	Key-value
3	Redshift	Columnar

ID	1	2	3
BBDD	MongoDB	Cassandra	RedShift
TIPO	Documental	Key-value	Columnar

- En el disco duro los datos se muestran de manera unidimensional:
  - 1, MongoDB, Documental, 2, Cassandra, Key-value..
  - 1, 2, 3; MongoDB, Cassandra, Redshift...
- Aconsejado para evaluación en BigData
- Desaconsejado en aplicaciones transaccionales

### Columnar:

1	Things	A	foo	B	bar	C	baz
2	Things	C	bam	E	coh	People	A Emmanuel
3	Languages	A	C	B	Java	C	Ceylon

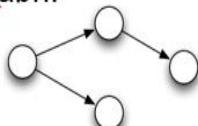
### Documental:

```
{
  "user": {
    "id": "124",
    "name": "Emmanuel",
    "addresses": [
      { "city": "Paris", "country": "France" },
      { "city": "Atlanta", "country": "USA" }
    ]
  }
}
```

### Key-value:

key	value
123	Address@23
126	"Booya"

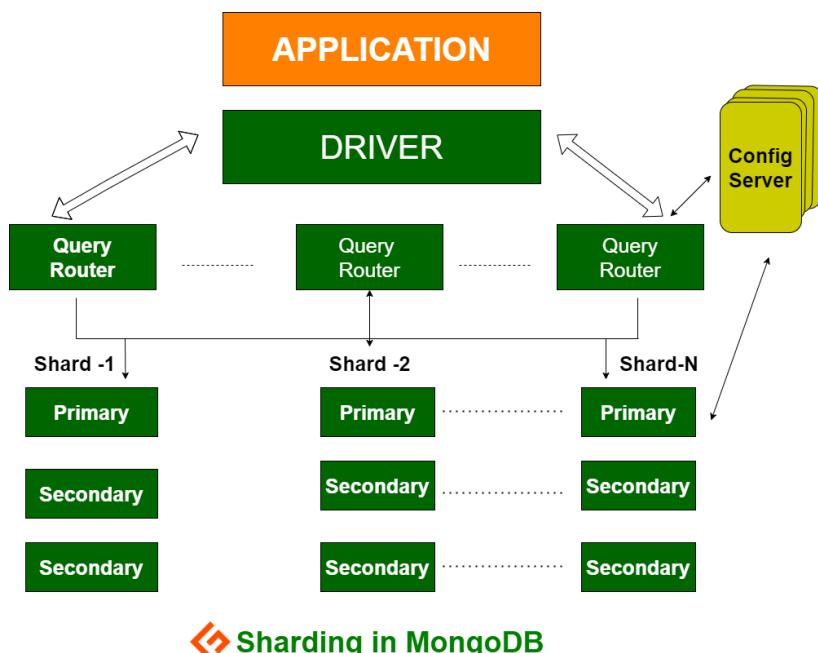
### Graph:



## 2. Introducción a MongoDB

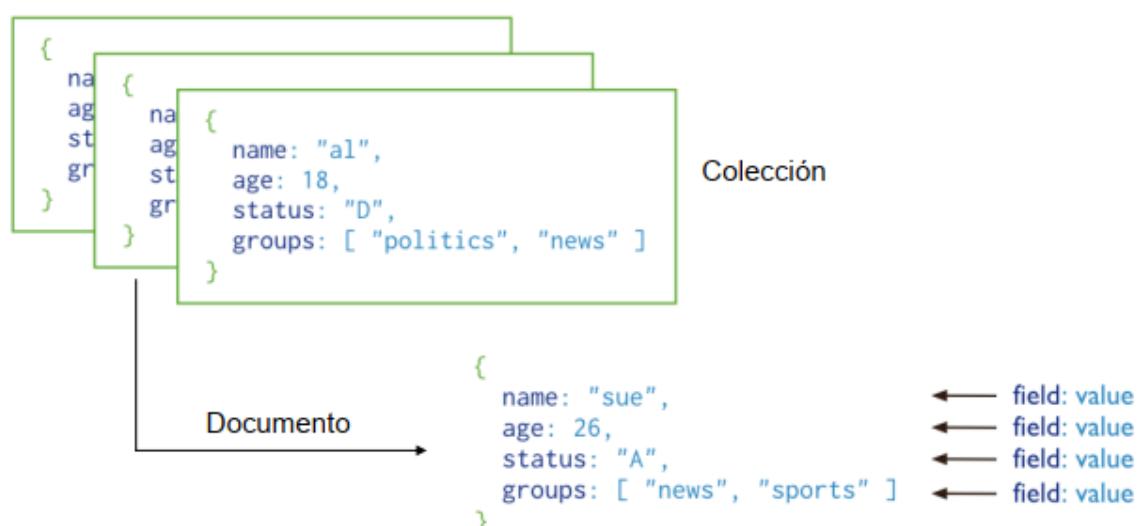
MongoDB es una base de datos que nativamente se concibió como documental, de forma que los datos se almacenan en documentos JSON de una forma flexible, semiestructurada y escalable horizontalmente.

En definitiva, intenta ser una base de datos que resuelva ineficiencias del modelo relacional clásico en aspectos de complejidad, flexibilidad, orientación a la programación, escalabilidad y rendimiento.



### ❖ Sharding in MongoDB

En MongoDB la información se almacena en colecciones de documentos, presentados en JSON, aunque internamente se integran de forma binaria (BSON)



### 2.1. Características de MongoDB

- Orientada a documentos (**JSON**).
- No sigue ningún esquema (**schemaless**).

- Permite que en una misma colección podamos tener: documentos json con diferente estructura.
- Dispone de una consola (shell) construida sobre Javascript lo que permite ejecutar muchas de sus funciones.
  - Si tienes conocimientos de JavaScript verás que son similares.
- Ampliamente utilizada en aplicaciones para Internet y de BigData.

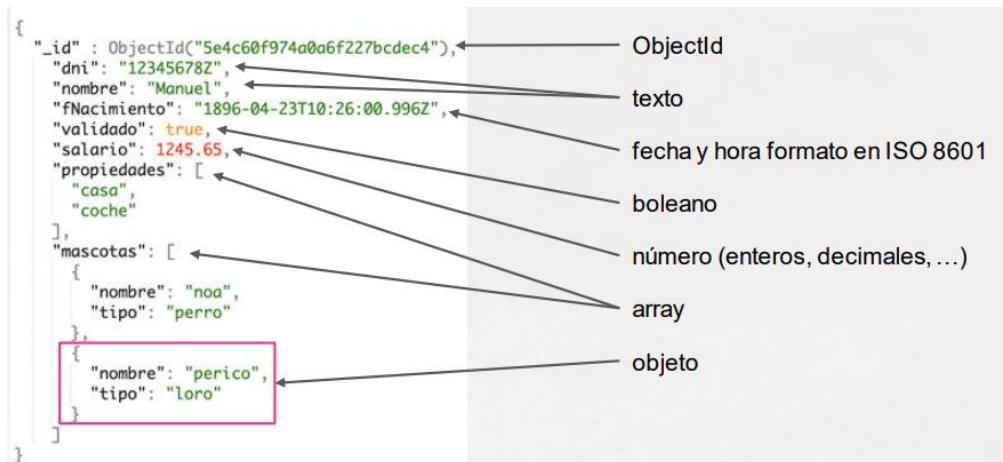
### 2.1.1. Formato JSON

El formato JSON (JavaScript Object Notation) es un formato de texto sencillo para el intercambio de datos. A su vez, es un conjunto de notación literal de objetos de JavaScript.

Cuando apareció lo hizo como una alternativa al formato XML, de forma que resultara más fácil de leer y escribir. Es uno de los formatos más ampliamente utilizados en aplicaciones de BBDD, sobre todo en todas las aplicaciones basadas en JavaScript.

El formato JSON se basa en la siguiente estructura: (**cadena:valor**)

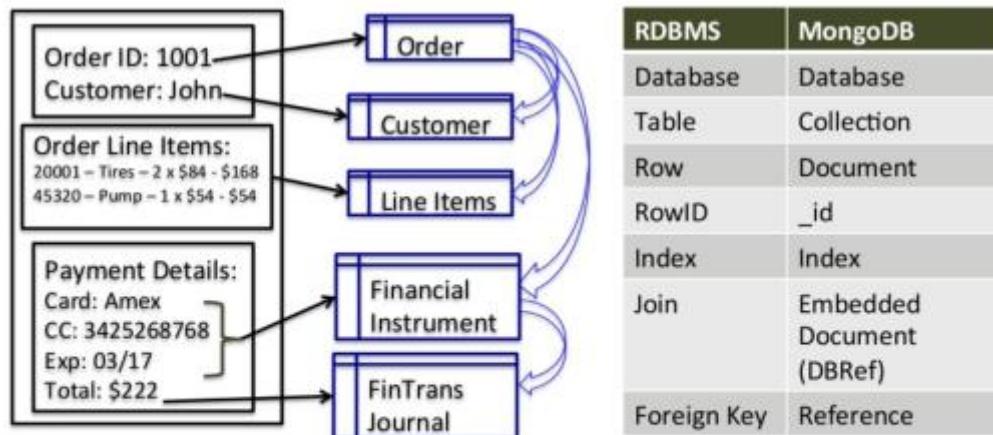
Un ejemplo de un objeto JSON:



Es un formato orientado a objetos que pueden contener elementos simples, arrays y otros documentos anidados, que unido al Javascript como lenguaje de manejo, denota su fuerte orientación a la programación. Ejemplo 2:

```
1  --Switch to database named 'foo':
2  use foo;
3
4  --Insert three documents into 'purchase_hist' collection:
5  db.purchase_hist.insert([
6      "_id" : ObjectId("52ce3648a4a0bb15a8e07b75"),
7      "CustomerID" : 23424,
8      "First_Name" : "Mary",
9      "Middle_Name" : "Joeanne",
10     "Last_Name" : "Black",
11     "Email_Address" : "mary0120@yahoo.com.au",
12     "Phone_Number" : "03-8573-9455",
13     "Purchased_Units" : 1,
14     "Purchased_Value" : 29.99,
15     "Item_SKU" : "RMK973",
16     "Rec_Created_Date" : ISODate("2011-12-31T13:00:00Z")
17 },
18 {
19     "_id" : ObjectId("52ce3648a4a0bb15a8e07b76"),
20     "CustomerID" : 84933,
21     "First_Name" : "John",
22     "Middle_Name" : "Lorance",
23     "Last_Name" : "Moore",
24     "Email_Address" : "johnnym@awol.com",
25     "Phone_Number" : "03-3423-1155",
26     "Purchased_Units" : 1,
27     "Purchased_Value" : 49.0,
28     "Item_SKU" : "MKK833",
29     "Rec_Created_Date" : ISODate("2011-12-31T13:00:00Z")
30 },
31 {
32     "_id" : ObjectId("52ce3648a4a0bb15a8e07b77"),
33     "CustomerID" : 19583,
34     "First_Name" : "Martin",
35     "Middle_Name" : null,
36     "Last_Name" : "Laser",
37     "Email_Address" : "mlaser91@aol.com",
38     "Phone_Number" : "03-2355-1109",
39     "Purchased_Units" : 5,
40     "Purchased_Value" : 2099.49,
41     "Item_SKU" : "HHY009",
42     "Rec_Created_Date" : ISODate("2012-03-31T13:00:00Z")
43 });
44
45  --Show inserted documents from 'purchase_hist' collection:
46  db.purchase_hist.find();
```

Hay que tener en cuenta que MongoDB tiene una jerarquización con equivalencias al modelo relacional:

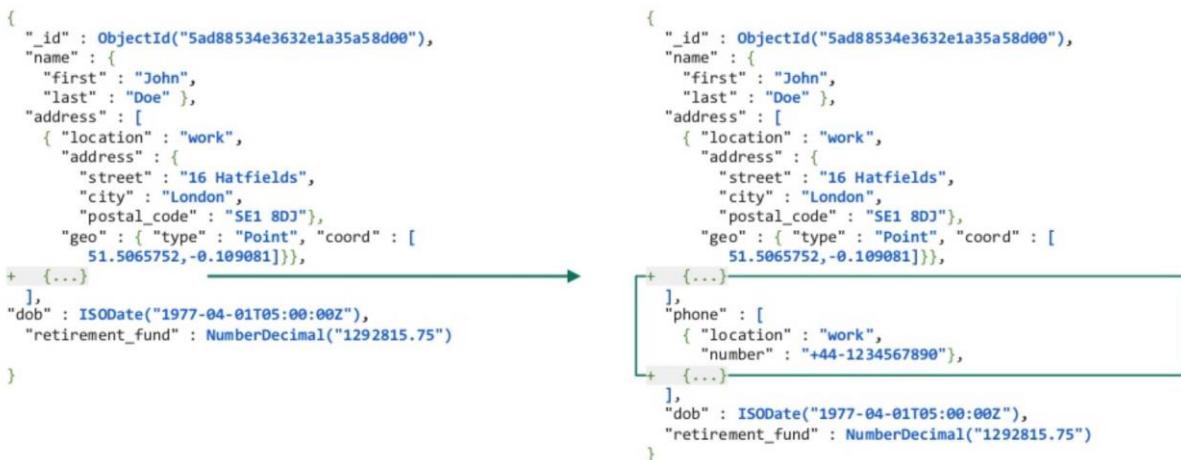


```
{
  "order_id": "1001", "customer": "John",
  "orderitems": [ {"prodid": "20001", "prodname": "Tires", "Qty": 2, "price": 168},
    {"prodid": "45320", "prodname": "Pump", "Qty": 1, "price": 54} ],
  "pcard": "Amex", "pcc": "3425268768", "pexp": "03/17", "ord_tot": 222 }
```

Frente a un modelo relacional completamente normalizado, el modelo documental aglutina gran cantidad de entidades en un documento. Esto resulta atractivo y natural para la programación:

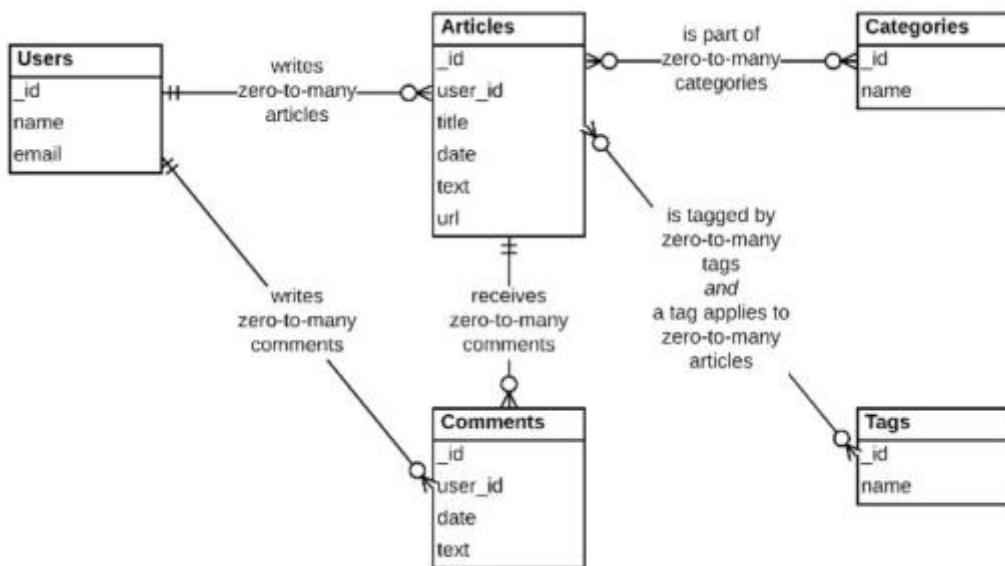


Este sistema es excelente ante evoluciones propias del modelo, de forma que realizar modificaciones resulta fácil:



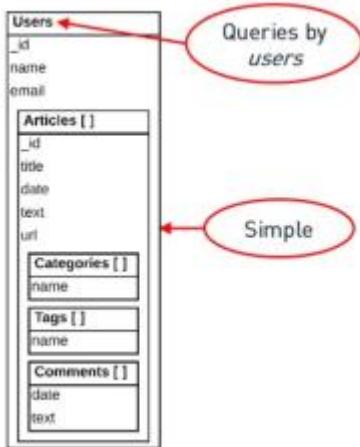
En algún momento, es posible que necesitemos cambiar de una BBDD relacional a un documental como MongoDB. En el siguiente ejemplo, convertiremos la BBDD actual en una documental. En este tipo de conversiones, no siempre tenemos una única solución, por ello, presentamos 3.

Tenemos una BBDD relacional de un blog web:

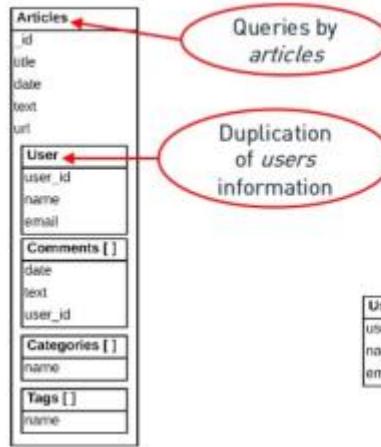


Cualquiera de las siguientes soluciones será válida:

*Solution A*



*Solution B*



*Solution C*



Al haber varias implementaciones de la estructura de la BBDD, la solución debe orientarse a la propia programación, de forma que nos facilite el trabajo.

## 2.2. Instalación de MongoDB en Linux

Si bien es cierto que en Big Data el uso de las BBDD generalmente se realiza en la nube, para poder practicar con este tipo de bbdd vamos instalarla en local, dejando para el siguiente punto el uso del Cloud de MongoDB.

