

MÓDULO

HERRAMIENTAS BIG DATA

UNIDAD 6

TECNOLOGÍAS BIG DATA III: NOSQL

ÍNDICE

TU RETO EN ESTA UNIDAD.....	3
1. INTRODUCCIÓN	5
1.1. LAS BASES DE DATOS RELACIONALES	6
1.2. LIMITACIONES DEL MODELO RELACIONAL	8
2. NOSQL	9
2.1. ¿QUÉ ES NOSQL?	9
2.2. CARACTERÍSTICAS	10
2.2.1. DISTRIBUIDAS	10
2.2.2. FLEXIBILIDAD DE ESQUEMA	12
2.2.3. ADIOS AL MODELO RELACIONAL	13
2.2.4. SIN LENGUAJE COMUN.....	15
2.3. PROPIEDADES Y TEOREMA CAP	16
2.3.1. CONSISTENCIA.....	17
2.3.2. DISPONIBILIDAD	18
2.3.3. TOLERANCIA AL PARTICIONADO	18
2.3.4. TEOREMA CAP	19
3. FAMILIAS NOSQL	21
3.1. CLAVE-VALOR	21
3.2. DOCUMENTOS	23
3.3. FAMILIA DE COLUMNAS	24
3.4. GRAFOS	28
3.5. MULTIMODELO	29

¿QUÉ HAS APRENDIDO?	31
AUTOCOMPROBACIÓN	33
SOLUCIONARIO	37
BIBLIOGRAFÍA	39

TU RETO EN ESTA UNIDAD

Esta es la última unidad del bloque de introducción a las tecnologías Big Data. Es probable que alguna vez te ha tocado trabajar con alguna base de datos. Tal vez te suenen Oracle, MySQL, tal vez Access. Estas son bases de datos de un tipo que llamamos *relacional*. Ahora vamos a presentarte una serie especial de bases de datos surgidas de la necesidad de manejar la enorme cantidad de datos que tenemos en nuestro mundo *Big Data*: las bases de datos NoSQL. Te explicaremos qué aportan de diferente, sus principales características y qué tipos hay. ¿Empezamos?

1. INTRODUCCIÓN

Para trabajar con datos lo primero que necesitamos es un soporte o sistema donde almacenarlos y poder acceder a ellos para procesarlos.

La forma más básica de almacenar datos es en ficheros. Ya sean ficheros de texto libre, o con campos delimitados (como los omnipresentes CSV) o bien usando formatos binarios más complejos, como las hojas de cálculo, los ficheros son una manera sencilla de guardar nuestros datos en un soporte persistente de donde poder recuperarlos cuando nos hagan falta.

Sin embargo, cuando hablamos de tener una aplicación o un sistema de información más complejo, utilizar simples ficheros tiene sus limitaciones.

El problema principal es el del acceso concurrente: es complejo controlar que varios usuarios puedan acceder a un mismo fichero leyendo y escribiendo a la vez, y garantizar que la estructura y contenido del fichero no se va a corromper. Hay que asegurar que no se sobrescriben, duplican o eliminan datos erróneamente, y que las operaciones de lectura y escritura de distintos usuarios se ejecutan en la secuencia correcta. Además de mantener el formato interno del fichero para poder leer los datos posteriormente.

Otro inconveniente de almacenar los datos en ficheros es que, si queremos buscar un registro en concreto, en general tendremos que ir recorriendo el fichero registro a registro secuencialmente. Si además queremos modificar un valor y actualizarlo en el fichero, muy probablemente tendremos que reescribir todo el fichero de nuevo (a no ser que los registros sean de tamaño fijo). Como ves, estas operaciones pueden ser muy ineficientes.

Estos son algunos de los motivos por los que se desarrollaron los sistemas de bases de datos. En particular, las bases de datos relacionales se han convertido durante décadas en el sistema fundamental de almacenamiento y gestión de datos para la mayoría de aplicaciones.

1.1. LAS BASES DE DATOS RELACIONALES

Una base de datos relacional es un sistema complejo para gestionar información, que permite al usuario guardar, modificar, leer y borrar datos de forma sencilla y uniforme. Existen numerosos sistemas de bases de datos relacionales (Oracle, PostgreSQL, MySQL, ...) y están presentes de una u otra forma en la mayor parte de aplicaciones y sistemas de información.

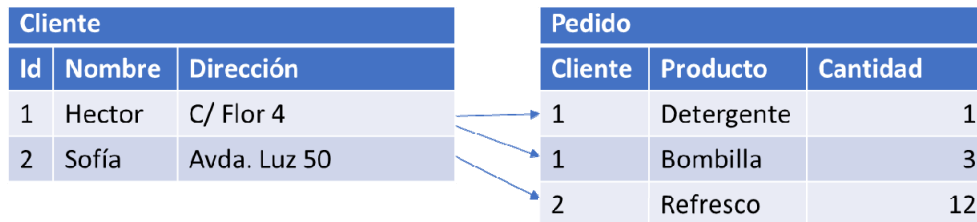
Una de las principales características de las bases de datos relacionales es que separan la lógica de la organización y relación entre las estructuras y tipos de datos de la forma de almacenar físicamente en disco esas estructuras con su contenido.

En las bases de datos relacionales la estructura básica de almacenamiento es la tabla. Una tabla representa una entidad u objeto determinado del que queremos almacenar los datos (p.ej. una tabla de empleados, de productos, etc.).

Las tablas están compuestas por un número fijo de columnas. Cada columna representa una propiedad de una entidad u objeto (p.ej. el nombre, la fecha de nacimiento) Cada columna tiene asociado un tipo de dato, para indicar cuál es la clase válida de valores que puede almacenar. Cada fila de una tabla representa un registro de una entidad particular (un empleado o un producto en concreto). Una tabla puede albergar a priori una cantidad ilimitada de filas.

Además, podemos vincular los datos de tablas distintas que estén relacionados entre sí. Por ejemplo, supón que tenemos una tabla con todos los productos que vendemos en nuestra tienda *online*. La tabla contiene columnas con el código del producto, la marca o el precio. Si tenemos una tabla con la cesta de la compra de un cliente, usaremos una columna con el código de los productos comprados, que corresponderá al código usado en la tabla de productos disponibles. Las bases de datos relacionales permiten definir y controlar cómo están vinculados o relacionados los registros (filas) de dos o más tablas distintas (de ahí el nombre de bases de datos *relacionales*). El motor de la base de datos se asegurará de que una columna que referencie a otra tabla contenga siempre valores válidos y no se rompan los vínculos entre entidades por error. A esto lo denominamos mantener la *integridad referencial*.

Al conjunto de definiciones de tablas, columnas, tipos y relaciones lo denominamos *esquema*.



Cliente			Pedido		
Id	Nombre	Dirección	Cliente	Producto	Cantidad
1	Hector	C/ Flor 4	1	Detergente	1
2	Sofía	Avda. Luz 50	1	Bombilla	3
			2	Refresco	12

Ejemplo de tablas relacionadas

Al separar la organización lógica del formato de almacenamiento físico, las bases de datos relacionales permiten a los usuarios centrarse en definir la estructura de las tablas y en las operaciones de alto nivel con los datos, en lugar de preocuparse de leer y escribir más o menos bytes de un fichero.

Para facilitar estas operaciones, las bases de datos relacionales proporcionan además un lenguaje especialmente diseñado para manipular las estructuras y contenido almacenado: el lenguaje SQL (*Structured Query Language*). Con este lenguaje no sólo podemos crear tablas y modificar su contenido. Nos proporciona también la forma de realizar búsquedas o *consultas* sobre una o más tablas para seleccionar los datos que necesitamos. Se trata de un lenguaje muy potente que nos permite construir expresiones de consulta, transformación y agregación realmente complejas.

El hecho de que el lenguaje SQL disponga de un estándar hace que podamos operar de forma uniforme (o con muy pocos cambios) con distintos motores o sistemas de bases de datos relacionales.

Estas bases de datos también permiten que varios usuarios trabajen concurrentemente sobre los mismos datos. El sistema gestor de la base de datos se encarga de gestionar esta concurrencia para asegurar la coherencia de los datos que ven los usuarios en todo momento y evitar posibles errores. Esto se consigue mediante un mecanismo llamado transacción. Una transacción es un bloque de operaciones de un usuario que se ejecutan de manera atómica, es decir, como si fueran una única operación. El efecto de las operaciones de una transacción no es visible para los usuarios hasta que no se completa toda la transacción. Si se produce algún error en uno de los pasos, la transacción se aborta y el sistema puede volver al estado inicial, como si nada se hubiese ejecutado. Mientras se ejecuta una transacción, es posible que las operaciones de otros usuarios que accedan a las mismas tablas queden bloqueadas hasta que la transacción acabe.

1.2. LIMITACIONES DEL MODELO RELACIONAL

A pesar de todas estas características, las bases de datos relacionales también tienen ciertas limitaciones. Con la explosión de los contenidos y aplicaciones en la web, el volumen de datos a manejar ha ido creciendo de manera exponencial. Las bases de datos tenían que ser capaces de soportar grandes cantidades de operaciones de lectura y escritura concurrentes, tiempos de respuesta muy bajos para que los sitios web funcionaran adecuadamente, y ofrecer alta disponibilidad y tolerancia a errores para evitar caídas de los sitios web.

Las bases de datos relacionales no estaban bien preparadas para hacer frente a estas exigencias. La gran mayoría de estas bases de datos estaban diseñadas para operar en una sola máquina, de manera que para mejorar el rendimiento y adecuar el sistema a un mayor volumen de datos y operaciones, la única alternativa era mejorar el procesador o añadir más memoria o discos duros más grandes y rápidos. El problema es que esta opción es cara, y tampoco funciona indefinidamente. El ritmo de crecimiento de los datos puede ser mucho mayor que la capacidad de ampliar la máquina.

La estrategia alternativa consiste en utilizar varios servidores. Existen bases de datos relacionales que permiten utilizar varias máquinas. Sin embargo, características como mantener la integridad referencial o el mecanismo de transacciones son complejos de implementar en un sistema distribuido si se quiere garantizar al mismo tiempo el acceso concurrente de los usuarios a la misma información en todo momento. Las bases de datos relacionales distribuidas en general son complejas de mantener y utilizar.

Por otro lado, en una base de datos relacional tienes que estudiar tus datos, definir un esquema o estructura, y cargar los datos conforme a ese esquema.

Sin embargo, la mayor parte de los datos que se generan ya no tienen una estructura clara y estática. Podemos tener que almacenar registros con gran variedad de campos distintos entre ellos, o que contengan campos repetidos con múltiples valores, o que vayan evolucionando con el tiempo...

Esta naturaleza no uniforme y dinámica de gran parte de los datos que se generan hoy día no encaja bien con el modelo relacional.

Estas limitaciones impulsaron a distintas empresas y grupos de desarrollo a construir soluciones alternativas que cubrieran las nuevas necesidades de sus sistemas de información, dando lugar a toda la variedad de tecnologías NoSQL existentes en la actualidad.

2. NOSQL

2.1. ¿QUÉ ES NOSQL?

Aunque no existe una definición precisa del término, se suele identificar como NoSQL a aquellas bases de datos que no siguen el modelo relacional.