La instalación de mongoDB tanto en Linux como en Windows está en un documento enlazado a Aules, donde podéis encontrar paso a paso toda la instalación que debemos hacer para instalarlo en local. No obstante, incluyo los pasos para instalar en Linux:

Importar la clave pública utilizada por el sistema de gestión de paquetes

```
wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key add -
```

Crear el /etc/apt/sources.list.d/mongodb-org-5.0.list

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

Recargar

```
sudo apt-get update
```

Instalar

```
sudo apt-get install -y mongodb-org
```

Para comprobar que todo ha ido bien a través del terminal puedes probar las siguientes opciones:

Iniciar

- sudo systemctl start mongod

Saber el estado:

- sudo systemctl status mongod

Parar:

- sudo systemctl stop mongod

Reiniciar:

- sudo systemctl restart mongod

Para entrar a la consola de MongoDB:

- mongo

Robo3T:

Descargamos el paquete:

```
wget -c https://download-test.robomongo.org/linux/robo3t-1.3.1-linux-x86\_64-7419c406.tar.gz
```

Descomprimimos el archivo

```
tar -xvzf robo3t-1.3.1-linux-x86_64-7419c406.tar.gz
```

Movemos la carpeta a otro directorio

```
sudo mv robo3t-1.3.1-linux-x86_64-7419c406/* /usr/local/bin/robo3t
```

Nos movemos la carpeta:

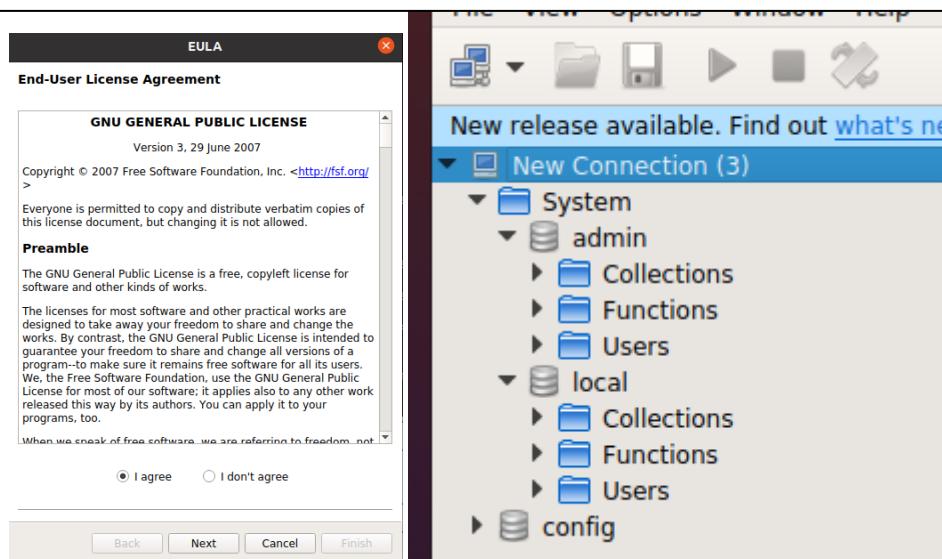
```
cd /usr/local/bin/robo3t/bin
```

Le damos permisos:

```
sudo chmod +x robo3t ./robo3t
```

Para ejecutar:

```
./robo3t
```



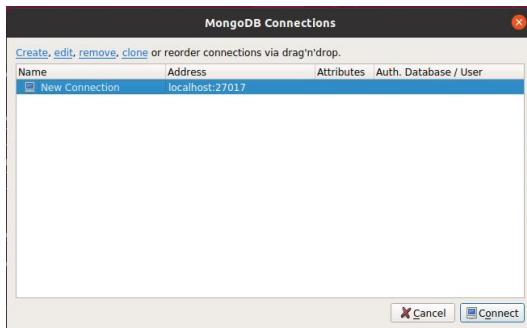
**Actividad 1:** realiza la instalación de MongoDB y de Robo3T. Comprueba que funciona y que puedes establecer una conexión con la base de datos.

### 2.2.1. Entorno de MongoDB

El interfaz de usuario de MongoDB podemos repartirlo en dos partes: la primera sería mediante el uso del terminal, simplemente escribiendo mongo, accediendo a su shell. Si el servicio no estuviera iniciado, obtendríamos el siguiente error:

```
bigdata@BigData:~$ mongo
MongoDB shell version v5.0.5
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Error: couldn't connect to server 127.0.0.1:27017, connection attempt failed: SocketException: Error connecting to 127.0.0.1:27017 :: caused by :: Connection refused :
connect@src/mongo/shell/mongo.js:372:17
@(connect):2:6
exception: connect failed
Exiting with code 1
```

El otro entorno sobre el que vamos a trabajar será el de robo3T, donde podemos hacer la conexión a nuestra máquina local, justo como en la imagen del punto anterior.



### 2.3. Primeros pasos

Para listar las bbdd creadas podemos hacerlo tanto por terminal como por Robo3T. Para hacerlo mediante terminal:

- Show dbs: revisar las BBDD creadas

En cambio, en Robo3T nos aparece al principio del entorno

Si queremos cambiar/crear una nueva bbdd usaremos el siguiente comando:

```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
>
```

- Use + nombreBBDD:
  - Ejemplo: use nuevaBBDD

Sin embargo, tal y como se puede ver en la imagen, la nueva BBDD no se ha creado, esto es debido a que hasta que no introducimos una colección en ella, es como si no se hubiera creado.

- db.getName() o db

```
> db.stats()
{
    "db" : "nuevaBBDD",
    "collections" : 2,
    "views" : 0,
    "objects" : 102,
    "avgObjSize" : 39.03921568627451,
    "dataSize" : 3982,
    "storageSize" : 57344,
    "freeStorageSize" : 16384,
    "indexes" : 3,
    "indexSize" : 77824,
    "indexFreeStorageSize" : 16384,
    "totalSize" : 135168,
    "totalFreeStorageSize" : 32768,
    "scaleFactor" : 1,
    "fsUsedSize" : 13750726656,
    "fsTotalSize" : 52044496896,
    "ok" : 1
}
>
> db.getName()
nuevaBBDD
> db
nuevaBBDD
```

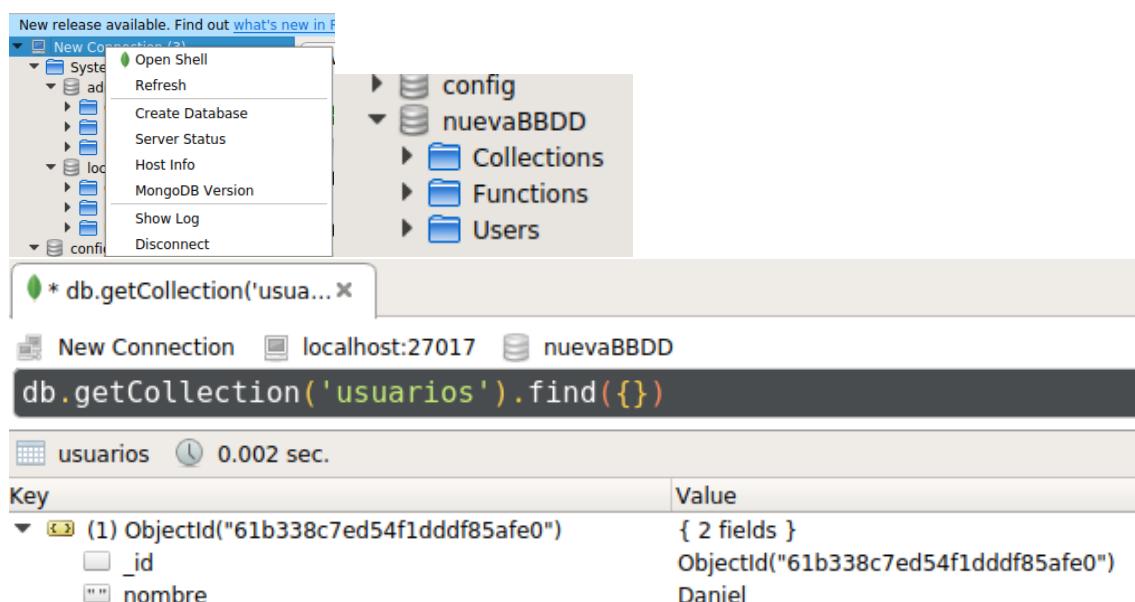
Informacion de la BBDD en uso:

- Db.stats()
  -

Para crear colecciones utilizamos el siguiente comando:

- Db.nombreTabla.insert({“clave”：“valor”})
  - Ejemplo: `> db.usuarios.insert({“nombre”：“Daniel”})  
WriteResult({ “nInserted” : 1 })`

Ahora esta bbdd estará disponible en Robo3T. Si no aparece forzaremos el refresco pulsando sobre la conexión con el botón derecho y seleccionar “refresh”



The screenshot shows the Robo3T interface. On the left, there's a sidebar with 'New Connection (1)' selected, showing options like 'Open Shell', 'Refresh', 'Create Database', 'Server Status', 'Host Info', 'MongoDB Version', 'Show Log', and 'Disconnect'. Below this, the main area shows a database named 'nuevaBBDD' with subfolders 'config', 'nuevaBBDD', 'Collections', 'Functions', and 'Users'. At the bottom, a terminal window displays the command 'db.getCollection('usuarios').find({})' and its execution results. The results table has columns 'Key' and 'Value', showing a single document with '\_id' (ObjectId("61b338c7ed54f1dddf85afe0")) and 'nombre' (Daniel).

Key	Value
❸ (1) ObjectId("61b338c7ed54f1dddf85afe0")	{ 2 fields }
❹ _id	ObjectId("61b338c7ed54f1dddf85afe0")
❺ nombre	Daniel

El **INSERT** anterior ha creado un archivo JSON con los datos que hemos indicado. Como se puede ver en la imagen, al crear una colección nos ha creado un id relacionado con los datos insertados. Insertar un dato nuevo es con el mismo procedimiento y quedaría de la siguiente forma:

Key	Value
▼ (1) ObjectId("61b338c7ed54f1dddf85afe0")	{ 2 fields }
└ _id	ObjectId("61b338c7ed54f1dddf85afe0")
└ nombre	Daniel
▼ (2) ObjectId("61b39560b2d6878e9c277a39")	{ 2 fields }
└ _id	ObjectId("61b39560b2d6878e9c277a39")
└ nombre	Núria

Como se puede apreciar, el cliente de Robo3T es más sencillo de seguir que con el terminal de MongoDB, durante el tema, usaremos más el entorno de Robo3T por la sencillez que presenta, no obstante, veremos al principio algunos comandos típicos de la Shell de Mongo.



En formato que hemos visto anteriormente es el formato de tabla, pero podemos mostrar los datos en formato de JSON, simplemente clicando en el icono situado a la parte superior derecha:

```
/* 1 */
{
  "_id" : ObjectId("61b338c7ed54f1dddf85afe0"),
  "nombre" : "Daniel"
}

/* 2 */
{
  "_id" : ObjectId("61b39560b2d6878e9c277a39"),
  "nombre" : "Núria"
}
```

En la shell de Mongo, para mostrar todas las colecciones de que disponemos en las BBDD:

- Show collections

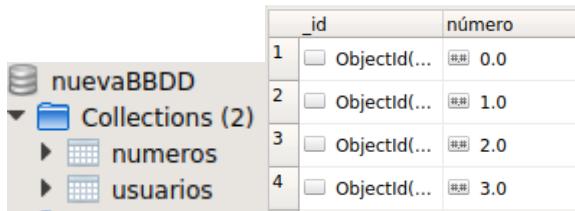
```
> show collections
  usuarios
```

Como curiosidad, y para los que controláis de JavaScript, debido a que utiliza un formato muy común para enviar información a las páginas web, los comandos que puede utilizar son idénticos a los de JavaScript:

```
> print ("Hola mundo")
Hola mundo
> var rango =[1, 2, 3, 4]
> rango
[ 1, 2, 3, 4 ]
```

Siguiendo con el parecido con JavaScript, vamos a insertar una nueva colección en el que insertaremos los números del 1 al 100. Para ello, recurriremos a un bucle for:

```
> for (i=0; i<100;i++){
... db.numeros.insert({"número":i})
...
WriteResult({ "nInserted" : 1 })
```



	_id	número
1	ObjectId(...)	0.0
2	ObjectId(...)	1.0
3	ObjectId(...)	2.0
4	ObjectId(...)	3.0

Con un count podemos comprobar la cantidad de entradas:

```
> db.numeros.count()
100
```

Para realizar queries en MongoDB utilizamos el comando find:

```
> db.numeros.find({"número":25})
{ "_id" : ObjectId("61b3999ab2d6878e9c277a53"), "número" : 25 }
```

Lo que devuelve mongo es un identificador único por defecto del dato devuelto, es la manera de hacer que queda campo sea único, de forma que realiza una especie de cifrado creando un objectId.

Es posible que necesitemos conocer el plan de ejecución del comando, para ello, en cualquier comando insertaremos lo siguiente:

```
> db.numeros.find({"número":25}).explain()
{
  "explainVersion" : "1",
  "queryPlanner" : {
    "namespace" : "nuevaBBDD.numeros",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "número" : {
        "$eq" : 25
      }
    },
    "queryHash" : "DDE29B54",
    "planCacheKey" : "BA659F4F",
    "maxIndexedOrSolutionsReached" : false,
    "maxIndexedAndSolutionsReached" : false,
    "maxScansToExplodeReached" : false,
    "winningPlan" : {
      "stage" : "COLLSCAN",
      "filter" : {
        "número" : {
          "$eq" : 25
        }
      }
    }
}
```

Donde se ejecuta el comando

Filtros aplicados

Condición de búsqueda

Como se ha explicado al principio del tema, Mongo tiene una gran capacidad de optimización, una de las acciones que podemos realizar es la creación de un índice:

```
> db.numeros.createIndex({"número":1})
{
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "createdCollectionAutomatically" : false,
  "ok" : 1
}
```

Lo que hemos hecho es decirle que nos ordene el campo número de forma incremental, creando así el índice, con lo que conseguiremos la optimización de consultas.

**Actividad 1:** Debes realizar capturas de pantalla de los pasos:

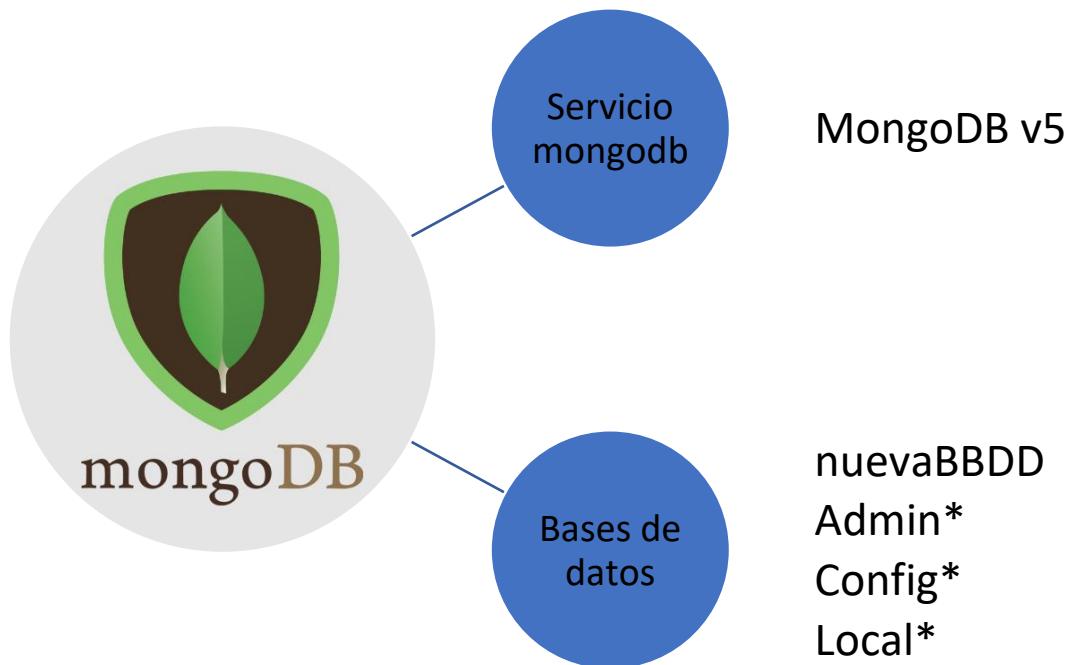
1. Crea una nueva BBDD llamada Universidad
2. Inserta 5 alumnos con nombre genérico y utilizando un índice para generar los datos.
3. Comprueba todas las BBDD del sistema
4. Comprueba las colecciones que hay en universidad

▼ (3) ObjectId("63d158d4421233e055489802")	{ 4 fields }
└ _id	ObjectId("63d158d4421233e055489802")
└ nombre	Alumno0
└ apellidos	apellido0
└ edad	0.0
▼ (4) ObjectId("63d158d4421233e055489803")	{ 4 fields }
└ _id	ObjectId("63d158d4421233e055489803")
└ nombre	Alumno1
└ apellidos	apellido1
└ edad	1.0
▼ (5) ObjectId("63d158d4421233e055489804")	{ 4 fields }
└ _id	ObjectId("63d158d4421233e055489804")
└ nombre	Alumno2
└ apellidos	apellido2
└ edad	2.0

## 2.4. Administración de las BBDD

Los comandos anteriores no están destinados a la gestión y creación de BBDD, por lo que este apartado está dedicado a este punto.

La estructura típica de una BBDD mongo es la siguiente:



### 2.4.1. Gestión de BBDD

Dentro del servidor de MongoDB están situadas las BBDD (\* indican que se crean por defecto) y ellas se gestionan mediante el servicio instalado en el S.O.

Como se indicó en el punto anterior, para crear una nueva BBDD:

- Use + bbdd:

```
> use concesionario
   o  Use concesionario switched to db concesionario
```

Como no hay ningún elemento, no aparece cuando hacemos show hasta que no insertamos elementos.

Para eliminar una BBDD:

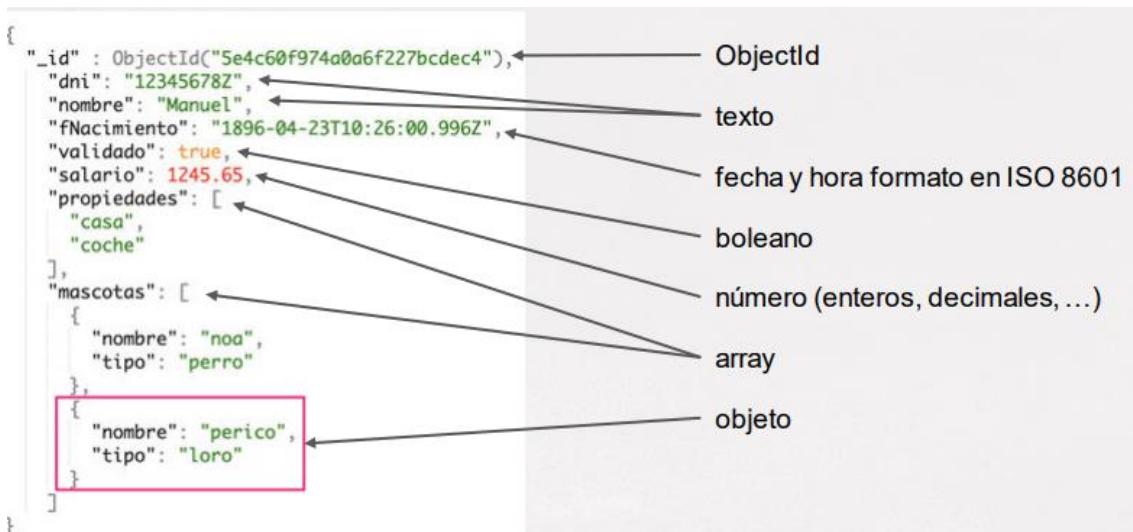
- Con la BBDD seleccionada db.dropDatabase():

```
> db.dropDatabase()
   o  { "ok" : 1 }
```

### 2.4.2. Tipos de datos

Tenemos dos categorías básicas, los simples y los datos compuestos

- Simples:
  - Números
  - Cadenas de texto
  - Fecha
  - Hora
  - Booleanos
- Compuestos:
  - Arrays
  - Objetos
  - Datos binarios
  - ObjectId: cada vez que creamos un registro nuevo, mongo asigna un campo de identificación para que sea único.
  - Expresiones regulares



**Actividad 2:** Realiza la creación de una BBDD. Debes realizar capturas de pantalla de los pasos:

1. Crea una base de datos llamada concesionario. Recuerda insertar al menos un documento en una colección que se llame coches. Como propiedades del documento json, puedes usar matrícula, marca, modelo, versiones (sport, confort), kms y fecha de matriculación. (Aunque veremos cómo insertar documentos JSON en próximas clases, puedes usar el comando db.insertOne(aquí va tu documento json))
2. Visualiza el listado de todas las bases de datos disponibles y de su contenido.
3. Elimina la base de datos creada y comprueba que ya no existe al pedir el listado.