Como curiosidad, el término tal como se usa actualmente viene de un *hashtag* que se utilizó para etiquetar a algunas de las primeras bases de datos alternativas que se desarrollaron para su uso en aplicaciones de Big Data y tiempo real. Se trataba de bases de datos distribuidas, no relacionales, y ninguna de ellas implementaba el lenguaje SQL. Esto es todo lo que tenían en común. Por lo demás, el modelo de almacenamiento y acceso a los datos, y los problemas en los que se enfocaba cada una de aquellas primeras bases de datos NoSQL no tenían ninguna relación entre sí.



Ojo al dato

Hoy en día existe una enorme variedad de soluciones NoSQL (más de 225 según <http://nosql-database.org/>). Tal cantidad se debe a la diversidad de propósitos y necesidades a cubrir. Tenemos desde bases de datos NoSQL con gran flexibilidad para adaptarse a cualquier tipo de dato, hasta bases de datos especializadas en unos tipos de datos o formas de uso muy concretas, solucionando problemas específicos de manera muy eficiente.

Y aunque hablamos de NoSQL, con el tiempo algunas de estas bases de datos sí que han terminado incluyendo lenguajes de consulta derivados de SQL, por lo que a veces se las denomina “Not only SQL” (*No solamente SQL*) para indicar que opcionalmente pueden soportar esta característica.

En el siguiente apartado vamos a ver qué características comparten la enorme variedad de bases de datos NoSQL.

2.2. CARACTERÍSTICAS

2.2.1. DISTRIBUIDAS

Una de las principales necesidades a cubrir por las bases de datos NoSQL ha sido la de poder gestionar enormes cantidades de información. Para ello han adoptado la misma estrategia que el resto de tecnologías *Big Data* que hemos visto: repartir el trabajo entre múltiples máquinas interconectadas.

A diferencia de sus primas relacionales, las bases de datos NoSQL son distribuidas por naturaleza. Aunque se puede instalar y configurar una base de datos NoSQL en un único servidor, la verdadera potencia de estos sistemas se aprovecha cuando disponemos de un *cluster* de máquinas donde hacerlos correr.

Las bases de datos NoSQL están diseñadas para funcionar de manera coordinada en varios nodos, mientras que para el usuario final es como si se conectara a una base de datos normal en un único servidor.

Escalado horizontal

Como ya te explicamos en la Unidad 11, la escalabilidad horizontal es la capacidad de un sistema informático distribuido para crecer añadiendo más máquinas o *nodos* al conjunto, y así aumentar la capacidad de procesamiento y almacenamiento.

Los sistemas Big Data, como las bases de datos NoSQL de las que hablamos, están diseñados para poder añadir nuevos nodos al sistema sin muchas complicaciones. En la mayoría de casos basta con instalar y configurar el software de la base de datos en la nueva máquina a añadir, y conectarla al resto del *cluster*. Un *cluster* NoSQL puede llegar a estar formado por cientos de máquinas.

Esta facilidad para añadir (o quitar) nodos del sistema nos permite una gran flexibilidad para poder adaptar los recursos de *hardware* (procesador, memoria, disco, conexiones de red...) de nuestro sistema NoSQL a las necesidades reales de carga de trabajo y almacenamiento. Y además no requiere parar la base de datos para aumentar el sistema, por lo que los usuarios y aplicaciones pueden seguir trabajando sin verse afectados.

Particionado

Si disponemos de varios nodos en nuestro sistema NoSQL, es para repartir los datos y carga de trabajo entre ellos, como ya hemos dicho.

Este reparto se lleva a cabo mediante un proceso de particionado horizontal (*sharding* en inglés) que asegura que los datos se distribuyen uniformemente entre los nodos del *cluster*.

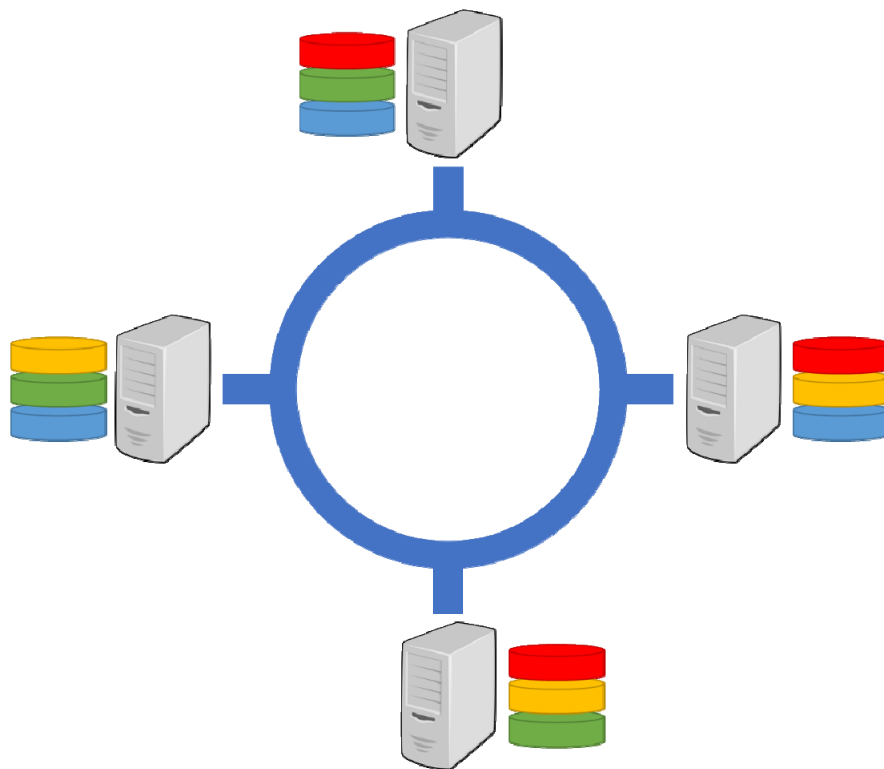
Imagina que el reparto acabara almacenando la mayor parte de los datos en un nodo, mientras el resto apenas guardan información. Cada vez que haya que consultar el contenido, el nodo que concentra la mayoría de los datos tendrá que procesar muchas más peticiones y acabará *sobrecargado*, y enlenteciendo a todo el sistema.

El particionado debe garantizar que la cantidad de datos que se reparten a cada nodo se mantenga equilibrada.

Replicación

Por otro lado, si tenemos un *cluster* NoSQL con varias máquinas repartiéndose los datos, ¿qué pasa si uno de las máquinas no responde? En el mejor de los casos, puede ser una caída de conexión momentánea y sólo se perdería su respuesta con su porción de los datos. En el peor de los casos, el nodo ha podido fallar y perderíamos sus datos de forma permanente (o al menos hasta que podamos recuperarlos de un *backup*).

En las bases de datos NoSQL la solución suele pasar por crear un número suficiente de copias de cada partición o bloque de datos y repartirlas a su vez entre nodos distintos. Al replicar las particiones de esta manera, nuestro sistema NoSQL puede seguir funcionando y respondiendo a consultas aunque una de las máquinas caiga.



Nodos distribuidos de una base de datos NoSQL se reparten bloques de datos, con varias réplicas para evitar pérdidas de información.

2.2.2. FLEXIBILIDAD DE ESQUEMA

En las bases de datos relacionales es necesario definir el esquema de la información a almacenar, es decir, su estructura: las tablas, columnas, tipos... Solamente cuando hemos definido y creado este *esquema* de datos, podemos comenzar a incorporar la información.

Sin embargo, las bases de datos NoSQL son mucho más flexibles. No es necesario conocer la estructura precisa que van a tener nuestros datos con antelación para poder empezar a almacenarlos y trabajar con ellos. Gran parte de los sistemas NoSQL se caracterizan por esta ausencia de esquemas de datos fijos. Simplemente al cargar la información, la base de datos se ocupa de crear las estructuras necesarias para almacenarla.

Es más, esta flexibilidad permite que, si la estructura de nuestros datos va cambiando o evolucionando con el tiempo, podamos incorporarlos igualmente a nuestra base de datos sin necesidad de realizar cambios en el esquema ni actualizar la estructura de los datos antiguos ya cargados. La ausencia de esquemas estáticos hace que las bases de datos NoSQL sean capaces de adaptarse y manejar datos con distintas formas y estructuras.

El beneficio principal de esta ausencia de esquemas estáticos es que da mucha más flexibilidad a los usuarios y programadores de aplicaciones, y permite agilizar el trabajo.

No obstante, no todas las bases de datos NoSQL ofrecen tanta flexibilidad. Como veremos más adelante, existen distintas familias con modelos de datos diferentes. Algunas ofrecen gran flexibilidad y versatilidad, mientras que otras poseen modelos de almacenamiento algo más restringidos (aunque permitiendo cierta adaptabilidad igualmente).

El motivo es que todo tiene un coste, y el precio que se paga por una total flexibilidad es una menor eficiencia a la hora de manejar y recuperar la información. Una estructura fija de datos es mucho más fácil de optimizar para reducir su tamaño, y más sencilla de indexar para encontrar los datos que buscamos. Los distintos tipos de bases de datos NoSQL ofrecen una variedad de compromisos entre cuánta flexibilidad te ofrecen para tu esquema de datos y cómo de optimizado y eficiente es su almacenamiento y recuperación.

2.2.3. ADIÓS AL MODELO RELACIONAL

La característica que define a las bases de datos relacionales es justamente este concepto de *relación*. El hecho de que los registros de una tabla estén relacionados con los de otra tabla distinta, de forma que podamos vincular las tablas para cruzar sus datos y combinarlos. Las bases de datos relacionales están diseñadas para manejar estas relaciones y permitir al usuario construir consultas complejas uniendo tablas relacionadas (las típicas operaciones de tipo JOIN).

La existencia de este mecanismo de relaciones permite poder separar entidades o conceptos distintos en tablas interrelacionadas entre sí, en lugar de tener todos los datos en la misma tabla.

Cliente			Pedido				Producto	
Id	Nombre	Dirección	Cliente	Fecha	Producto	Cantidad	Producto	Precio
1	Hector	C/ Flor 4	1	10/06/2016	Detergente	1	Detergente	6 €
2	Sofía	Avda. Luz 50	1	25/06/2016	Bombilla	3	Bombilla	1.50 €
			1	25/06/2016	Yogur	4	Yogur	1 €
			2	18/07/2016	Refresco	12	Refresco	1.20 €

Ejemplo relacional cliente y pedidos con artículos comprados

Las bases de datos NoSQL se caracterizan por abandonar este modelo relacional. Los sistemas NoSQL no almacenan información de cómo se relacionan las filas de una tabla con otra. El principal motivo es que no tenemos esquemas de datos fijos. En una misma “tabla” NoSQL podemos tener registros con campos diferentes, lo que hace muy complejo que la propia base de datos pueda controlar las relaciones entre registros de distintas tablas.

Esta pérdida del soporte relacional puede parecer una desventaja importante. Y para algunas aplicaciones sí puede ser una limitación. Pero recuerda que el objetivo de las bases de datos NoSQL es distinto de las relacionales: tienen que hacer frente a grandes cantidades de datos sin estructura fija. Para poder permitir esquemas de datos flexibles y mantener la eficiencia al manejar cientos de millones de datos de forma distribuida, las relaciones entre tablas son un problema.

En su lugar, en las bases de datos NoSQL se opta habitualmente por combinar todos los datos en una única “tabla” o colección, anidando o *embebiendo* los diferentes tipos de registros. A esto se le denomina *desnormalización*.

La ventaja de almacenar así los datos es que toda la información está unida directamente, y buscar o manipular valores relacionados es mucho más rápido al no tener que unir o cruzar tablas.