## 2.5. Gestión de los datos

### 2.5.1. Inserción, consulta, actualización y eliminación

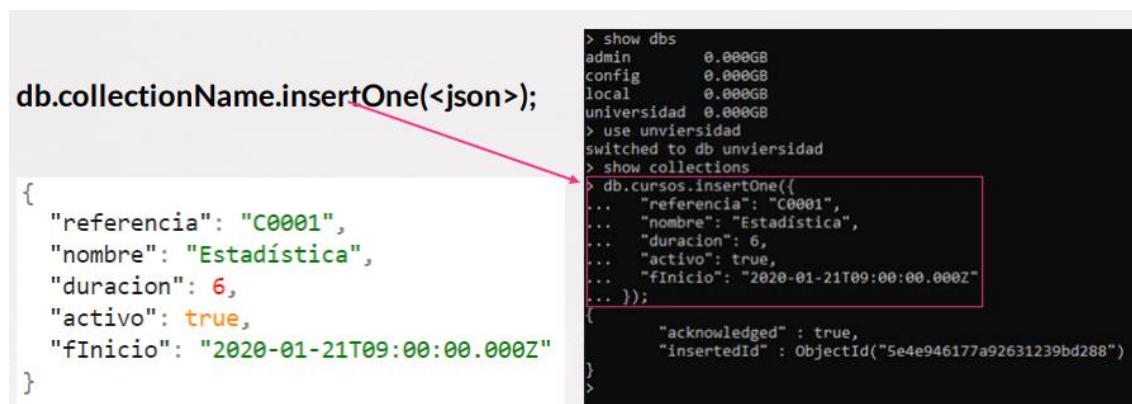
Insertar un solo documento:

- db.collectionName.InsertOne(<JSON>)

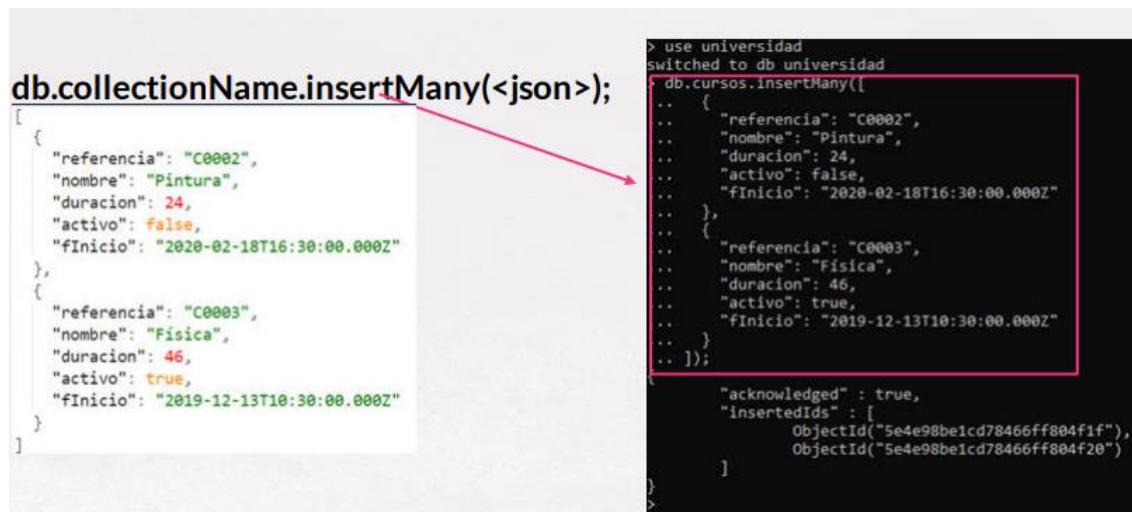
Insertar varios documentos:

- db.collectionName.InsertMany(<JSON>)

Ejemplo de inserción:



```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
universidad 0.000GB
> use universidad
switched to db universidad
> show collections
> db.cursos.insertOne({
...   "referencia": "C0001",
...   "nombre": "Estadística",
...   "duracion": 6,
...   "activo": true,
...   "fInicio": "2020-01-21T09:00:00.000Z"
... });
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e4e946177a92631239bd288")
}
>
```



```
> use universidad
switched to db universidad
> db.cursos.insertMany([
...   {
...     "referencia": "C0002",
...     "nombre": "Pintura",
...     "duracion": 24,
...     "activo": false,
...     "fInicio": "2020-02-18T16:30:00.000Z"
...   },
...   {
...     "referencia": "C0003",
...     "nombre": "Física",
...     "duracion": 46,
...     "activo": true,
...     "fInicio": "2019-12-13T10:30:00.000Z"
...   }
... ]);
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5e4e98be1cd78466ff804f1f"),
    ObjectId("5e4e98be1cd78466ff804f20")
  ]
}
>
```

**Actividad 3:** inserta una nueva colección llamada persona dentro de una bbdd llamada universidad. Los datos a insertan son:

- Nombre: María

- Apellidos: Suárez Manrique
- Especialidad: Biología
- esTitular: si
- esAsociado: no
- edad: 51

Una vez acabada, podéis consultar los datos usando el siguiente comando:

```
db.nombreColección.find().pretty()
```

```
> db.persona.find().pretty()
{
    "_id" : ObjectId("61b886de0fe07c6bee5abce5"),
    "nombre" : "María",
    "apellidos" : "Suárez Manrique",
    "especialidad" : [
        "biología"
    ],
    "esTitular" : true,
    "esAsociado" : false,
    "edad" : 51
}
```

Para realizar la importación de datos usamos el comando import, el cual importa datos desde un CSV o un JSON a nuestra BBDD. Hay que tener en cuenta que estos comando no se ejecutan en la Shell de mongo, sino en la propia del S.O.

- mongoimport --helpdb
- mongoimport --db myBBDD --collection nombreColeccion --jsonArray --file archivo.json

Vamos a insertar un archivo JSON en una nueva BBDD llamada baresIrlanda, este documento se encuentra en Aules como baresIrish.json

```
bigdata@BigData:~$ mongoimport --help
Usage:
  mongoimport <options> <connection-string> <file>
```

```
bigdata@BigData:~/Descargas$ mongoimport --db irlanda --collection bares --jsonArray --file baresIrish.json
2021-12-14T19:10:58.627+0100      connected to: mongodb://localhost/
2021-12-14T19:10:58.671+0100      7 document(s) imported successfully. 0 document(s) failed to import.
bigdata@BigData:~/Descargas$
```

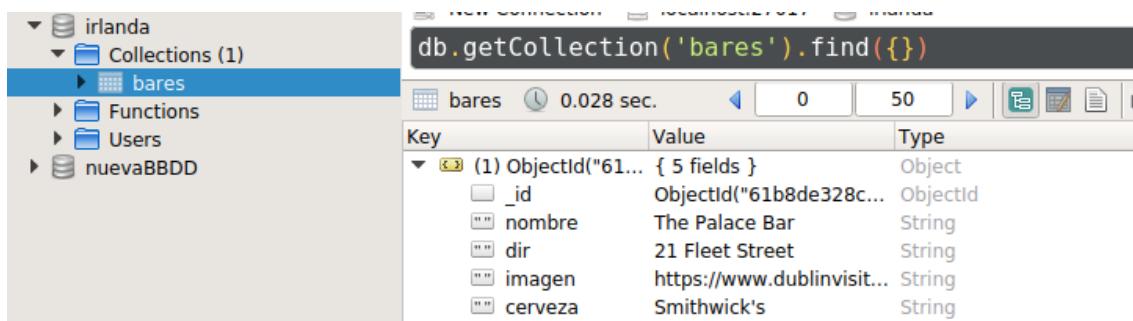
```
> show dbs
admin      0.000GB
config     0.000GB
irlanda    0.000GB
local      0.000GB
nuevaBBDD 0.000GB
```

Una vez se han importado de manera satisfactoria podemos abrir mongo:

- use irlanda
- db.bares.count()
- db.bares.find({"nombre":"No name bar"}) → realizamos una búsqueda con una condición

```
> db.bares.count()
7
> db.bares.find({"nombre":"No name bar"})
{ "_id" : ObjectId("61b8de328c06717b23f11174"), "nombre" : "No name bar", "dir" : "3 Fade Street",
  "imagen" : "https://media-cdn.tripadvisor.com/media/photo-s/0a/e2/46/dc/no-name-milatos.jpg",
  "cerveza" : "Murphy's" }
```

Podemos ver con el interfaz gráfico que los datos se han insertado correctamente:



Key	Value	Type
1 ObjectId("61b8de328c...") { 5 fields }		Object
_id	ObjectId("61b8de328c...")	ObjectID
nombre	The Palace Bar	String
dir	21 Fleet Street	String
imagen	https://www.dublinvisit...	String
cerveza	Smithwick's	String

Con el comando de la imagen, podemos visualizar todos los datos que tenemos en la colección, en esa misma sección, podemos escribir los comandos que necesitemos para trabajar con los datos, por ejemplo, vamos a contar el número de elementos que hemos insertado, para ejecutar el comando hemos de apretar F5 o el botón de play del entorno.

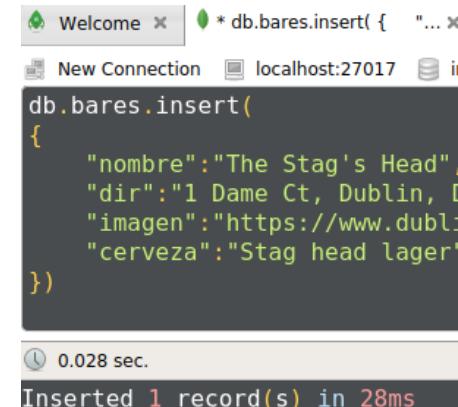
```
db.bares.count()
0.018 sec.
7
```

**Actividad 4:** busca el bar que tenga la cerveza llamada Galway Hooker en el entorno gráfico con la dirección 2 Suffolk Street.

### Inserción de datos

Para insertar valores se utilizar el método `insert({": ":"}, ...)`. Insertaremos un nuevo bar:

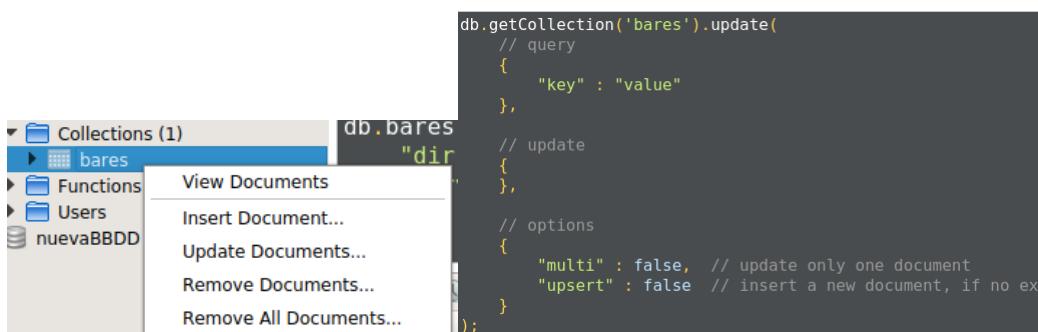
- Nombre: The Stag's Head
- Dir: 1 Dame Ct, Dublin, D02 TW84
- Imagen: <https://static.rutasdeescape.com/wp-content/uploads/2018/06/2-iStock-The-Stag%280%99s-Head-1024x683.jpg>
- Cerveza: Stag's head lager



```
Welcome * db.bares.insert( ...
New Connection localhost:27017 ...
db.bares.insert(
{
  "nombre": "The Stag's Head",
  "dir": "1 Dame Ct, Dublin, D02 TW84",
  "imagen": "https://static.rutasdeescape.com/wp-content/uploads/2018/06/2-iStock-The-Stag%280%99s-Head-1024x683.jpg",
  "cerveza": "Stag's head lager"
})
0.028 sec.
Inserted 1 record(s) in 28ms
```

### Actualización de datos

Una de las cosas que ofrece este cliente, es que si queremos actualizar la tabla nos ofrece una ayuda. Iremos a la tabla y con el botón derecho seleccionaremos actualizar donde podremos ver la ayuda en cuanto a la estructura:



```
db.getCollection('bares').update(
  // query
  {
    "key" : "value"
  },
  // update
  {
    ...
  },
  // options
  {
    "multi" : false, // update only one document
    "upsert" : false // insert a new document, if no existing document is found
  }
);
```

Para actualizar los datos usamos el método:

- `update({query},{$set{"": ""}})`

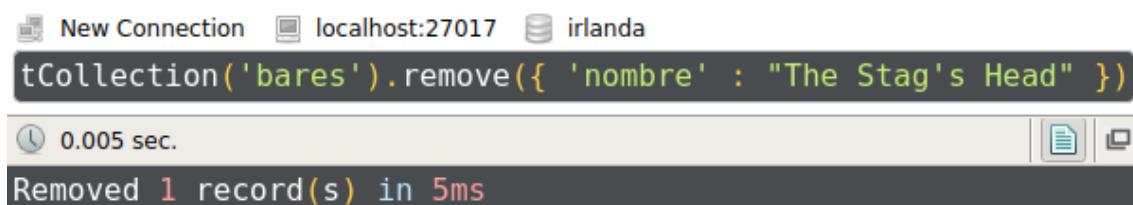
El operador `$set` reemplaza el valor de un campo con el valor especificado.

Vamos a modificar el campo de la cerveza del último bar añadido, en lugar de Stag's head lager, vamos a insertar Estrella Galicia el camino. Usando la plantilla que nos ofrece roboT3 quedará así:

```
db.getCollection('bares').update(  
    // query  
    {  
        "nombre" : "The Stag's Head"  
    },  
    // update  
    {  
        $set:{ "cerveza": "Estrella Galicia El Camino"}  
    },  
    // options  
    {  
        "multi" : false, // update only one document  
        "upsert" : false // insert a new document, if no exi  
    }  
);  
▼ (8) ObjectId("61b9c83... { 5 fields }  
  _id ObjectId("61b9c833cab39684...  
  nombre The Stag's Head  
  dir 1 Dame Ct, Dublin, D02 TW84  
  imagen https://www.dublintown.ie/wp...  
  cerveza Estrella Galicia El Camino
```

## Eliminar de datos

Para eliminar los datos podemos usar la ayuda del entorno igual que antes y utilizando el método con el método **remove({query})**



The screenshot shows the MongoDB Compass interface. The top navigation bar has tabs for 'New Connection', 'localhost:27017', and 'irlanda'. Below the navigation, a command line window displays the command: `tCollection('bares').remove({ 'nombre' : "The Stag's Head" })`. To the right of the command window, a status bar shows '0.005 sec.' and 'Removed 1 record(s) in 5ms'.

### 2.5.2. Adición de campos al documento

Con el método **update** podemos modificar uno o varios documentos existentes en una colección. Dicho método puede modificar campos específicos de un documento o varios, así como añadir nuevos en función de cómo se informen los diferentes parámetros que contiene.

- Con el operador **\$set** se puede añadir un campo al documento o actualizar el valor de uno ya existente
- Con el operador **\$inc** podemos incrementar o decrementar un campo numérico.

Dentro de los parámetros que podemos encontrar en el update destacamos:

- **upsert**: El valor por defecto es false, es decir, no inserta un nuevo documento cuando no encuentre ninguna coincidencia con el valor de búsqueda. Si ponemos true, entonces creará un nuevo documento con el campo y valor buscado.
- **multi**: El valor predeterminado es false, en este estado, solo actualiza un documento. Si es true entonces actualiza todos los documentos que cumplan con los criterios de búsqueda.

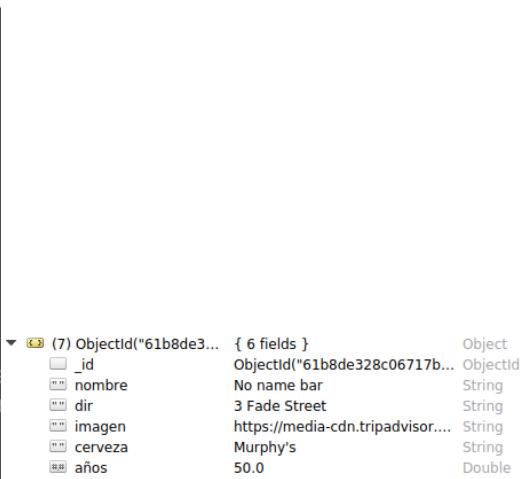
Lo primero que vamos a hacer es añadir un nuevo campo a los bares, este campo será el tiempo que llevan abiertos. Podemos hacerlo de varias formas, la primera es añadir un nuevo campo a

todos los elementos con el mismo número y luego actualizarlos. Otra opción es ir uno a uno actualizándolo. Al haber 7 bares deberemos insertar 7 campos, los datos serán los siguientes:

- 1: 63
- 2: 57
- 3: 84
- 4: 103
- 5: 36
- 6: 92
- 7: 98

Vamos a elegir el primer método para ver como añadir un campo a todos los elementos:

```
db.getCollection('bares').update(  
    // query  
    {  
    },  
  
    // update  
    {  
        $set:{"años":50}  
    },  
  
    // options  
    {  
        "multi" : true, // update  
        "upsert" : false // insert  
    }  
);
```



Ahora asignaremos cada edad a cada uno de los campos:

```
db.getCollection('bares').update(  
    // query  
    {  
        "nombre" : "The Palace Bar"  
    },  
  
    // update  
    {  
        $set:{"años":63}  
    },  
  
    // options  
    {  
        "multi" : false, // update  
        "upsert" : false // insert  
    }  
);
```

Otra opción es utilizar \$inc para actualizar el número.

**Actividad 5:** actualiza el campo años en cada uno de los bares.

Si la búsqueda que realizamos no da ningún resultado, mediante upsert podemos indicar que nos añada los elementos que estamos escribiendo.

### 2.5.3. Filtrar, ordenar y contar

Con el comando `find` podemos decidir qué campos mostrar o no en nuestras consultas a través de 1 ó 0. `find({query},{“campo”:1})`. Como se puede ver, con el 1 indicamos que solo queremos el campo. Como ejemplo:

```
New Connection localhost:27017 irlanda
db.bares.find({}, {"nombre":1})

bares 0.002 sec.
_id nombre
1 ObjectId(...) The Palace Bar
2 ObjectId(...) Oliver St. John Gogarty
```

Si queremos que no nos aparezca el ID:

```
db.bares.find({}, {"_id":0, "nombre":1})

bares 0.003 sec.
nombre
1 The Palac...
```

El único campo que se puede indicar que no se muestre con el valor 0 es el campo ID.