Cliente					
Id	Nombre	Dirección	Pedido		
1	Hector	C/ Flor 4	Fecha	Producto	Cantidad
			10/06/2016	Detergente	1
			25/06/2016	Bombilla	3
			25/06/2016	Yogur	4
2	Sofía	Avda. Luz 50	Fecha	Producto	Cantidad
			18/07/2016	Refresco	12

Ejemplo en NoSQL, pedidos y artículos embebidos

Pero también tiene un inconveniente. Y es que es habitual acabar teniendo la misma información duplicada en varias tablas distintas. Esto es un problema no tanto por el espacio de almacenamiento extra para las múltiples copias de los datos (el espacio de almacenamiento cada vez es más barato). El problema principal reside en que, al modificar un registro, hemos de asegurarnos de que actualizamos todas las copias en las diferentes tablas para que la información sea coherente. Y hemos de hacerlo nosotros, la base de datos no lo hará automáticamente.

Por otro lado, que las bases de datos NoSQL no den soporte propio a las relaciones no significa que no podamos unir los datos de distintas tablas o colecciones. Simplemente tendremos que hacerlo nosotros mismos con código, mientras que en las bases de datos relacionales es el propio sistema el que nos permite hacerlo.

Esto es algo que aplica también a otras diferencias entre relacional y NoSQL. Funcionalidades presentes en las bases de datos relacionales y accesibles a través del lenguaje SQL, cuando pasamos a NoSQL tendremos que implementarlas nosotros mismos con nuestro lenguaje de programación favorito (¿R o Python tal vez?).

2.2.4. SIN LENGUAJE COMÚN

Las diferentes bases de datos relacionales utilizan un lenguaje estándar común: SQL (*Structured Query Language*, o lenguaje de consulta estructurado). El lenguaje SQL permite crear objetos en la base de datos (tablas, vistas, columnas, índices...) y realizar consultas sobre una o más tablas, devolviendo resultados también en forma tabular.

En el mundo NoSQL no existe un lenguaje común para manejar los datos. Las diferencias tan grandes que existen entre las numerosas bases de datos NoSQL hacen que no sea fácil disponer de un único lenguaje capaz de dar cabida a la variedad de modelos de datos y disparidad de funcionalidades.

Cada base de datos NoSQL suele desarrollar su propio lenguaje de consultas, adaptado a las características propias del sistema. En muchos casos se utilizan expresiones o librerías directamente derivadas de lenguajes de programación como Javascript, Java, Perl o Python.

Pero también es cada vez más común que las bases de datos NoSQL ofrezcan un lenguaje de consulta derivado o basado en la sintaxis de SQL. Esto facilita la adaptación al usuario final que ya está familiarizado con el lenguaje SQL, incluso si la versión NoSQL no sigue perfectamente al estándar y tenga diferencias.

Además, existe una serie de herramientas que podemos llamar *motores de consulta*, cuya misión consiste en proporcionar una capa de acceso común por encima de distintas bases de datos. Estos motores de consulta ofrecen una interfaz con un lenguaje único para el usuario final. Es el propio motor quien se encarga de traducir esas consultas al lenguaje que *entiende* cada tipo de base de datos, a través de controladores o *drivers* específicos. Algunos de los principales, como Apache Drill o Presto, proporcionan un lenguaje común de consulta basado en SQL.



Resumiendo

Las principales características de las bases NoSQL:

- Son distribuidas por diseño.
- Permiten esquemas de datos flexibles.
- No utilizan el modelo relacional.
- No tienen un lenguaje común para consultas.

2.3. PROPIEDADES Y TEOREMA CAP

No nos hemos cansado de repetir que uno de los principios básicos de los sistemas Big Data en general, y las bases de datos NoSQL en particular, es que son sistemas distribuidos en los que muchas máquinas interconectadas se reparten la carga de datos y cálculo de forma coordinada.

En un sistema de almacenamiento de datos distribuido podemos identificar tres propiedades deseables:

2.3.1. CONSISTENCIA

Explicado de modo sencillo, esta propiedad viene a significar que en todo momento cualquier usuario que haga una consulta tendrá como resultado la última versión de los datos escritos. En un sistema distribuido donde los datos se reparten y replican entre varios nodos esto no siempre es fácil de garantizar.

Es posible que los últimos datos cargados se hayan escrito correctamente en uno de los nodos, pero tal vez otro nodo todavía no ha podido actualizarse (porque su conexión de red esté yendo más lenta, o haya más carga de otras tareas que ocupen tiempo de procesador...). Por tanto, la versión de los datos que devolverá este nodo rezagado será la versión previa a la última escritura (que aún no ha podido completar).

Si varios nodos de la misma base de datos devuelven versiones distintas de un registro, ¿cómo debe comportarse la base de datos? ¿Debe esperar a que todos los nodos coincidan con la misma versión antes de responder? ¿Debe devolver la versión mayoritaria? ¿O la más reciente, aunque sólo provenga de un nodo? ¿Debe devolver un error temporalmente?

Exigir una consistencia perfecta (que todos los nodos tengan siempre la misma versión de los datos antes de responder a ninguna consulta) puede ser imprescindible para algunas aplicaciones que requieran certeza total sobre los datos que manejan. Sin embargo, poner de acuerdo a todos los nodos y confirmar que todos comparten la misma visión de los datos tiene un coste en términos de tiempo de respuesta.

En muchas aplicaciones, sobre todo en aplicaciones interactivas en Internet, lo importante es que el tiempo de respuesta sea rápido para que aplicación sea ágil y no parezca que se cuelga. En estos casos no es un problema muy grave que unos usuarios vean los últimos datos disponibles, mientras otros reciben datos todavía sin actualizar del todo, siempre y cuando en algún momento (tal vez unos segundos, tal vez un par de minutos...) la versión más reciente de los datos acabe siendo la que todos reciban si vuelven a consultar.