`find.sort()` nos permite ordenar las consultas de la forma `sort({“campo”:1})` donde toma el valor 1 para ordenar de forma ascendente y -1 descendente.

```
New Connection localhost:27017 irlanda
db.bares.find().sort({"nombre":1})
```

Con `find.limit(numero)` se puede limitar los documentos devueltos. Ejemplo: mostrar solo un resultado:

```
db.bares.find().limit(1)
```

`find.skip(numero)` nos permite saltar un número determinado de documentos. Ejemplo de salto cada 3:

```
New Connection localhost:27017 irlanda
db.bares.find().limit(2).skip(1)
```

A través de `count({condición})` se pueden contar los elementos que cumplan una determinada condición.

```
New Connection localhost:27017 irlanda
db.bares.count()

0.002 sec.
7
```

### 2.5.4. Selectores de consulta

Igual que en bash de Linux, utilizamos los siguientes operadores para realizar comparaciones

- **\$eq** devuelve documentos con campos cuyo valor es igual al valor de consulta.
- **\$gt/gte** devuelve documentos con campos cuyo valor es mayor/mayor o igual al valor de consulta.
- **\$in** devuelve documentos con campos cuyo valor es igual a algunos de los valores de consulta especificados en un array.
- **\$lt/lte** devuelve documentos con campos cuyo valor es menor/igual al valor de consulta
- **\$ne** devuelve todos los documentos con campos cuyo valor no coincide con el valor de consulta.
- **\$nin** devuelve documentos con campos cuyos valores no coincide con ninguno de los valores de un array especificado en la consulta.

#### Actividad 6:

- Busca bares que tengan más de 70 años
- Bares con un aforo igual a 50

### 2.5.5. Operadores lógicos

- **\$and**, dentro de una cláusula de consulta, devuelve todos los documentos que coinciden con todas las condiciones marcadas.
- **\$not**, invierte el efecto de una cláusula de consulta, y devuelve todos los documentos que NO coinciden con la expresión marcada.
- **\$nor**, dentro de una cláusula de consulta, devuelve todos los documentos que no coinciden con ninguna de las condiciones marcadas.
- **\$or**, dentro de una cláusula de consulta, devuelve todos los documentos que coinciden con algunas de las condiciones marcadas.
- **\$exists**, devuelve los documentos que tengan el campo especificado.
- **\$type**, devuelve documentos si un campo es de un determinado tipo.

```
New Connection localhost:27017 irlanda
db.bares.find({$and:[{"nombre":"O'Neill's"}, {"cerveza":"Galway Hooker"}]})
```

### 2.5.6. Operadores de evaluación de consultas

- **\$expr**, nos permite utilizar expresiones de agregación dentro del lenguaje de consulta.  
.find({\$expr:{<selector>[<campo1>,<campo2>]}
- **\$regex**, devuelve documentos cuyos campos coincidan con una expresión regular especificada. Mongo proporciona la búsqueda de patrones de consultas en las cadenas a través de las expresiones regulares compatible con PERL (PCRE, Perl Compatible Regular Expressions).Tenemos diferentes tipos de notación:
  - {<field>:{\$regex:/pattern/,\$options:'<options>'}}
  - {<field>:{\$regex:'pattern',\$options:'<options>'}}
  - {<field>:{\$regex:/pattern/,<options>}}
- **\$text**, realiza búsqueda de texto.
- **\$where**, devuelve documentos que cumplan con una determinada expresión de Javascript.

Para el siguiente ejemplo cargaremos el archivo llamado canciones.json. Podemos buscar los campos de tipo texto usando el siguiente comando:

```
db.canciones.find({"artista":"nombre del artista a buscar"})
```

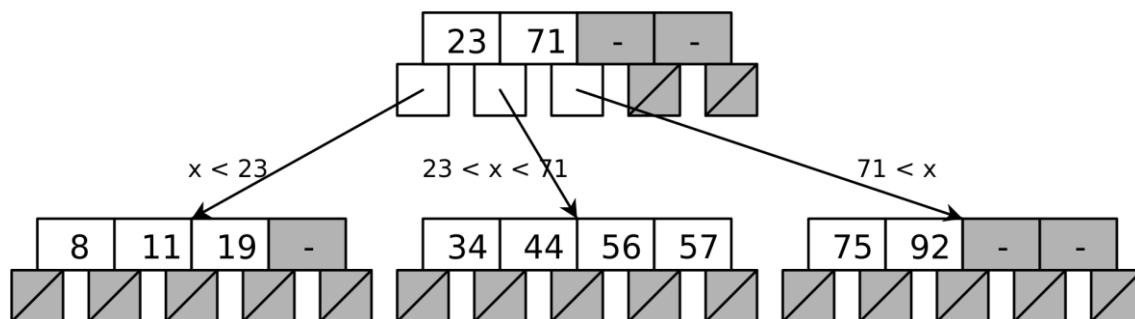
Siempre que trabajemos con texto es importante crear un índice. En las bases de datos relacionales, los índices son algo indispensable. Sería inconcebible consultar una tabla con millones de registros si no hemos configurado al menos un índice.

Con MongoDB pasa lo mismo. No podemos pensar en tener una colección con millones de documentos, sin tener índices sobre uno o varios campos. La diferencia entre realizar una consulta sobre campo con índice, y realizarla sin él, puede ser abismal.

Imagina una colección que tiene cientos de documentos. Para encontrar documentos que coincidan con un filtro en particular, **MongoDB** tendría que comparar el filtro con todos los documentos que están presentes en la colección, lo cual es tedioso y requiere mucho tiempo. En lugar de indexar documentos (campos), se reducirá significativamente el tiempo de búsqueda.

Los índices son estructuras de datos que se empaquetan con un conjunto de datos parcial dentro de sí mismo. El índice almacena el valor de un campo en particular en los documentos de manera ordenada que admite operaciones como comparadores, ecuadores y consultas basadas en rangos en orden secuencial.

Los índices en MongoDB se generan en forma de Árbol-B o B-Tree.



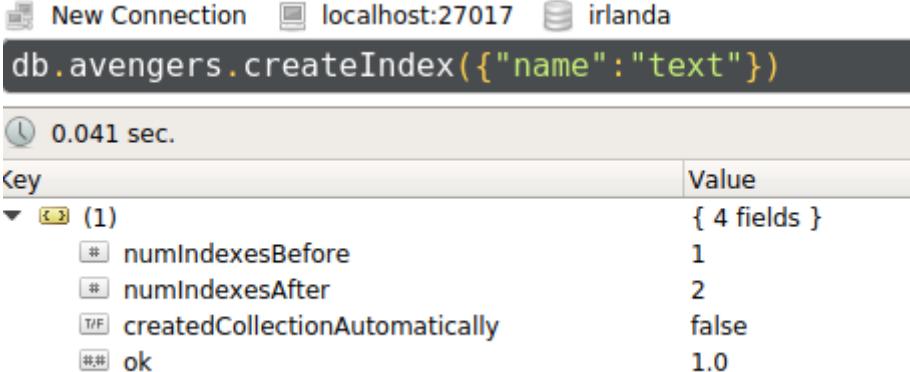
Es decir, que los datos se guardan en forma de árbol, pero manteniendo los nodos balanceados. Esto incrementa la velocidad a la hora de buscar y también a la hora de devolver resultados ya ordenados. De hecho, MongoDB es capaz de recorrer los índices en ambos sentidos, por lo que, con un solo índice, podemos conseguir ordenación tanto ascendente como descendente.

**Actividad 7:** importa el archivo canciones.json en una BBDD llamada música. Después crea un índice de tipo texto en función del artista. Una vez creado el índice, realiza una búsqueda de un artista.

Si no tenemos creado un índice:

```
Error: error: {
  "ok" : 0,
  "errmsg" : "text index required for $text query",
  "code" : 27,
  "codeName" : "IndexNotFound"
}
```

### Ejemplo de creación en avengers



The screenshot shows the MongoDB Compass interface. A connection named 'localhost:27017' is selected, and the database is 'irlanda'. In the query bar, the command `db.avengers.createIndex({ "name": "text" })` is entered. Below the command, the results show a duration of 0.041 sec. A table displays the key and value of the created index:

Key	Value
numIndexesBefore	1
numIndexesAfter	2
createdCollectionAutomatically	false
ok	1.0

Realiza una búsqueda de texto en el contenido de los campos indexados con un índice de texto. El comando `$text` nos permite realizar búsquedas de texto. Además, posee algunas funciones como por ejemplo `$search`, que nos permite buscar más de un término con un solo comando.

La expresión `$text` tiene la siguiente sintaxis:

```
{  
  $text:  
    {  
      $search: <string>,  
      $language: <string>, Opcional  
      $caseSensitive: <boolean>,Opcional  
      $diacriticSensitive: <boolean> Opcional  
    }  
}
```

Una cadena de términos que MongoDB analiza y usa para consultar el índice de texto. MongoDB realiza una búsqueda lógica o de los términos a menos que se especifique como una frase entre “`\\"frase\\"`”, con la opción “`-text`” se elimina de la búsqueda

`$language: <string>`, Opcional

Determina la lista de palabras de detención para la búsqueda y las reglas para el stemmer y el tokenizer. Si no se especifica, la búsqueda usa el idioma predeterminado del índice.

`$caseSensitive: <boolean>`,Opcional

Búsqueda sensible a minúsculas y mayúsculas.

`$diacriticSensitive: <boolean>` Opcional

Búsqueda sensible diacrítica, si tiene en cuenta las tildes.

```
}  
}  
  
Ejemplo 1, donde podemos buscar dos términos en un solo comando:  
db.canciones.find({$text:{$search:" Bob Dylan Bonnie Tyler"}})
```

```
db.avengers.find({$text:{$search: "Thor Captain America"}})
```

Key	Value
▼ (1) ObjectId("61bcbb3cd74a201f9bf06d1b")	{ 5 fields }
_id	ObjectId("61bcbb3cd74a201f9bf06d1b")
name	Captain America
age	103
secretIdentity	Steve Rogers
powers	[ 3 elements ]
▼ (2) ObjectId("61bcbb3cd74a201f9bf06d1f")	{ 5 fields }
_id	ObjectId("61bcbb3cd74a201f9bf06d1f")
name	Thor
age	1500
secretIdentity	Thor Odinson
powers	[ 3 elements ]

**Ejemplo 2:** si quisiéramos buscar todas las coincidencias de un término, pero que no aparezca el segundo término usaríamos “-“ antes del segundo:

```
db.canciones.find({$text:{$search:" Bob Dylan -Bonnie Tyler"}})
```

Key	Value
▼ (1) ObjectId("61bcbb3cd74a201f9bf06d1f")	{ 5 fields }
_id	ObjectId("61bcbb3cd74a201f9bf06d1f")
name	Thor
age	1500
secretIdentity	Thor Odinson
powers	[ 3 elements ]

**Ejemplo 3:** para poner una frase completa, donde hayan acentos o signos de puntuación que puedan dar problemas necesitamos el valor de escape “\”: db.canciones.find({\$text:{\$search:"\” Bob Dylan \”}}). Debemos tener en cuenta que la estructura del comando es:

- Primera “ → indica que tendremos un elemento tipo string
- “\ → esta combinación indica que si hay caracteres especiales en el string se tengan también en cuenta en la búsqueda
- \” → finalizamos el string de caracteres especiales.
- “ → final del string

Key	Value
▼ (1) ObjectId("61bcbb3cd74a201f9bf06d1e")	{ 5 fields }
_id	ObjectId("61bcbb3cd74a201f9bf06d1e")
name	Iron-Man
age	42
secretIdentity	Antony Stark
powers	[ 2 elements ]
[0]	Technological armor
[1]	Genius

**Ejemplo 4:** Si queremos que en la búsqueda tengan en cuenta los acentos es necesario poner la opción \$diacriticSensitive: db.canciones.find({\$text:{\$search: "\” Like a Rolling Stone"\”, \$diacriticSensitive: true}})

Con Text Search Score, la búsqueda de texto asigna una puntuación a cada documento que contiene el término de búsqueda en los campos indexados. La puntuación determina la relevancia de un documento para una consulta de búsqueda. Esto es interesante en el campo del big data y la entrada masiva de datos. Pensad en el score que hicimos en el tema 1 con los datos de Twitter y la relevancia de los resultados.

Para cada campo indexado en el documento, MongoDB multiplica el número de coincidencias por el peso y suma los resultados. Con esta suma, MongoDB calcula la puntuación del documento. El peso predeterminado es 1 para los campos indexados.

Un ejemplo, que para usarlo descargaremos la colección articulos.json de la carpeta de Aules:

- db.articulos.find({\$text:{\$search:"baking"}},{score:{\$meta:"textScore"}})

### 2.5.7. Operadores con arrays

Ahora vamos a trabajar con lista de elementos dentro de las propias colecciones, también les podemos llamar Array. Tienen una estructura similar a esta:

```
[  
  {  
    "name" : "Captain America",  
    "age" : 103,  
    "secretIdentity" : "Steve Rogers",  
    "powers" : [  
      "Super Soldier Serum",  
      "Tactician and field commander",  
      "Artits"  
    ]  
  },  
  {  
    "name" : "Iron-Man",  
    "age" : 42,  
    "secretIdentity" : "Antony Stark",  
    "powers" : [  
      "Technological armor",  
      "Genius"  
    ]  
  },  
]
```

Vamos a importar los datos que tenemos en el documento superheroes.json. Este archivo lo vamos a insertar dentro de la BBDD de irlanda (aunque no tengan nada que ver)

Algunos de los comandos que podemos hacer con arrays:

- \$all Arrays de coincidencias que contienen todos los elementos especificados en la consulta, similar al AND.
- \$elemMatch selecciona documentos si el elemento en el campo del array coincide con todas las condiciones (querys) especificadas.
- \$size Selecciona documentos si el campo array tiene un tamaño especificado.
- Con el operador \$slice muestra partes de un array y tiene dos formas
  - clave: {\$slice:n}, muestra las n primeras componentes del array. Si n es negativo muestra las n últimas.
  - Clave:{\$slice:[n1,n2]}, n1 es la cantidad de valores que se saltan y n2 la cantidad de valores consecutivos a mostrar

Vamos a buscar todos los héroes que tengan Superstrength y Mjolnir dentro del array de Powers.

**Actividad 8:** realiza las siguientes búsquedas:

- Todos los poderes que tengan superfuerza y el Mjolnir

### 2.5.8. Modificadores de actualización de arrays

- El operador `$` actúa como marcador de posición para actualizar el primer elemento que coincide con la condición de consulta.
- El operador `$[]` actúa como marcador de posición para actualizar todos los elementos en un array para los documentos que coinciden con la condición de consulta.
- El operador `$[ <identificación> ]` Actúa como marcador de posición para actualizar todos los elementos que coinciden con la condición arrayFilters para los documentos que coinciden con la condición de consulta.
- El parámetro arrayFilters especifica qué elementos a modificar en un campo del array.
- **\$addToSet** Agrega elementos a un array solo si todavía no existen en el conjunto.
- Para especificar un `<campo>` en un documento incrustado o un array se usa la notación punto.
- No garantiza un orden particular de elementos en el array modificado.
- Si el valor es un array, **\$addToSet** agrega el array completo como un solo elemento.
- **\$pop** Elimina el primer o el último elemento de un arreglo
- **\$push** Agrega un elemento a un array
- **\$pull** Elimina todos los elementos de la matriz que coinciden con una consulta especificada
- **\$pullAll** Elimina todos los valores coincidentes de un arreglo

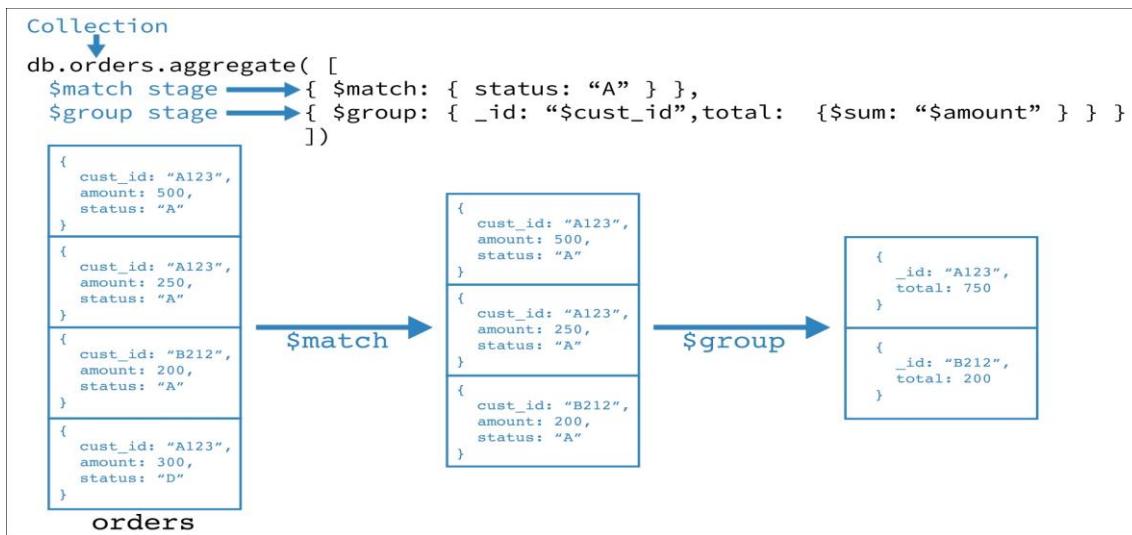
## 2.6. Agregaciones y operaciones pipeline

Las operaciones de agregación procesan registros de datos y devuelven el resultado calculado. Estas operaciones agrupan **valores de varios documentos** y pueden realizar una variedad de operaciones en los datos agrupados para obtener un único resultado.

El marco de agregación de MongoDB se basa en el concepto de canalización de procesamiento de datos. Los documentos entran en una “tubería” de etapas que transforma los documentos en un resultado.

En el método `db.collection.aggregate` y el método `db.aggregate`, las etapas de canalización (pipes) aparecen en un array. Los documentos pasan por las etapas en secuencia. Todas las etapas excepto `$out` y `$geoNear` pueden aparecer varias veces en una tubería (pipes).

Ejemplo pipeline:



En el ejemplo anterior, primero hemos buscado las coincidencias de Status: A, lo cual nos ha devuelto 3 resultados, pero sería interesante sumar el total de los campos de todos los Status, por ello realizamos la operación \$group para unir todos los estatus idénticos y después aplicamos la operación de suma (\$sum).

Para los ejemplos, vamos a importar el archivo de articulos.json:

Veamos algunos ejemplos ilustrando con Robo3t para que sea más legible

- Vamos a hacer un pipeline:
  - Etapa 1 \$match : Buscaremos los datos de “Autor”: “xyz”
  - Etapa 2 \$group: Agrupamos el resultado anterior y los contamos.

Captura de robot3 para ilustrarlo mejor

```
db.articulos.aggregate([
  {$match:{"author":"xyz"}},
  {$group:{"_id":"publicados", count:{$sum:1}}}
])
```

El resultado es que sustituimos el valor de \_id por publicados. No es posible añadir un nuevo campo.