A esta forma de funcionar la llamamos *consistencia eventual*.

2.3.2. DISPONIBILIDAD

Cuando hablamos de disponibilidad en bases de datos distribuidas nos referimos a la disponibilidad de los datos.

En concreto, esta propiedad se refiere a garantizar que una petición o consulta siempre obtendrá una respuesta con datos válidos.

Imagina que hay un fallo en una parte de la red que conecta los nodos de nuestra base de datos distribuida y estos quedan separados en dos partes. Si dos usuarios se conectan a nodos en mitades distintas y lanzan una consulta, un sistema disponible debería responderles justamente con los datos que tenga disponibles.

Fíjate que no decimos nada acerca de que sea la versión más reciente. Simplemente decimos que el sistema responderá inmediatamente al usuario con los datos disponibles, sin bloquearse, sin esperar al resto de nodos y sin errores.

En este caso, es posible que los dos usuarios reciban versiones diferentes (inconsistentes) de los datos.

2.3.3. TOLERANCIA AL PARTICIONADO

Acabamos de poner un ejemplo en el que los nodos de nuestra base de datos distribuida quedan divididos en dos mitades por un problema de red. Ojo, no tienen porqué ser mitades iguales; llevado al extremo puede que quede aislado un único nodo del resto del *cluster*.

¿Cómo debería comportarse la base de datos partida en dos? ¿Deberían seguir funcionando ambas partes y atendiendo peticiones de los usuarios? ¿O debería el sistema dar un error y cerrarse ordenadamente?

Y cuando las conexiones de red se reestablezcan, ¿deberían las mitades reunificarse y operar normalmente de nuevo?

Un sistema que es capaz de mantener sus mitades funcionando de forma independiente ante un caso así, y que finalmente es capaz de recuperarse y volver a operar como un único *cluster* decimos que posee *tolerancia al particionado*.

Piensa en las implicaciones que tiene un sistema tolerante al particionado. No sólo significa que las partes desconectadas son capaces de seguir funcionando y respondiendo consultas hasta que vuelvan a fusionarse.

¿Qué pasa si en una de las mitades un usuario añade o modifica datos? ¿Y si en la otra mitad un usuario distinto modifica los mismos datos? Cuando ambas mitades se reúnan de nuevo, el sistema tendrá versiones distintas en los nodos provenientes de cada partición. El sistema no solo debe ser capaz de reconectar los nodos. También debe unificar la versión de los datos en todos los nodos hasta dejar una imagen consistente.

2.3.4. TEOREMA CAP

El teorema CAP fue enunciado por Eric Brewer en el año 2000 (también se lo denomina teorema de Brewer) en relación con las propiedades que acabamos de discutir de los sistemas de datos distribuidos.

Las siglas CAP vienen precisamente de los términos en inglés: *Consistency*, *Availability* (disponibilidad) y *Partition tolerance* (tolerancia al particionado).

Este teorema determina que una base de datos distribuida no puede cumplir al mismo tiempo con las tres propiedades de manera total. Siempre habrá una de las características que deberá ser descartada total o parcialmente. Toma dos, descarta la restante.

La siguiente figura muestra un diagrama con las combinaciones de propiedades posibles.

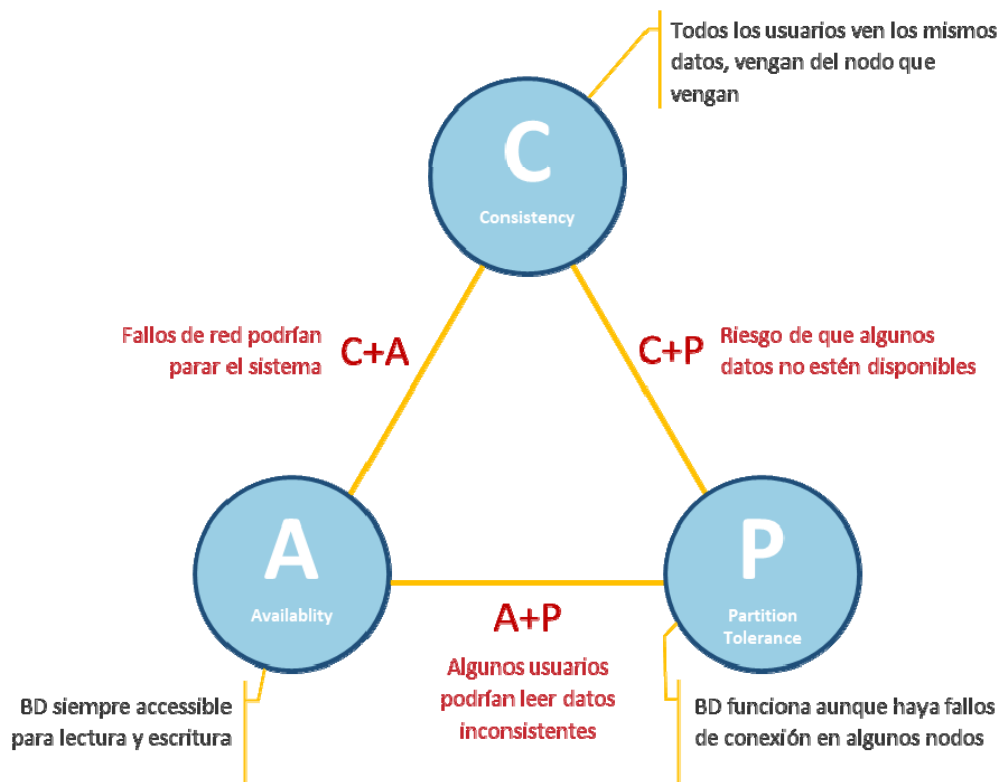


Diagrama resumen del teorema CAP

Las bases de datos NoSQL se caracterizan por ser distribuidas, por lo que en general suelen ofrecer tolerancia al particionado. Así que en general, habrá que decidirse por sistemas NoSQL que garanticen también la consistencia (sistemas C+P) o por sistemas que opten por primar la disponibilidad (A+P).