Key	Value
▼ (1) publicados	{ 2 fields }
## _id	publicados
## count	3.0

- Números de artículos con más de 50 visitas por autor

```
db.articulos.aggregate([
  {$match:{"views":{$gt:50}}},
  {$group:{_id:"$author", count:{$sum:1}}}
])
```

- Con el comando \$out escribimos el resultado de la agregación pipeline a una colección

```
db.articulos.aggregate([
{$group:{"_id":"$author","libros":{$push:"$subject"}},},
{$out:"librosAutores"}
])
```

- Añadimos \$sort

```
db.articulos.aggregate([
{$group:{"_id":"$author","libros":{$push:"$subject"}},},
{$sort: {"_id": -1}},
{$out:"librosAutores"}
])
```

En la primera etapa \$match filtra los documentos por el campo estado y pasan a la próxima fase aquellos documentos que tienen el estado = "A"

En la segunda etapa el \$group filtra los documentos por el campo cust\_id para calcular la suma para cada clave. Una vez entendido cómo funcionan las diferentes fases para la agregación veamos unos ejemplos donde utilizaremos las siguientes:

- **\$group** agrupa los documentos de entrada por una expresión de identificador especificada. Consumo todos los documentos de entrada y genera un documento por cada grupo distinto.
- **\$project** pasa los documentos con los campos solicitados a la siguiente etapa. Los campos especificados pueden ser campos existentes de los documentos de entrada o campos recién calculados.
- **\$lookup** realiza una combinación left join con otra colección.
- **\$out** escribe el resultado de la agregación pipeline a una colección.
- **\$set** agrega nuevos campos a los documentos
- **\$unset** excluye campos de los documentos

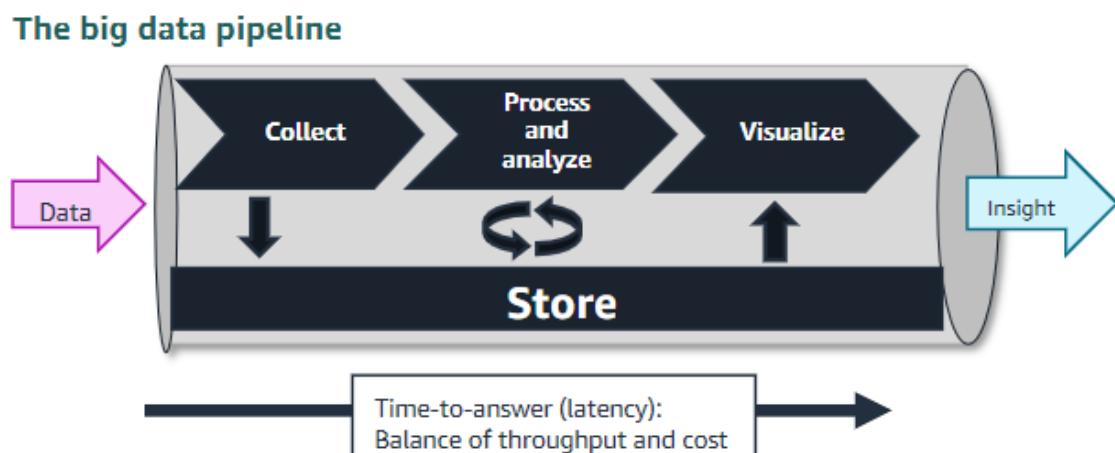
**Actividad 9:** realiza los mismos ejemplos del documento.

### 3. El sistema de almacenamiento Cloud

El análisis de grandes conjuntos de datos requiere una capacidad de cálculo significativa que puede variar en tamaño en función de la cantidad de datos de entrada, el tipo de análisis y el tiempo necesario para completarlo. Esta característica de las cargas de trabajo de Big Data requiere de un modelo de computación que permita adecuar las necesidades en función de la demanda, pudiendo ampliarse los recursos en función de la demanda.

A medida que cambian los requisitos, la computación en la nube permite fácilmente redimensionar el entorno horizontal o verticalmente. Además, los servicios y el hardware están gestionados por las empresas suministradoras y, por tanto, se encuentran en un estado de alta disponibilidad.

Un flujo habitual de datos Cloud podría ser el siguiente:



Los siguientes puntos se especificarán almacenamientos ubicados en AWS (Amazon Web Services)

#### 3.1. Amazon S3 Estándar (S3 Estándar)

Amazon S3 ofrece una variedad de clases de almacenamiento entre las cuales puede elegir en función de los requisitos de acceso a los **datos, resiliencia y costos de sus cargas de trabajo**. Las clases de almacenamiento de S3 se crearon específicamente para brindar el menor costo posible de almacenamiento para los diferentes patrones de acceso.

S3 Estándar ofrece almacenamiento de objetos de alta durabilidad, disponibilidad y rendimiento para datos a los que se obtiene acceso con frecuencia.

Dada su baja latencia y alto nivel de procesamiento, el tipo S3 Estándar es apropiado para una amplia variedad de casos de uso, como aplicaciones en la nube, sitios web dinámicos, distribución de contenido, aplicaciones para dispositivos móviles y videojuegos, y el análisis de big data.

#### 3.2. AWS Glacier

Amazon S3 Glacier es un servicio de almacenamiento en la nube seguro para archivar datos y realizar copias de seguridad a largo plazo. El servicio está optimizado para datos a los que se

obtiene acceso con poca frecuencia para los que sea aceptable tener un tiempo de recuperación de varias horas.



### 3.2.1. Características de AWS Glacier

- Es un almacén seguro, duradero y de bajo coste
- Puede guardar gran cantidad de datos (escala de PB) durante un tiempo prolongado. Además, es un almacén, es un almacenamiento de ficheros en frío, es decir, no se pueden consultar en tiempo real.
- El almacén (Vault):
  - Guardar archivos
  - Al crearlo se debe especificar un nombre y una región de uso
  - Puedes crear políticas para tus almacenes. Por ejemplo, bloquear notificaciones de los datos contra el acceso por vía M, no permitir borrar ficheros durante un año, etc.
- Almacén bloqueado (Vault locked):
  - Sirve para construir almacenes de datos con una configuración específica que no se podrá cambiar.
  - Esto es útil para cumplir normativas en cuanto a algún almacén seguro de datos durante un cierto tiempo.
  - Políticas creadas con IAM (Seguridad de AWS). Una vez que se crean las políticas para ese almacén, tendrás 24 horas para revisarlas, ya que pasado ese tiempo ya no se podrá modificar y quedará así de forma permanente.
  - Con lo cual, os aseguro, nos aseguramos así que cumplimos la normativas, porque ese almacén está bloqueado con esas propiedades y no se podrá cambiar.
- Archivos
  - Glacier puede almacenar cualquier tipo de formato: fotos, videos, documentos, etc. Cada archivo tiene un multiplicador único y una descripción que puede ser opcional.
  - Al crear el archivo se crea un identificador único.
  - Trabajo:
    - los trabajos pueden servir para consultar archivos, recuperarlos o para obtener el inventario de un almacén, un almacén o un contenedor.
    - Es posible tener varios trabajos en ejecución al mismo tiempo.
- Notificación

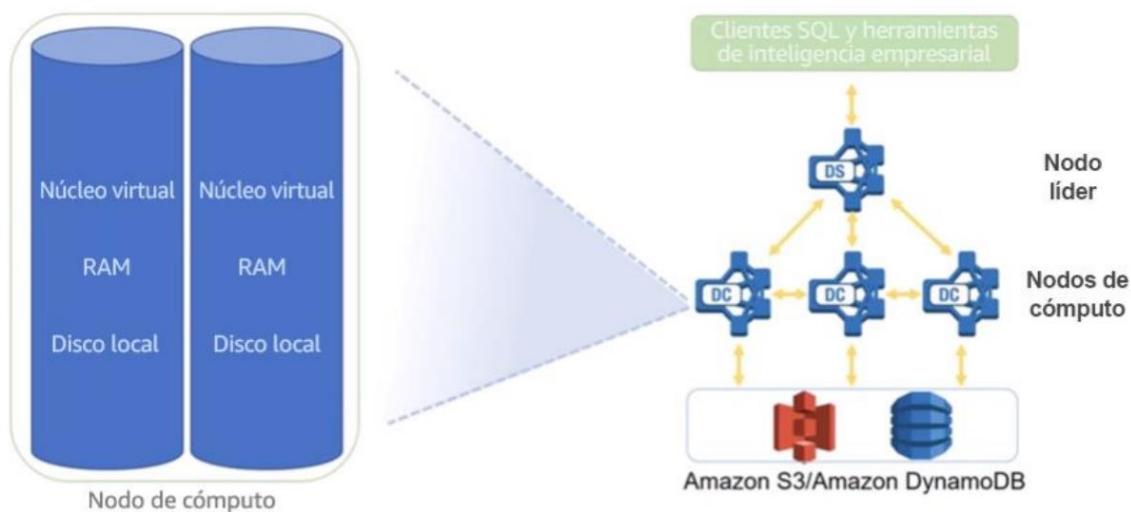
- Los trabajos de Glacier tardan un tiempo en completarse. Glacier tiene un mecanismo de notificación para avisar al completarse.

Tutorial de cómo usar Glacier

- [Creación de un almacén de S3 Glacier](#)
- [Carga de un archivo en S3 Glacier](#)
- [Carga multipart en S3 Glacier](#)

### 3.3. AWS RedShift (datawarehousing)

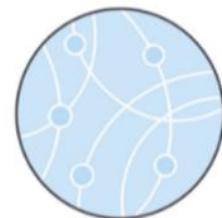
Redshift es un servicio de datawarehousing, con soporte para SQL, que se basa en clústeres de computación paralela, que pueden integrarse en clientes convencionales o producir grandes cantidades de información que se almacenen en servicios complementarios:



Normalmente se utilizará en escenarios donde abundan los clientes con consultas complejas, modelado, Business Intelligence, etc, interesante para todos los tamaños de empresa pero en especial aquellos que no se podrían permitir invertir en sistemas de datawarehousing convencional:

#### Almacén de datos de la compañía (EDW)

- Migrar a un ritmo cómodo para los clientes
- Experimentar sin grandes costos o compromisos iniciales
- Responder más rápidamente a las necesidades del negocio



#### Big data

- Precios bajos para clientes pequeños
- Servicio administrado para una implementación y mantenimiento sencillo
- Mayor enfoque en los datos, menor enfoque en la administración de la base de datos



Aunque es un clúster basado en Postgres, debido a la paralelización y al rendimiento I/O hay diferencias operativas importantes: compatibilidad SQL, restricciones como clave primaria solo informativa, almacenamiento columnar, compresión, etc

## Designed for I/O Reduction

Columnar storage

Data compression

No enforced constraints

```
CREATE TABLE deep_dive (
    aid      INT          ENCODE LZO
    ,loc     CHAR(3)      ENCODE BYTEDICT
    ,dt      DATE         ENCODE RUNLENGTH
);
```



En el caso de Redshift lo más crítico en el rendimiento son las particiones de datos para la paralelización de las consultas. En general, tiene un rendimiento muy bueno de forma automática repartiendo la carga homogéneamente, pudiendo realizarse de forma manual si apreciamos una ventaja:

### Distribution Key

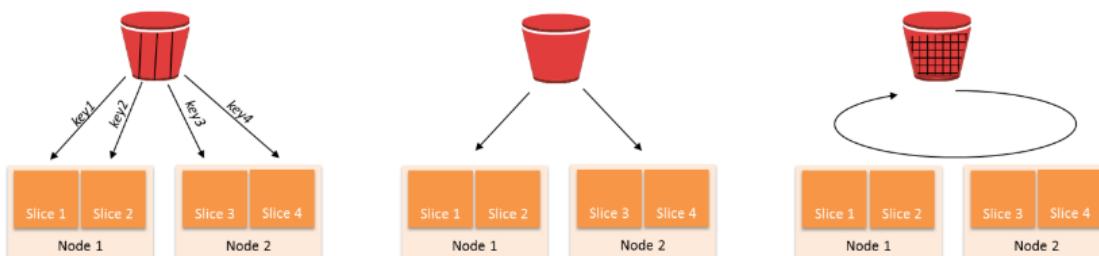
*key value to same location*

### All

*All data on every node*

### Even

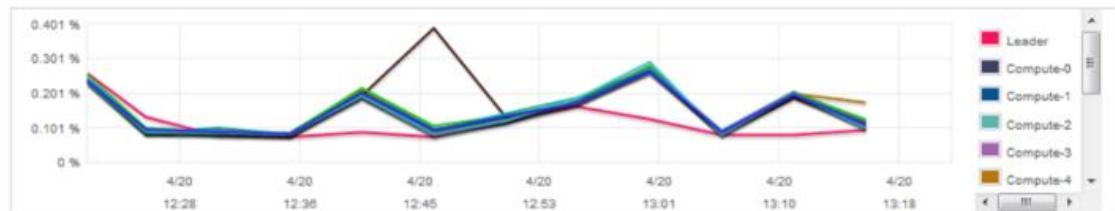
*Round robin distribution*



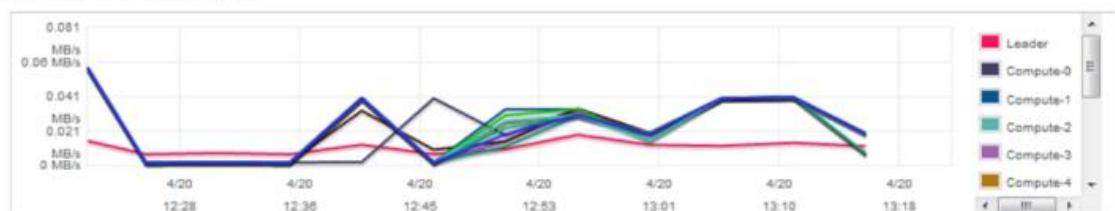
Permite las características de Datawarehousing potentes y clásicas para el análisis de las consultas, optimización y mejoras. En las partes críticas de nuestra operativa habrá que intentar maximizar el uso de los nodos de cómputo, de forma que la carga se reparta lo mayor posible.



CPU utilization



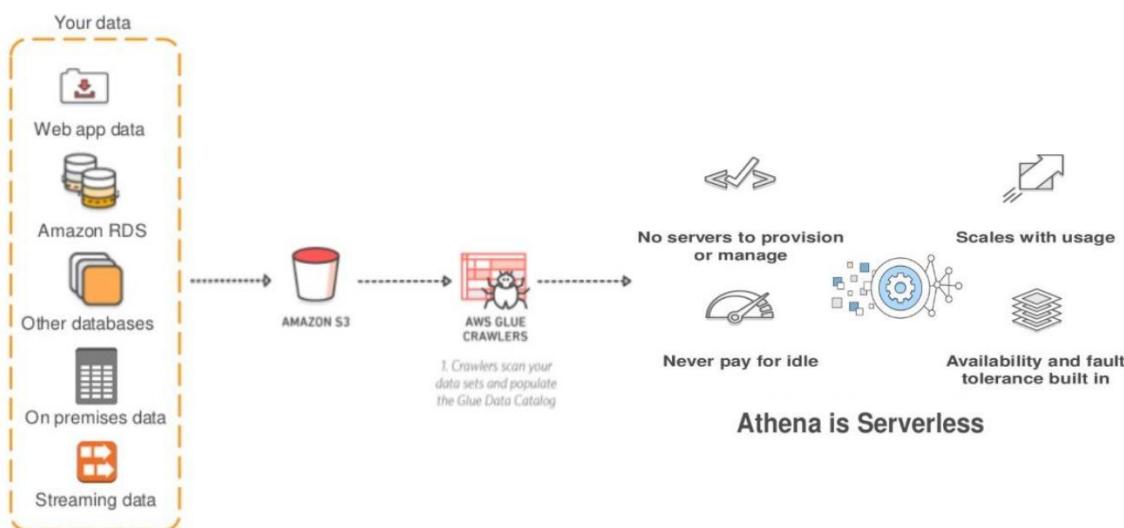
Network receive Throughput



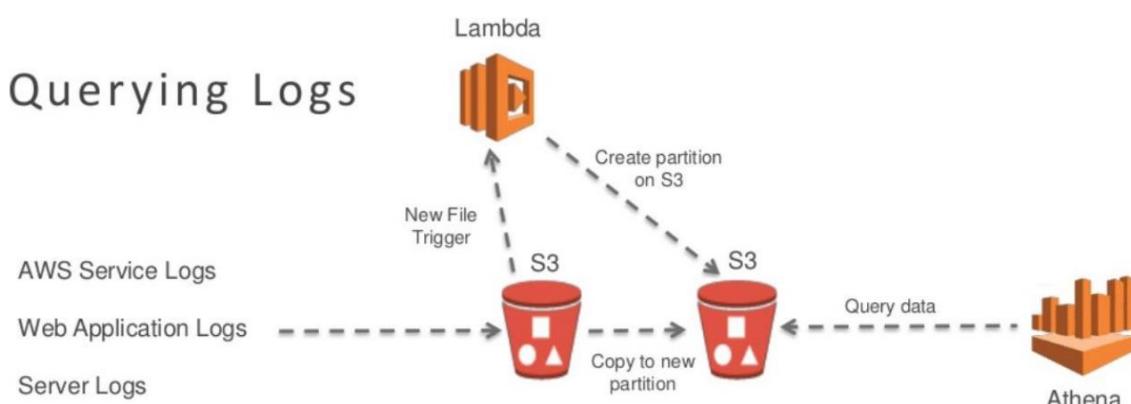
### 3.4. Athena AWS

Amazon Athena es una base de datos serverless que trabaja directamente sobre S3 en diversos formatos como CSV, ORC, Parquet, y otros comprimidos, de forma directa sin importaciones. Se factura simplemente 5\$ por Terabyte procesado y los resultados se almacenan igualmente en S3.

Amazon Athena es un servicio sin servidor, por lo que no hay que administrar infraestructura. No es necesario preocuparse por configuraciones, actualizaciones del software, errores ni del escalado de la infraestructura cuando crezcan sus conjuntos de datos y cantidad de usuarios. Athena se ocupa de todo esto automáticamente, para que pueda concentrarse en los datos y no en la infraestructura.



Es habitual su uso dentro del ecosistema de AWS para procesamientos periódicos de grandes cantidades de información sistemática como logs, series temporales, eventos, etc, previo procesado serverless mediante servicios ETL como Kinesis, Glue, o procesamientos personalizados en Lambda.



La integración mediante API es sencilla y compatible con ODBC-JDBC y tiene librerías en todos los entornos. Su origen es el motor Presto, soportando SQL así como consultas HiveQL. El origen del servicio es Facebook como desarrolladores y Netflix como pioneros de uso en cloud:

<https://netflixtechblog.com/using-presto-in-our-big-data-platform-on-aws-938035909fd4>



### Connecting to Amazon Athena - API

Asynchronous Query API

```
StartQueryExecution
client.startQueryExecution({
  QueryString: 'SELECT * FROM table_name LIMIT 100',
  ResultConfiguration: { OutputLocation: 's3://bucket/output/' },
  EncryptionConfiguration: { EncryptionOption: 'SSE_S3' },
  QueryExecutionContext: { Database: 'default_db' }
}, (err, result) => {})

GetQueryResults
client.getQueryResults({
  QueryExecutionId: '2ef5d590-025a-48ec-895e-6bedfe72bc95',
  MaxResults: 1000,
  NextToken: null
}, (err, data) => {})
```

## 3.5. AWS DynamoDB

Amazon DynamoDB es un servicio de base de datos NoSQL totalmente administrado que ofrece un rendimiento rápido y predecible, sin limitaciones de almacenamiento y que permite gestionar los permisos de almacenamiento, así como una perfecta escalabilidad.

DynamoDB permite delegar las cargas administrativas que supone tener que utilizar y escalar bases de datos distribuidas, para que no tener que preocuparse del aprovisionamiento, la instalación ni la configuración del hardware, ni tampoco de las tareas de replicación, aplicación de parches de software o escalado de clústeres. DynamoDB también ofrece el cifrado en reposo, que elimina la carga y la complejidad operativa que conlleva la protección de información confidencial.

Es un almacén de claves/valor, flexible y sin estructura fija, diseñado para garantizar un determinado rendimiento, así como una determinada disponibilidad para cada tabla (en NoSQL suele haber pocas tablas), es decir, se definen parámetros por tabla y se paga según lo exigido en cada una.



Amazon DynamoDB

#### Provisioned Throughput Capacity:

##### Provisioned Throughput Calculator

The amount of read and write capacity you need to provision depends on the factors listed below. This tool will help you determine your read and write capacity requirements.

Estimate Average Item Size (KB):	<input checked="" type="radio"/> less than 1 KB <input type="radio"/> 1 KB or more <input style="width: 100px;" type="text" value="item size (integer)..."/>
Estimate items read/sec:	<input type="text" value="1000"/>
Read consistency:	<input type="radio"/> Strongly Consistent <input checked="" type="radio"/> Eventually Consistent
Estimate items written/sec:	<input type="text" value="1"/>
<input type="button" value="Calculate"/>	

**⚠️** Throughput capacity for this table will cost up to \$66.17 per month if you have exceeded the free tier.

Por ejemplo, en esta tabla concreta especificamos un rendimiento garantizado de 1000 lecturas/s y 1 escritura/s, con una coherencia eventual (es decir, que permite desorden de peticiones), y eso nos costará \$67,17 al mes.

En una base de datos NOSQL clave-valor, respecto a un modelado relacional, se juntan muchas tablas y relaciones en una o pocas tablas con un sistema de claves de partición y ordenación, basándonos en el tipo de consultas específico que se van a realizar (y si éstas cambian podría alterar el modelo) y eliminando la necesidad de un ORM:

Primary Key		Attributes											
PK	SK (GSI-1-PK)	GSI-1-SK	StartDate	EndDate	JobID	JobTitle	PhoneNumber	Email	ManagerID	Country	City	Region	Department
HR-EMPLOYEE1	EMPLOYEE1	Data (Full Name) John Smith											
	QUOTA-2017-Q1	Data (Order Totals USD) 50000	EmployeeName										
	HR-CONFIDENTIAL	Data (Hire Date) 2015-11-08	EmployeeName	Salary	CommissionPct								
	WA SEATTLE	Data (Desk Location) B01 F07 A27 R05	EmployeeName										
	J-AM3	Data (Job Title) Principal Account Manager	DepartmentID	StartDate	EndDate	JobID							
	JH-AM2	Data (Job Title) Senior Account Manager	DepartmentID	StartDate	EndDate	JobID							
	JH-AM1	Data (Job Title) Account Manager	DepartmentID	StartDate	EndDate	JobID							
	HR-REGION1	PNW	RegionName										
	HR-COUNTRY1	USA	CountryName	RegionID									
	HR-LOCATION1	WA SEATTLE	CityName	PostalCode	StreetAddress	StateProvince	CountryID						
HR-JOB1	J-AM3	Data (Job Title) Principal Account Manager	JobTitle	MinSalary	MaxSalary								
	HR-DEPARTMENT1	COMMERCIAL	DepartmentName	ManagerID	City	Location							
	OE-CUSTOMER1	CUSTOMER1	Address	IncomeLevel	PhoneNumber	NLSLanguage	NLSTerritory	CreditLimit	CustEmail	CustLocatid	DateOfBirth	MaritalStatus	
	OE-ORDER1	CUSTOMER1	Data (StatusDate) OPEN2018_08_11	GSI-Bucket (GSI-2-PK) RND(0,N)	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionID				
	EMPLOYEE1	Data (StatusDate) OPEN2018_08_11	Order Total	EMPLOYEE1									
	PRODUCT1	PRODUCT1	Data (StatusDate) OPEN2018_08_11	GSI-Bucket (GSI-2-PK) RND(0,N)	OrderQuantity	UnitPrice							
	OE-PRODUCT1	PRODUCT1	Data (Product Name) Quickcrete Cement - 50 lb bag	ProductDescription	WAREHOUSE1	WAREHOUSE2	CategoryID	WeightClass	WarrantyPeri	SupplierID	ProductSta	ListPrice	
	PNW	Data (Region Name) Pacific Northwest	TranslatedName	Description	InventoryQty	InventoryQty							
	PNW	Data (Warehouse Type) Building supplies	WarehouseSpec	Location	WHGeoLocation								

### 3.5.1. Componentes básicos

- Tablas:** al igual que otros sistemas de base de datos, DynamoDB almacena datos en tablas. **Una tabla es una colección de datos.**
- Elementos:** Cada tabla contiene cero o más elementos. **Un elemento es un grupo de atributos que puede identificarse de forma exclusiva entre todos los demás elementos.** Los elementos de DynamoDB son similares en muchos aspectos a las filas o los registros. En DynamoDB, no existe ningún límite respecto al número de elementos que pueden almacenarse en una tabla.
- Atributos:** cada elemento se compone de uno o varios atributos. Un atributo es un componente fundamental de los datos, que no es preciso dividir más. Por ejemplo, un elemento de una tabla People contiene los atributos PersonID, LastName, FirstName, etc. En DynamoDB, los atributos se parecen en muchos aspectos a los campos o columnas en otros sistemas de bases de datos.

### 3.5.2. Características de DynamoDB

Servicios de BBDD NoSQL. No hay limitaciones de almacenamiento y permite gestionar los permisos mediante accesos con IAM. Es una colección de tablas.

- Las tablas son la estructura de más alto nivel en estas bases de datos de dynamoDB.

Los requerimientos de rendimiento, es decir, el número de lecturas o escrituras por segundo se especifica a nivel de tablas

- WCU (Write Capacity Units) = Nº de bloques 1 KB que puede escribir por segundo
- RCU (Read Capacity Units) = Nº de bloques 1 KB que puede leer por segundo

Por tanto, el coste de nuestra base de datos es DynamoDB vendrá determinada por estos dos valores, según el valor que pongamos en el WCU y en el RCU de cada tabla tendremos el coste.

Es un sistema distribuido, es decir, que cada vez que grabamos un dato se distribuye por otras zonas geográficas. Cada región es independiente y se encuentra aislada de las demás regiones

de AWS. Por ejemplo, si tenemos una tabla denominada People en la región us-east-2 y otra tabla denominada People en la región us-west-2, se consideran dos tablas completamente independientes.

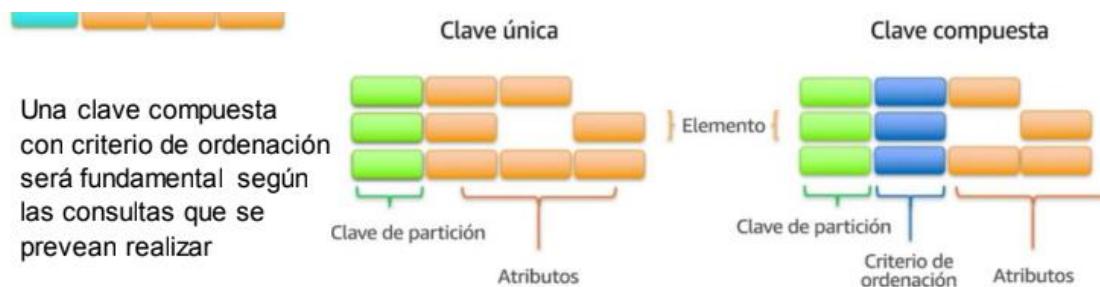
- Cada región de AWS consta de varias ubicaciones distintas denominadas **Zonas de disponibilidad**. Cada zona de disponibilidad está aislada de los errores que se produzcan en otras zonas de disponibilidad y proporciona conectividad de red de baja latencia económica con otras zonas de disponibilidad de la misma región. Esto permite la replicación rápida de los datos entre varias zonas de disponibilidad de una región.
- Cuando la aplicación escribe datos en una tabla de DynamoDB y recibe una respuesta **HTTP 200 (OK)**, la escritura se ha realizado y es duradera. Los datos presentan consistencia final en todas las ubicaciones de almacenamiento, normalmente en el plazo de un segundo o menos.

Como es una base de datos de tipo noSQL, no tiene esquemas definidos para cada una de las tablas, es decir, las tablas son flexibles, donde podemos añadir más columnas en cualquier momento.

- Las filas de cada tabla se denominan ítems, mientras que las columnas de la tabla se denominan atributos. Cada fila o cada ítem tiene un número variable de atributos o columnas rellenos.
  - Es decir, que podemos crear una fila con cuatro columnas llenas y otra fila solamente con dos.
  - Podemos llenar un número arbitrario de columnas o atributos.

DynamoDB admite dos tipos distintos de clave principal:

- **Clave de partición (Partition Key o Hash Key)**: contiene un valor único para cada ítem o cada fila de la tabla.
  - DynamoDB utiliza el valor de clave de partición como información de entrada a una función hash interna. El resultado de la función hash determina la partición (almacenamiento físico interno de DynamoDB) donde se almacenará el elemento.
  - En una tabla que solo tiene una clave de partición, no puede haber dos elementos que tengan el mismo valor de clave de partición.
- **Clave de ordenación (Sort Key)**: contiene un segundo valor que puede servir para ordenar los datos. Por ejemplo, un campo de fecha de tiempo, logotipos. Los tipos de atributos que pueden tener las tablas pueden ser de tipo string numérico binario booleano que tipo null, sin datos de tipo documento o de tipo set.



Integración directa con la mayoría de servicios de AWS: S3, EMR, Data pipeline, lambda, kinesis, etc.

### 3.5.3. Particiones

Las particiones en DynamoDB son los nodos de almacenamiento de procesamiento de DynamoDB. Inicialmente una tabla equivale a una partición. Todos los datos de la tabla se almacenan en una partición.

Una partición puede almacenar hasta un máximo de 10 GB de datos, además, también puede manejar hasta un máximo de 3000 RCU y 1000 WCU. El número de particiones afecta al rendimiento del sistema, es decir, hay que diseñar las tablas y aplicaciones para evitar problemas de entrada, salida.

Si superas alguno de los límites de 10 gigas es 1000 RCU o 1000 WCU, tus datos serán divididos en varias particiones.

Los datos se dividen basándose en atributo Partition Key para ir distribuyendo los valores por las distintas partes.

La gestión de las particiones, es decir, su número es gestionado por DynamoDB según requerimiento de rendimiento capacidad. Si aumentamos su rendimiento, es decir, si aumentamos los requisitos de rendimiento aumentando o solicitando mayores valores para RCU y WCU, lo que hará entonces será aumentar el número de particiones.

En el momento que queramos bajar los requerimientos, bajando así los rangos RCU y WCU, DynamoDB debe ya no reducirá este número de particiones, con lo cual tendremos un exceso de almacenamiento sin usar en cada partición. Por tanto, hay que ser cuidadoso a la hora de incrementar los requerimientos de rendimiento RCU y WCU.

### 3.5.4. Índices secundarios globales y locales

DynamoDB B ofrece dos operaciones de recuperación de datos como son el SCAN y QUERY:

- **SCAN** se utiliza para revisar la estructura de una tabla, los atributos, etc.
- **QUERY** se utiliza para obtener los datos de una tabla mediante partition key sort key e indicando una partición.
  - Puedes obtener una única fila de la tabla pasándole un valor concreto en la Partition Key o puedes obtener múltiples filas de la tabla pasándole una Partition Key y una Sort Key.

Además, existen dos tipos de índices, como hemos indicado antes, el global secondary index, es decir, el índice secundario global y el índice secundario local.

Los índices permiten mejorar el rendimiento de las consultas a las tablas.

Este índice sólo se puede crear en el momento en el que se crea la tabla. Contiene la clave de partición, la clave de ordenación, la nueva clave de ordenación, y también podemos incluir otros valores opcionales, es decir, atributos de la tabla que queramos introducir en este caso, que serían atributos proyectados.

Con ello, cualquier dato grabado una tabla es copiada de forma asíncrona a todos los índices locales secundarios de esa tabla.

Comparte el RCU i WCU con la tabla es un índice independiente, y este índice local solamente contendrá aquellas filas que tengan rellena la información de la nueva clave de ordenación o de aquellos atributos opcionales incluidos en la clave, es decir, los atributos proyectados, con lo cual tiene un menor número de filas de que solamente tendrá aquellas filas que tienen

información en la nueva clave o en los atributos que serán añadido. **Por tanto, las consultas serán más eficientes.**

Los índices secundarios globales que pueden crearse en cualquier momento. Pueden ser una alternativa a la clave de partición y a la de ordenación. Podemos configurar a los atributos de proyección serían:

- CASE ONLY, donde sólo se incluyen clave de partición y clave ordenación.
- INCLUDE donde especifica los atributos proyectados, seleccionados
- ALL donde todos los atributos son seleccionados para esta clave global.

Índice secundario global puede definir su propio RCU y WCU, al igual que una tabla y también los cambios descritos en la tabla son escritos en este índice global secundario de forma asíncrona.

### 3.5.5. DynamoDB Streams

DynamoDB Streams es una característica opcional que captura los eventos de modificación de datos que se producen en las tablas de DynamoDB. Los datos de estos eventos aparecen en la secuencia prácticamente en tiempo real y en el orden en que se han producido.

Cada evento se representa mediante un registro de secuencia. Si habilita una secuencia en una tabla, DynamoDB Streams escribe un registro de secuencia cada vez que se produzcan los siguientes eventos:

- Se agrega un nuevo elemento a la tabla: la secuencia captura una imagen del elemento completo, incluidos todos sus atributos.
- Se actualiza un elemento: la secuencia captura las imágenes de "antes" y "después" de los atributos del elemento que se han modificado.
- Se elimina un elemento de la tabla: la secuencia captura una imagen del elemento completo antes de eliminarlo.

Cada registro de secuencia también contiene el nombre de la tabla, la marca temporal del evento y otros metadatos. Los registros de secuencia tienen una vida útil de 24 horas; después, se eliminan automáticamente de la secuencia.

Se pueden configurar distintas formas:

- KEY\_ONLY: sólo se almacena en el string los atributos que son clave
- NEW\_IMAGE: sólo almacena los strings la fila entera que se ha actualizado
- OLD\_IMAGE: se almacena la fila entera previa al cambio.
- NEW\_AND\_OLD\_IMAGES: se almacena la fila previa al cambio y también como ha quedado después de realizar el cambio.

Puede utilizar DynamoDB Streams conjuntamente con AWS Lambda para crear un desencadenador; es decir, un código que se ejecute automáticamente cada vez que aparezca un evento de interés en una transmisión.

### 3.5.6. Rendimiento

Las particiones van a definir el rendimiento que tendrá nuestra BBDD en DynamoDB. Hay dos fórmulas en las que nos basaremos para calcular el número de particiones.

- La primera hace referencia al rendimiento:

$$\text{Nº Particiones} = (\text{RCU deseado} / 30000 \text{ RCU}) + (\text{WCU deseado} / 1000 \text{ WCU})$$

- Este valor será el número de particiones según el rendimiento que queremos que tenga nuestra base de datos.
- La segunda fórmula hace referencia a la capacidad:

$$\text{Nº particiones} = (\text{Tamaño de los datos en B} / 10 \text{ GB})$$

- Entonces este valor nos daría el número de particiones según la capacidad que deseamos en nuestra base de datos.

Por tanto, el número de particiones totales será el valor máximo de estas dos fórmulas. Las lecturas y las escrituras en esa base de datos son distribuidas entre las distintas particiones, es decir, se distribuirá de forma equitativa si tenemos bien definida nuestra clave de partición, es decir, tenemos que definir bien la clave de partición de nuestra base de datos de una vez para que los datos vayan de forma equitativa en las distintas particiones. Y así, en esas lecturas escrituras serán también equitativas. Es decir, iremos las mismas veces a cada una de las particiones.

¿Cómo seleccionamos esta clave de partición? Eligiendo un atributo que debe tener muchos valores diferentes. Debe ser también un atributo que tenga un patrón uniforme de lecturas escrituras en todas las particiones. Y si no encontramos ese atributo único, podemos crear esta tabla mediante la unión de varios atributos.

## 4. Laboratorio DynamoDB

Una vez creado el laboratorio por parte del profesor (en esta primera prueba he añadido un Ubuntu 20 como instancia de Docker en la creación de los módulos del lab) procederemos a arrancarlo pulsando en el botón que se habrá vuelto verde de AWS:

AWS  Start Lab End Lab AWS Details Readme Reset  
Used \$0 of \$100 05:52

e ir a la consola de administración de AWS:

The screenshot shows the AWS Management Console with the following elements:

- Header:** AWS logo, Services button, search bar ("Buscar servicios, características, blogs, documentos y mucho más [Alt+S]"), and navigation icons.
- Main Title:** "Consola de administración de AWS"
- Servicios de AWS Section:** A sidebar with two collapsed sections:
  - Servicios visitados recientemente:** Shows "DynamoDB" with a database icon.
  - Todos los servicios:** Shows "Informática", "Satélite", and "Seguridad, identidad".

Este laboratorio es una instancia vacía, es decir, no hay ningún servicio instalado por lo que debemos empezar instalando el servicio de DynamoDB.

## Servicios de AWS

**Buscar servicios**  
Puede escribir nombres, palabras clave o acrónimos.

X

**DynamoDB**  
Base de datos NoSQL administrada

▼ Servicios visitados recientemente

 **DynamoDB**

▶ Todos los servicios

Mostraremos todos los servicios disponibles de AWS y seleccionaremos el de DynamoDB. Una vez se nos abra la pantalla, clicaremos en CREAR TABLA:

## Amazon DynamoDB

Un servicio de base de datos NoSQL rápido y flexible a cualquier escala

DynamoDB es una base de datos de clave-valor y documentos completamente administrada que ofrece un desempeño de milisegundos de un solo dígito a cualquier escala.

### Introducción

Cree una tabla nueva para comenzar a explorar DynamoDB.

[Crear tabla](#)

No es necesario crear una BBDD, ya que esto lo gestiona internamente DynamoDB, por lo que vamos directamente a la creación de la tabla, donde indicaremos el nombre de la tabla y la clave principal o de partición.