Resumiendo

Existen tres propiedades de un sistema de datos distribuido:

- Consistencia.
- Disponibilidad (*Availability*).
- Tolerancia al particionado (*Partition tolerance*).

El teorema CAP determina que estos sistemas no pueden satisfacer las tres propiedades simultáneamente.

3. FAMILIAS NOSQL

A lo largo de la unidad hemos comentado varias veces que existe un gran número de bases de datos NoSQL, con diferentes características y funcionalidades que las hacen más aptas para distintas clases de problemas o tipos de datos.

Las bases de datos NoSQL permiten almacenar gran variedad de datos, con estructuras simples o complejas, incluso datos cuya estructura cambia con el tiempo y en formas que no podemos anticipar.

No obstante, si nos centramos en los modelos de datos, podemos clasificar las bases de datos NoSQL en cuatro grandes grupos

- Clave-Valor.
- Documentos.
- Columnas.
- Grafos.

Vamos a ver las características de cada grupo.

3.1. CLAVE-VALOR

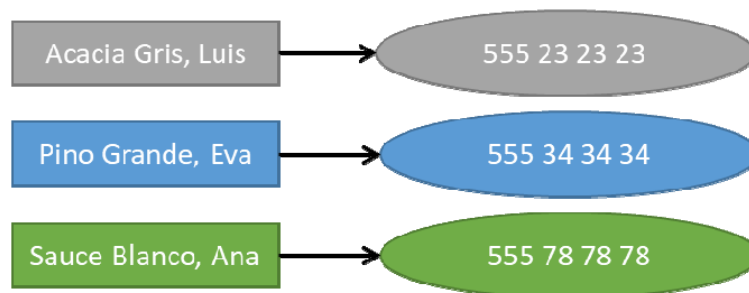
Las bases de datos de tipo clave-valor son la familia más sencilla de base de datos NoSQL. Como su nombre indica, el modelo de datos consiste en dos elementos: una clave y un valor.

Las claves son identificadores únicos a los que asociamos un valor u objeto a almacenar. Gráficamente, puedes imaginar las claves como etiquetas con un nombre o un código que ponemos a distintos objetos para poderlos identificar.

Utilizando la clave, podemos pedirle a la base de datos que nos devuelva directamente el objeto asociado, sin necesidad de examinar uno a uno todos los objetos hasta encontrar el que deseamos.

Las claves suelen ser valores numéricos o cadenas de texto, siguiendo cualquier convención de nombrado que se adecúe al problema. En algunos casos pueden formarse usando tipos o estructuras de datos más complejos, siempre que puedan compararse entre sí para asegurar que las claves son únicas

Respecto a los valores asociados a las claves, en general pueden ser elementos de cualquier tipo que deseemos: números, texto, listas, matrices, imágenes u otros objetos binarios.



La única forma de consulta en una base de datos clave-valor es a través de la clave. Solo podemos acceder de forma directa a un dato conociendo su clave o identificador correspondiente. No podemos buscar preguntando por un valor particular, o por un rango de valores. Visto así, puedes verlo como un directorio o un diccionario: buscas la palabra para obtener su definición o significado, y no al revés.

Ejemplos de bases de datos clave-valor son Redis (<https://redis.io/>) y Riak (<http://basho.com/products/riak-kv/>).

3.2. DOCUMENTOS

Las bases de datos orientadas a documentos utilizan también un modelo clave-valor para almacenar los datos, pero con algunas diferencias importantes.

Para empezar, los elementos que se almacenan como valores son documentos.

En este punto tenemos que detenernos un momento para explicar qué entendemos por documento en este contexto NoSQL. Un documento, en el mundo NoSQL y en general hablando de aplicaciones y servicios web, es una representación de datos en algún tipo de formato estándar, típicamente textual, como JSON o XML. A este tipo de datos se les denomina habitualmente *semiestructurados*, ya que, aunque están organizados en campos con tipos y valores, la estructura puede variar, apareciendo registros con distinto número de campos y valores.

En cualquier caso, dejamos claro que cuando hablamos de documentos, no nos referimos a documentos de texto tipo .doc o PDF.

En una base de datos orientada a documentos, en lugar de almacenar cada valor o atributo individual con una clave independiente, solemos encapsular los atributos de un mismo objeto en un documento y almacenarlo entero vinculado a su identificador.

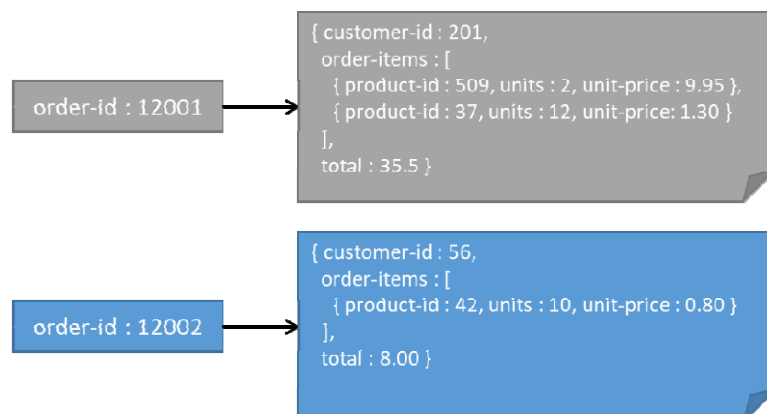
Una de las características más importantes de las bases de datos orientadas a documentos es que no es necesario definir el esquema interno de los documentos a almacenar. Al añadir un nuevo documento, la base de datos se encarga directamente de procesarlo, detectar los campos y almacenarlo con la estructura interna apropiada.