Para este ejemplo, vamos a crear una BBDD llamada **usuarios** y como clave principal o de partición **id\_usuarios**, la cual debe tener definida el tipo de datos que es de tipo numérico.

#### Detalles de la tabla Info

DynamoDB es una base de datos sin esquemas que solo requiere un nombre de tabla y una clave principal al crear la tabla.

##### Nombre de la tabla

Se utilizará para identificar su tabla.

usuarios

Entre 3 y 255 caracteres. Solo se pueden usar letras, números, guiones bajos (\_), guiones (-) y puntos (.)

##### Clave de partición

La clave de partición forma parte de la clave principal de la tabla. Se trata de un valor hash que se utiliza para recuperar elementos de la tabla, así como para asignar datos entre hosts por cuestiones de escalabilidad y disponibilidad.

id\_usuarios

Número ▾

De 1 a 255 caracteres, distingue entre mayúsculas y minúsculas.

Además, vamos a seleccionar la opción de configuración personalizada:

#### Personalizar configuración

Utilice estas características avanzadas para que DynamoDB funcione mejor de acuerdo a sus necesidades.

En la capacidad de lectura/escritura, vamos a activar auto scaling en las dos opciones (lectura/escritura). Lo que hace es que cuando nuestra capacidad de lectura supera al 70%, va añadiendo más capacidad, más unidades de 1 en 1 hasta un máximo de 10. Tanto en lectura como escritura es muy recomendable hacer auto scaling.

### Configuración de capacidad de lectura/escritura [Info](#)

#### Modo de capacidad

Bajo demanda

Simplifique la facturación pagando por las lecturas y escrituras reales que realiza su aplicación.

Aprovisionado

Administre y optimice los costos asignando la capacidad de lectura/escritura por adelantado.

#### Capacidad de lectura

Auto Scaling [Info](#)

Ajusta dinámicamente la capacidad de desempeño provisionada en su nombre en respuesta a los patrones de tráfico reales.

Activado

Desactivado

Unidades de capacidad mínimas

1

Unidades de capacidad máxima

10

Objetivo de utilización (%)

70

El resto de opciones las dejaremos como predeterminadas y clicamos en crear. Con esto hemos creado nuestra primera tabla en DynamoDB.

**ⓘ La nueva consola de DynamoDB ya está completa y se convierte en su experiencia predeterminada**

Tras la fase de vista previa en la que analizamos e incorporamos sus comentarios, hemos completado la vuelta a la consola anterior desde el panel de navegación.

**✓ La tabla usuarios se ha creado correctamente.**

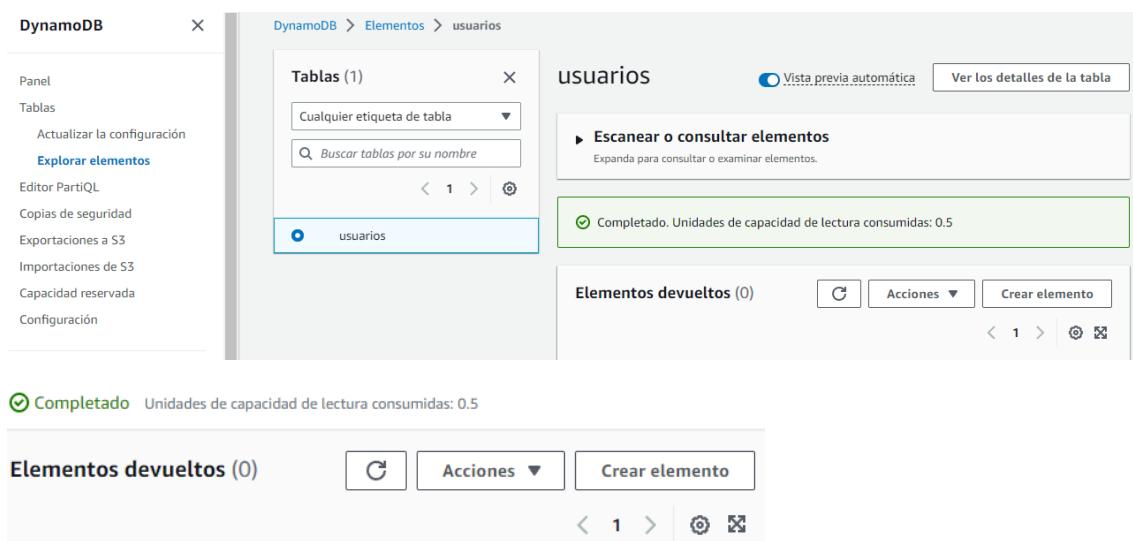
Cuando acabe de crearse aparecerá lo siguiente:

**✓ La tabla usuarios se ha creado correctamente.**

Tablas (1) <a href="#">Info</a>						
	<input type="checkbox"/>	Nombre ▲	Estado	Clave de partición	Clave de ordenación	Índices
	<input type="checkbox"/>	usuarios	<input checked="" type="radio"/> Active	id_usuarios (Number)	-	0 Aprovisionado con Auto Scaling (1)

## 4.1. Añadir elementos

Para añadir elementos nuevos a nuestra tabla, seleccionaremos el menú lateral izquierdo y clicaremos en **Tablas → Explorar Elementos**, luego iremos a la parte inferior hasta ver el botón crear elementos, lo que permite añadir filas a nuestra tabla:



Completo. Unidades de capacidad de lectura consumidas: 0.5

**Elementos devueltos (0)**

**Acciones ▾** **Crear elemento**

Para este ejemplo crearemos una fila con dos cadenas de texto y una cadena de cadenas de textos, además el valor de la clave de partición será 1:

Atributos		Agregar nuevo
Nombre de atributo	Valor	Tipo
id_usuario - Clave de partición	1	Número
Nombre	Daniel	Cadena
Apellidos	López	Cadena
Juegos	Insertar un campo	Conjunto de cadenas
0	The witcher 3	Cadena
1	Horizon Zero Dawn	Cadena
2	Spider-man	Cadena

Esta acción podemos desarrollarla mediante el uso de formato Formulario o directamente sobre un JSON:

Crear elemento		
Formulario		JSON
Atributos		
Nombre de atributo	Valor	Tipo
id_usuario - Clave de partición	0	Número
<b>Agregar nuevo atributo ▾</b>		

**Atributos**  Ver JSON de DynamoDB

```
1 ▼ {
2 ▼   "id_usuario": {
3     "N": "1"
4   },
5 ▼   "Apellidos": {
6     "S": "López"
7   },
8 ▼   "Juegos": {
9 ▼     "SS": [
10       "Horizon Zero Dawn",
11       "Spider-man",
12       "The witcher 3"
13     ]
14   },
15 ▼   "Nombre": {
16     "S": "Daniel"
17   }
18 }
```

Al darle a Crear elemento, nos agregará la fila que hemos insertado

<input type="checkbox"/>	id_usuar...	Apellido1	Nombre
<input type="checkbox"/>	1	López	Daniel

**Actividad 1:** añade más usuarios a la BBDD. Además, añade el campo edad y el apellido 1. Deja apellido1 sin rellenar ¿ha dado algún error?

**Actividad 2:** crea una nueva tabla con las siguientes características:

- Nombre: Almacen
- Clave partición: id\_almacen
- Clave de ordenación:id\_artículo (qué usuario va a recibir el artículo)

En este caso, la clave principal estará formada por la clave de partición + la clave de ordenación, ambos son numéricos.

- Configuración personalizada
- Aprovisionado: por defecto:
- Auto scaling: igual que la tabla anterior
- Resto: mismas condiciones.

Al darle a crear la tabla os aparecerá un fallo, solucionadlo y continuad. Una vez creada, añade los siguientes elementos:

	<b>id_usuarios</b>	<b>id_articulo</b>	<b>contenido</b> ⓘ
	1	1	Este es mi primer articulo
	1	2	Este es nuestro segundo articulo

Comprueba que la clave de ordenación funciona.

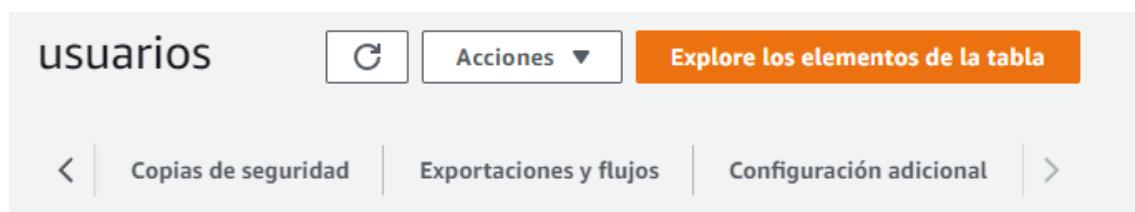
## 4.2. Capacidades de las tablas

Vamos al apartado de tablas, donde podemos ver las dos tablas que hemos creado

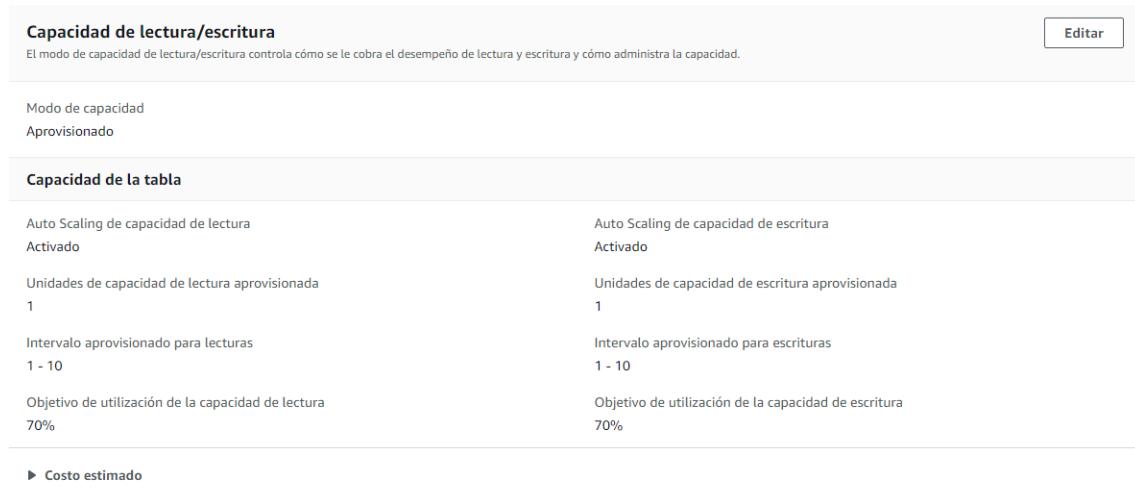
<input type="checkbox"/>	Nombre	▲	Estado
<input type="checkbox"/>	Articulos		Active
<input type="checkbox"/>	usuarios		Active

anteriormente:

Seleccionaremos la tabla de artículos y buscamos la pestaña de Configuración adicional, justo al final. En esta pestaña se encuentra el apartado de Capacidad, donde podemos modificar la capacidad de lectura/escritura.



The screenshot shows the 'Articulos' table configuration page. At the top, there's a navigation bar with 'usuarios' on the left, a central search icon, 'Acciones' with a dropdown arrow, and a prominent orange button 'Explore los elementos de la tabla'. Below this is a horizontal navigation bar with three tabs: 'Copias de seguridad', 'Exportaciones y flujos', and 'Configuración adicional'. The 'Configuración adicional' tab is currently selected.



The screenshot shows the 'Capacidad de lectura/escritura' configuration section for the 'Articulos' table. It includes a sub-section for 'Modo de capacidad' set to 'Aprovisionado'. The main table has two columns: 'Capacidad de la tabla' and 'Auto Scaling de capacidad de lectura' / 'Auto Scaling de capacidad de escritura'. Both are set to 'Activado'. Other parameters listed include 'Unidades de capacidad de lectura aprovisionada' (1), 'Intervalo aprovisionado para lecturas' (1 - 10), 'Objetivo de utilización de la capacidad de lectura' (70%), 'Unidades de capacidad de escritura aprovisionada' (1), 'Intervalo aprovisionado para escrituras' (1 - 10), and 'Objetivo de utilización de la capacidad de escritura' (70%). A 'Costo estimado' link is at the bottom.

Hay dos formas de definir nuestra capacidad de lectura y escritura:

Por un lado, podemos elegir la de Aprovisiona, es decir, apta para la gratuita. Aprovisionar significa que nosotros decidimos cuántas unidades de lectura o escritura simultánea se pueden hacer en nuestra tabla. En este caso hay puesto 1, 1 lectura simultánea y 1 escritura simultánea. El coste tendría 0,59\$/mes.

<b>Costo estimado</b>			
En función de su configuración, a continuación se muestra el costo total estimado de la capacidad de lectura y escritura aprovisionada para su tabla e índices. Para obtener más información, consulte <a href="#">Precios de Amazon DynamoDB</a> para la capacidad aprovisionada.			
Unidades de capacidad de lectura total 1	Unidades de capacidad de escritura total 1	Región us-east-1	Costo estimado 0,59 US\$/mes

Podemos ajustar el aprovisionamiento en función de la carga de trabajo que vayamos a tener. Lo mismo podemos hacer con el auto scaling.

Otra forma de configurar la BBDD es bajo demanda.

**Bajo demanda**

Simplifique la facturación pagando por las lecturas y escrituras reales que realiza su aplicación.

En este caso, no hace falta que planifiquemos cuántas unidades o cuántas capacidades de lectura o escritura ni el auto scaling, sino simplemente dejamos que sea Amazon quien gestione la demanda en base al uso real que se le vaya a dar.

En caso de que vengan Vale de repente vienen 200 peticiones de lectura por va a atender a las doscientas en ese momento y nos cobrará por ello. Si luego simplemente hay dos lecturas, por lo dejan dos y nos cobrará por ello, pagaremos exactamente por lo que utilicemos sin tener que provisionar nosotros ni decir cuál es nuestra capacidad.

Como se puede ver, es muy cómodo a la hora de que no tengamos un error que Amazon se encarga de gestionar las capacidades para que siempre funcione y así atiende bien a los ciclos. Y cuando hay poca demanda también baja las capacidades y solo nos cobra por el uso real.

Así que, si queremos dejar de estar bajo demanda para que Amazon se encargue, lo marcamos y le damos a guardar. Y así nuestra tabla a partir de ahora, en cuanto a capacidad, no tendrá problema, será flexible en función de la carga que tenga.

**Capacidad de lectura/escritura**

El modo de capacidad de lectura/escritura controla cómo se le cobra el desempeño de lectura y escritura y cómo administra la capacidad.

Modo de capacidad

Bajo demanda

Esta es quizá la forma más eficiente de configurar nuestra tabla para no tener ningún tipo de problemas.

### 4.3. Operaciones sobre los datos

El siguiente apartado mostrará las acciones que podemos hacer sobre los datos de nuestras tablas. Nos dirigimos a la tabla artículos que tienen actualmente cuenta con dos elementos.

Las operaciones que podemos realizar son:

- **Añadir elementos:** Añadiremos un nuevo elemento para que el usuario 2 escriba su primer artículo. Usando la vista de JSON podemos escribir varias líneas:

	<b>id_usuar...</b>	<b>id_articu...</b>	<b>Contenido</b>
	2	1	Este es mi primer artículo

- **Editar elementos:** Al editar un elemento ya creado podemos modificar el contenido de los campos que tiene y además añadir más columnas conforme necesitemos.

	<b>id_usuar...</b>	<b>id_articu...</b>	<b>Contenido</b>	<b>Descripci...</b>	<b>Título</b>
	2	1	Este es mi p...		Artículo 1
	1	2	Este es mi s...		
	1	1	Este es mi p...		

- **Realizar consultas:** Es posible realizar búsquedas mediante la pestaña de elementos. en ella tenemos dos opciones para la realización de la búsqueda: examen o consulta. La consulta nos permite buscar por un identificador concreto, es decir, tenemos que poner la clave de partición de forma obligatoria y que identifique a un elemento. Si tenemos clave de ordenación si que es posible definirla mediante el selector

▼ Artículos

<b>Examen</b>	<b>Consulta</b>
Tabla o índice	
<input type="text" value="Artículos"/>	
id_usuarios (Clave de partición)	
<input type="text" value="Escribir valor de clave de partición"/>	
id_articulos (Clave de ordenación)	
<input type="button" value="Igual que"/>	<input type="text" value="Escribir valor de la clave de ordenación"/>
<input type="checkbox"/> Orden descendente	

Ejemplo:

## ▼ Articulos

Tabla o índice

Articulos

id\_usuarios (Clave de partición)

1

id\_articulos (Clave de ordenación)

Mayor que

1

Orden descendente

El modo examen sirve para buscar por el valor contenido dentro de la tabla.

## ▼ Articulos

Tabla o índice

Articulos

### ▼ Filtros

Nombre de atributo

Tipo

Condición

Valor

Primero debemos elegir la tabla donde vamos a realizar la búsqueda:

Tabla o índice

Articulos

Tabla

Articulos

Una vez seleccionada la tabla, crearemos un filtro para poder buscar el contenido de un campo. En el ejemplo vamos a buscar la palabra “mi”, indicando en el campo que se trata de un tipo “cadena”

### ▼ Filtros

Nombre de atributo

Tipo

Condición

Valor

- **Borrar elementos:** Es posible borrar todos los elementos de una tabla simplemente seleccionando los elementos y marcar la opción de borrar.



Acciones ▲

#### 4.4. DAX - Caché de tablas

El siguiente apartado de DynamoDB es DAX. DAX es un servicio de almacenamiento en caché compatible con DynamoDB que le permite beneficiarse de un rápido rendimiento en memoria para las aplicaciones más exigentes. DAX aborda tres escenarios principales:

1. Como caché en memoria, DAX reduce los tiempos de respuesta de las cargas de trabajo de lectura eventualmente consistente en un orden de magnitud, de cifras en milisegundos de un solo dígito a microsegundos.
2. DAX reduce la complejidad operativa y de las aplicaciones al ofrecer un servicio administrado que es compatible mediante API con DynamoDB. Por consiguiente, solo requiere unos mínimos cambios funcionales para poder usarlo con una aplicación existente.
3. Para las cargas de trabajo intensas o en ráfagas, DAX ofrece mayor rendimiento y posibles ahorros en el coste, porque disminuye la necesidad de aprovisionar unidades de capacidad de lectura de más. Esto resulta especialmente beneficioso para las aplicaciones que requieren lecturas reiteradas de claves individuales.

Por tanto, podemos describir DAX como una caché de memoria que reduce los tiempos de acceso a nuestras tablas. Por ejemplo, si vamos a tabla artículos, para acceder a ella desde el momento que clicamos tendremos acceso en milisegundos, es decir, cualquier consulta, el tiempo de respuesta será entre 1 y 9 milisegundos.

Es posible que en algunas aplicaciones interese reducir el tiempo de acceso, por ejemplo a microsegundos, es decir, 5 microsegundos en lugar de 5 milisegundos, para ello necesitaríamos crear una caché. Una caché sirve para almacenar valores de uso frecuente o con previsión de que se va a usar en el futuro, de forma que los usuarios accederán a la información sin necesidad de acceder a la propia tabla.

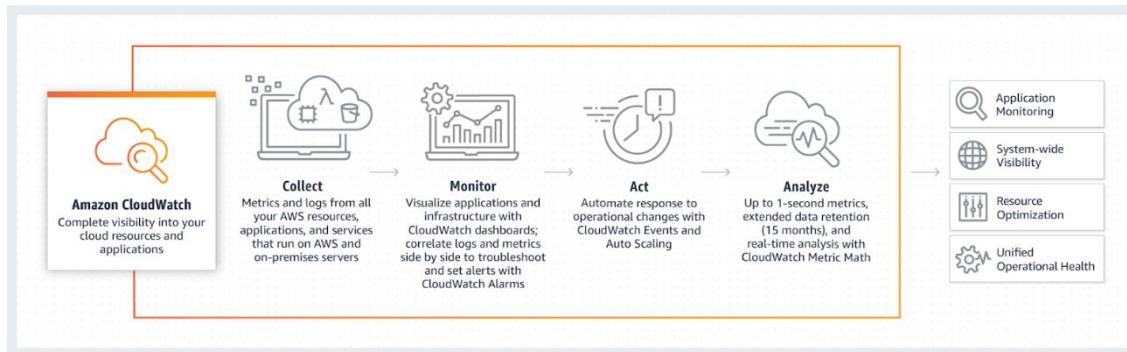
Este apartado se configuraría desde el clúster de DAX, aunque con la versión que tenemos de AWS actual no es posible crear Clústeres de DAX.

El siguiente [enlace](#) explica cómo configurarlo.

#### 4.5. Tablas con Lambda y Cloudwatch

**Lambda** es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. Lambda ejecuta el código en una infraestructura informática de alta disponibilidad y realiza todas las tareas de administración de los recursos informáticos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como la monitorización del código y las funciones de registro. Con Lambda, puede ejecutar código para prácticamente cualquier tipo de aplicación o servicio de backend. Lo único que tiene que hacer es suministrar el código en uno de los lenguajes que admite Lambda.

**Cloudwatch** es un servicio de monitoreo y observabilidad creado por desarrolladores, ingenieros de fiabilidad de sitios (SRE), administradores de TI, propietarios de productos e ingenieros de DevOps. **CloudWatch** proporciona datos e información procesable para monitorizar las aplicaciones, responder a cambios de rendimiento que afecten a todo el sistema y optimizar el uso de recursos.



Vamos a capturar la actividad de la tabla, artículos con flujos de dynamoDB. Cuando los configuremos seremos capaces de ver, por ejemplo, los logs de las operaciones que se hacen en nuestra tabla.

Como podemos ver en la siguiente imagen, el flujo de la BBDD está deshabilitado

Información general			
Clave de partición id_usuarios (Number)	Clave de ordenación id_articulos (Number)	Modo de capacidad Bajo demanda	Estado de la tabla <span style="color: green;">Active</span> <span style="color: green;">No hay alarmas activas</span>
▼ Información adicional			
Clase de tabla Estándar de DynamoDB			
Índices 0 globales, 0 locales	Flujo de DynamoDB <span style="color: red;">Disabled</span>	Recuperación a un momento dado	<span style="color: red;">Disabled</span>

Para poder capturar el flujo de datos de la tabla, iremos a la tabla que queremos capturar la actividad y pulsaremos sobre **exportaciones y flujo**:

Información general	Índices	Monitorear	Tablas globales	Copias de seguridad	Exportaciones y flujos
---------------------	---------	------------	-----------------	---------------------	------------------------

Para ir bajo a la opción de detalles del flujo

### Detalles del flujo de DynamoDB

Capture los cambios a nivel de elemento de la tabla y...

**Habilitar**

Estado del flujo

Desactivado

Cuando pulsemos sobre Habilitar se nos presentaran varias opciones:

## Detalles del flujo de DynamoDB

Capture los cambios a nivel de elemento de la tabla y envíe los cambios a un flujo de DynamoDB. Tras ello, podrá obtener acceso a la información de cambio a través de la API de los Flujos de DynamoDB.

### Ver tipo

Elija las versiones de los elementos modificados que desea enviar al flujo de DynamoDB.

**Solo atributos clave**

Solo los atributos clave del elemento modificado.

**Nueva imagen**

El elemento completo tal y como aparece después de cambiarlo.

**Imagen vieja**

El elemento completo tal y como aparece antes de cambiarlo.

**Imágenes nuevas y viejas**

Tanto la imagen nueva como la vieja del elemento cambiado.

- **Key attributes only (Solo atributos clave):** solo los atributos clave del elemento modificado.
- **New image (Nueva imagen):** el elemento completo tal y como aparece después de modificarlo.
- **Old image (Imagen anterior):** el elemento completo tal y como aparecía antes de modificarlo.
- **New and old images (Imágenes nuevas y anteriores):** ambas imágenes del elemento, la nueva y la anterior.

En nuestro caso vamos a coger la última opción. Ahora el flujo ya está habilitado:

▼ Información adicional

Clase de tabla

Estándar de DynamoDB

Índices

0 globales, 0 locales

Flujo de DynamoDB

 Enabled

Otra consecuencia de haber habilitado el flujo es que nos aparecerá en la misma sección del flujo los desencadenadores:

Detalles del flujo de DynamoDB

Capture los cambios a nivel de elemento de la tabla y envíe los cambios a un flujo de DynamoDB. Tras ello, podrá obtener acceso a la información de cambio a través de la API de los Flujos de DynamoDB.

[Desactivar](#)

Estado del flujo

 Habilitado

Ver tipo

Imágenes nuevas y viejas

ARN del flujo más reciente

 arn:aws:dynamodb:us-east-1:884642158813:table/Articulos/stream/2022-01-18T16:56:26.952

▼ Desencadenador (0)

Utilice desencadenadores para invocar una función de AWS Lambda cada vez que se cambie un elemento y, tras ello, se actualiza el flujo de DynamoDB.

[Desencadenador \(0\)](#)



[Configurar](#)

[Eliminar](#)

[Crear desencadenador](#)

¿Para qué tenemos que crear un desencadenador? Cada vez que se modifique o se crea un nuevo registro, se modifique o se borre un elemento de la tabla artículos. El desencadenador grabará los logs para que puedan ser consultados a posteriori.

Así, vamos a crear uno nuevo. Necesitamos una nueva función lambda, es posible reutilizar alguna de las que ya hayamos creado:

### Detalles de la función de AWS Lambda

Utilice desencadenadores para invocar una función de AWS Lambda cada vez que se cambie un elemento y, tras ello, se actualiza el flujo de DynamoDB.

#### Función de Lambda

Elija una función de AWS Lambda que deseé desencadenar cada vez que se cambie un elemento.

 [Elegir función](#)[Crear nuevo](#) 

Una función lambda es un fragmento de código serverless que ejecuta un procesamiento de datos

Introducimos el nombre de la función:

### Información básica

#### Nombre de la función

Escriba un nombre para describir el propósito de la función.

 **lambda\_articulos**

Utilice exclusivamente letras, números, guiones o guiones bajos. No incluya espacios.

Seguidamente, hemos de seleccionar el rol que va a ejecutar la función. Para nuestro caso, al ser un laboratorio de estudio, seleccionaremos el rol que viene por defecto llamado LabRole. No obstante, los siguientes párrafos explican cómo configurar un nuevo rol dentro de IAM.

#### Tiempo de ejecución INFORMACIÓN

Elija el lenguaje que desea utilizar para escribir la función. Tenga en cuenta que el editor de código de la consola solo admite Node.js, Python y Ruby.

 **Node.js 18.x** 

#### Arquitectura INFORMACIÓN

Elija la arquitectura del conjunto de instrucciones que desea para el código de la función.

- x86\_64**
- arm64

#### Permisos INFORMACIÓN

De forma predeterminada, Lambda creará un rol de ejecución con permisos para cargar registros en Amazon CloudWatch Logs. Puede personalizar este rol predeterminado más adelante al agregar los disparadores.

#### ▼ Cambiar el rol de ejecución predeterminado

##### Rol de ejecución

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#) .

- Creación de un nuevo rol con permisos básicos de Lambda
- Uso de un rol existente**
- Creación de un nuevo rol desde la política de AWS templates

##### Rol existente

Seleccione un rol existente que haya creado para usarlo con esta función de Lambda. El rol debe tener permiso para cargar registros en Amazon CloudWatch Logs.

 **LabRole** 

[View the LabRole role](#)  on the IAM console.

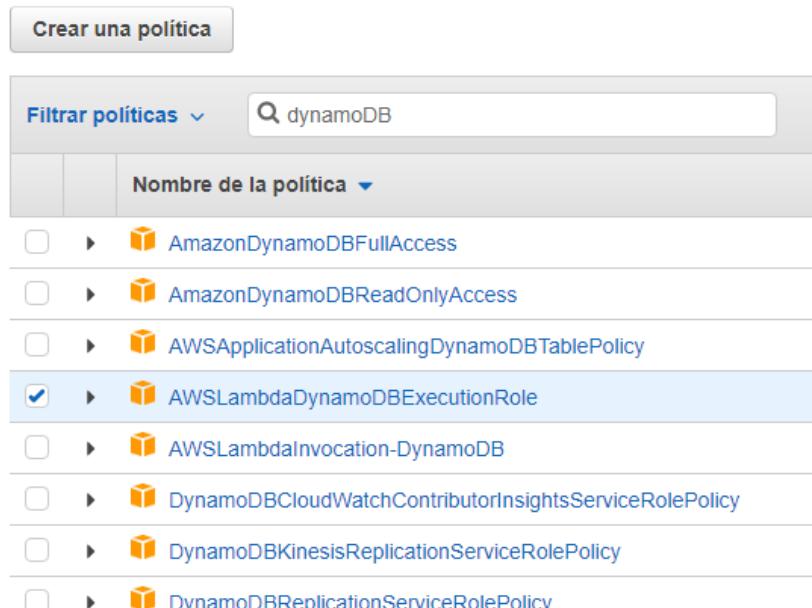
## Crear un rol

### Seleccionar el tipo de entidad



Permite a los servicios de AWS realizar

Y seleccionaremos los permisos que va a tener el rol, en este caso, ejecución de dynamoDB:



**Nombre de rol\***

Utilice caracteres alfanuméricos y "+=,.@-\_". 64 caracteres como máximo.

Una vez creada ya la función, vamos a crear el desencadenador:

## Crear un desencadenador

### Detalles de la función de AWS Lambda

Utilice desencadenadores para invocar una función de AWS Lambda cada vez que se cambie un elemento y, tras ello, se actualiza el flujo de DynamoDB.

#### Función de Lambda

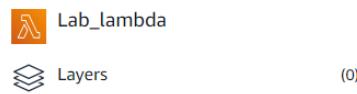
Elija una función de AWS Lambda que desee desencadenar cada vez que se cambie un elemento.

 X  
Usar función: lambda\_articulos  
MainMonitoringFunction  
LightsailMonitoringFunction  
**lambda\_articulos**[Crear nuevo](#) C **Habilitar desencadenador**

Especifique si el desencadenador debe estar habilitado al crearlo. Puede desactivarlo más adelante desde la consola de Lambda.

## Lab\_lambda

### ▼ Información general de la función [Info](#)

[+ Agregar desencadenador](#)

Clicamos en agregar desencadenador y buscamos DynamoDB.

### Configuración del desencadenador



#### Tabla de DynamoDB

Elija o escriba el ARN de una tabla de DynamoDB.

 X C

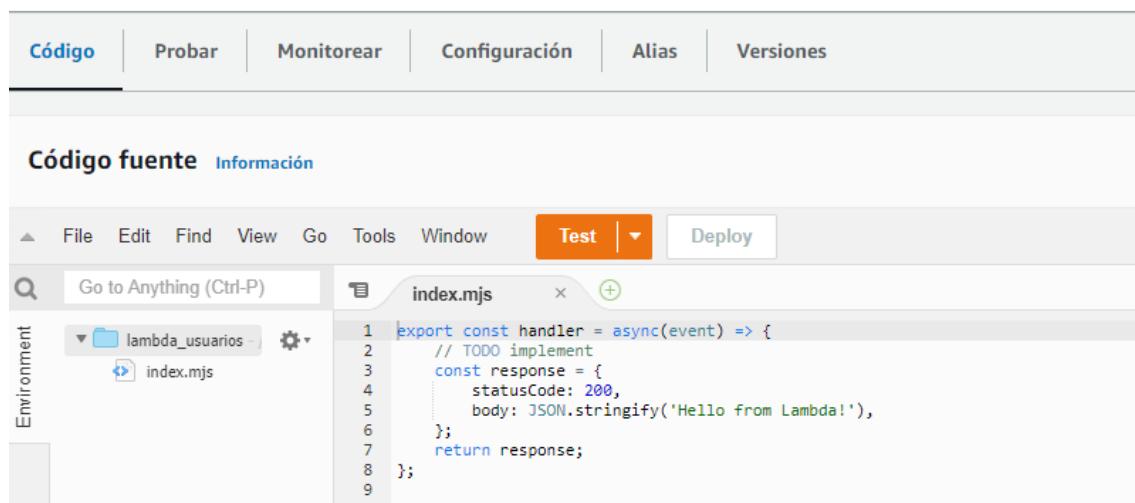
#### Tamaño del lote

Mayor número de registros que se van a leer a la vez en el flujo de actualizaciones de la tabla.

#### Intervalo del lote - Opcional

El tiempo máximo para recopilar registros antes de invocar la función, en segundos.

Una vez creado el desencadenador, nos dirigiremos al código que va a ejecutarse cuando se desencadene. Ahí veremos lo siguiente:



The screenshot shows the AWS Lambda function editor interface. At the top, there are tabs for 'Código', 'Probar', 'Monitorear', 'Configuración', 'Alias', and 'Versiones'. The 'Código' tab is selected. Below the tabs is a toolbar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', 'Window', 'Test' (which is highlighted in orange), and 'Deploy'. A search bar says 'Go to Anything (Ctrl-P)'. On the left, there's a sidebar labeled 'Environment' with a folder icon 'lambda\_usuarios' containing 'index.mjs'. The main area shows the code for 'index.mjs':

```
1 export const handler = async(event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Lambda!'),
6     };
7     return response;
8 };
9 
```

//----Función original -----

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

Simplemente lanza un mensaje a la consola para indicar que se ha ejecutado la función. Pero nosotros queremos capturar qué evento se ha producido y coger su ID y su nombre. Por ello, podemos o bien comentar la función anterior o bien, eliminarla para sustituirla por esta:

El código de la función:

```
//Función de monitorización de elementos
console.log('Loading function');
```

```
export const handler = async(event) => {
```

```
    for (const record of event.Records){
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB record: %j', record.dynamodb);
    }
    return `Successfully processed ${event.Records.length} record`;
};
```

Cuando estéis creando la función, os podéis preguntar, de dónde hemos sacado los eventID o eventName. Una forma fácil de saber qué nombre reciben los elementos a registrar es ir al ejemplo de prueba bajo el desencadenador:



En ese apartado debemos seleccionar la clase de prueba que vamos a hacer para nuestra función. Dynamodb-update

#### Evento de prueba

Invoque la función con un evento de prueba. Elija una plantilla que coincida con el servicio que activa la función o ingrese el documento de eventos en JSON.

- Nuevo evento  
 Evento guardado

Plantilla

dynamodb-update

En el siguiente apartado veremos los nombres que se producen cuando hay alguna modificación dentro de la BBDD:

```
1 ↴ []
2 ↴   "Records": [
3 ↴     {
4 ↴       "eventID": "c4ca4238a0b923820dcc509a6f75849b",
5 ↴       "eventName": "INSERT",
6 ↴       "eventVersion": "1.1",
7 ↴       "eventSource": "aws:dynamodb",
8 ↴       "awsRegion": "us-east-1",
9 ↴       "dynamodb": {
10 ↴         "Keys": {
11 ↴           "Id": {
12 ↴             "N": "101"
13 ↴           }
14 ↴         },
15 ↴         "NewImage": {
16 ↴           "Message": {
17 ↴             "S": "New item!"
18 ↴           },
19 ↴           "Id": {
20 ↴             "N": "101"
21 ↴           }
22 ↴         },
23 ↴         "ApproximateCreationDateTime": 1428537600,
24 ↴         "SequenceNumber": "4421584500000000017450439091",
25 ↴         "SizeBytes": 26,
26 ↴         "StreamViewType": "NEW_AND_OLD_IMAGES"
27 ↴       },
28 ↴       "eventSourceARN": "arn:aws:dynamodb:us-east-1:123456789012:table/ExampleTableWithStream/stream/2015-06-27T00:48:05.899"
29 ↴     },
...
```

Queda probar que la función funciona en el entorno de producción, por lo que añadiremos un nuevo elemento a la BBDD de artículos. Cuando se haya hecho, iniciaremos AWS Cloudwatch, donde debería aparecer una entrada nueva con la hora de la inserción y los datos de la acción que hemos hecho y que hemos configurado en la función:

▼	2022-01-21T16:38:14.492+01:00	2022-01-21T15:38:14.486Z	4aea34d1-7cb5-4608-acf0-1ff23feefe36	INFO	INSERT
	2022-01-21T15:38:14.486Z	4aea34d1-7cb5-4608-acf0-1ff23feefe36		INFO	INSERT
▶ 2022-01-21T16:38:14.493+01:00 2022-01-21T15:38:14.492Z 4aea34d1-7cb5-4608-acf0-1ff23feefe36 INFO DynamoDB record: {					

Si abrimos el apartado de dynamodb record veremos qué operación y qué datos han intervenido:

```
2022-01-21T15:38:14.486Z      4aea34d1-7cb5-4608-acf0-1ff23feefe36      INFO      INSERT

▼ 2022-01-21T16:38:14.493+01:00      2022-01-21T15:38:14.492Z 4aea34d1-7cb5-4608-acf0-1ff23feefe36      INFO      DynamoDB record:
2022-01-21T15:38:14.492Z      4aea34d1-7cb5-4608-acf0-1ff23feefe36      INFO      DynamoDB record:
{
  "Keys": {
    "Id": {
      "N": "101"
    }
  },
  "NewImage": {
    "Message": {
      "S": "New item!"
    },
    "Id": {
      "N": "101"
    }
  },
  "ApproximateCreationDateTime": 1428537600,
  "SequenceNumber": "442158450000000017450439091",
  "SizeBytes": 26,
  "StreamViewType": "NEW_AND_OLD_IMAGES"
}
```

## 5. Bibliografía

- <https://pandorafms.com/blog/es/bases-de-datos-nosql/>
- <https://www.mongodb.com/es/big-data-explained>
- Apuntes cloud IES Marenostrum
- Curso cefire: uso práctico del big data
- <https://blog.ida.cl/desarrollo/mongodb-atlas-el-salto-a-la-nube/>
- <https://www.youtube.com/watch?v=97FfXEy1zas>
- <https://www.youtube.com/watch?v=f0K9UI79JHQ>
- <https://www.jolugama.com/blog/2021/05/24/mongodb-tutorial-basico/>