Esta ausencia de esquema fijo permite a los usuarios y desarrolladores total libertad para gestionar la información. Distintos registros de un tipo en la misma colección de datos pueden tener diferentes atributos sin que haya conflictos ni errores.

No obstante, un documento JSON o XML no deja de ser un valor que podríamos almacenar también en una base de datos clave-valor. ¿Cuál es la diferencia?

Pues una fundamental. Al especializarse en documentos, estas bases de datos pueden procesar el contenido y almacenarlo internamente en estructuras de datos especiales. Gracias a esto, las bases de datos orientadas a documentos sí que permiten realizar búsquedas sobre el contenido de los documentos (las bases de datos clave-valor no permiten buscar sobre los valores).

Por ejemplo, podemos buscar los documentos que tengan cierto atributo con un determinado valor.



La base de datos NoSQL para documentos más popular es MongoDB (<https://www.mongodb.com/>). De hecho es la más popular de todas las bases de datos NoSQL en general (fuente: <https://db-engines.com/en/ranking>). MongoDB no solo es ampliamente usada en entornos Big Data, sino que también es un referente en el desarrollo de aplicaciones y servicios web. Otras bases de datos orientadas a documentos son CouchDB (<http://couchdb.apache.org/>) y Couchbase (<https://www.couchbase.com/>).

3.3. FAMILIA DE COLUMNAS

En el modelo de tablas de las bases de datos relacionales, cada registro se almacena como una fila de una tabla, donde cada columna corresponde a un campo o atributo del registro. El número de columnas (o atributos), sus nombres y tipos de datos están predefinido y es el mismo para todos los registros. Típicamente, las bases de datos relacionales están optimizadas para manejar tablas con gran número de filas (cientos de miles o millones) y un número acotado de columnas (hasta unas pocas decenas).

¿Qué ocurre si tenemos registros con múltiples valores de un atributo? ¿Y si el valor de un atributo cambia con el tiempo y queremos conservar la evolución de valores? ¿Y si las entidades a almacenar presentan una enorme variedad de atributos (cientos, miles) pero cada registro en particular solamente tiene datos de unos pocos, que varían de registro a registro?

En los dos primeros casos tendríamos que poder añadir múltiples copias o versiones de una columna. Las bases de datos relacionales no permiten esta forma de ampliar dinámicamente las columnas para un único registro. En su lugar, esto se solucionaría separando el atributo que puede tener múltiples valores y llevándolo a una tabla distinta, para almacenar tantas filas como versiones del atributo aparezcan.

Ciudad		Tiempo			
Id	Nombre	Id Ciudad	Fecha	Variable	Medida
1	París	1	10/06/2016	Temperatura	23
2	Roma	1	25/06/2016	Temperatura	26
		1	25/06/2016	Lluvia	0.5
		2	18/07/2016	Temperatura	32

Ejemplo de tabla auxiliar en un modelo relacional para almacenar distintos atributos con múltiples valores

En el tercer caso, tendríamos una tabla con cientos o miles de columnas predefinidas. Muchas bases de datos relacionales no están preparadas para ello. Pero incluso en las que sí lo están, tenemos un problema. La tabla tendrá una estructura fija, pero en cada fila (registro) solamente unas pocas columnas (atributos) tienen datos, y el resto están vacías. A esto se le denomina *dispersión de los datos* y supone un gran desperdicio de espacio al tener que almacenar *celdas* de la tabla aunque no tengan valores.

El problema de las celdas vacías en una tabla de estructura estática

De nuevo, la solución habitual en el mundo relacional es tener una tabla aparte en la que guardaremos únicamente los atributos con valores de cada registro, almacenando el nombre del atributo y su valor (una tabla clave-valor por filas).

Como ves, la receta para manejar estos casos con bases de datos relacionales consiste en separar los atributos que pueden aparecer múltiples veces (o no aparecer) y llevarlos a una tabla organizados por filas, en lugar de por columnas. A la hora de buscar y recuperar los datos, cruzaríamos las tablas (mediante operaciones tipo JOIN) para combinar las tablas y obtener la información que queremos.

Ahora volvamos al mundo Big Data. Tenemos decenas o cientos de millones de registros que almacenar, y no paran de crecer. Nuestra base de datos relacional no nos sirve y tenemos que pasar a una NoSQL distribuida. Pero aquí ya no tenemos relaciones, ya no podemos hacer JOINs.

Para manejar este tipo de datos tenemos las bases de datos NoSQL de columnas extendidas (*wide column stores*) o de familias de columnas.

En estas bases de datos cada fila tiene un identificador único y las columnas se almacenan de forma parecida a los pares clave-valor, donde la clave es el nombre de la columna.

Al acceder a una columna mediante una clave, podemos añadir más columnas a una fila en concreto agregando un nuevo par {nombre de nueva columna, valor}. Eso sí, siempre que el nombre de la nueva columna no esté ya en uso en esa fila.

Otra ventaja es que cada registro solamente almacena las columnas para las que tiene un valor. No se desperdicia espacio en atributos vacíos.

Ciudad				
Id	Nombre	Tiempo		
1	París	Temperatura 10/06/2016	Temperatura 25/06/2016	Lluvia 25/06/2016
		23	26	0.5
2	Roma	Temperatura 18/07/2016		
		32		

Ejemplo de cómo almacenar columnas con pares {nombre de columna, valor}, usando el nombre para guardar también información

Además, en estas bases de datos podemos definir familias o categorías de columnas que estén relacionadas entre sí (por ejemplo, por representar el mismo concepto o por ser distintas propiedades de un mismo tipo).

Otra ventaja de manejar así las columnas es que podemos utilizar la propia clave que sirve de nombre de columna para almacenar un dato útil. Por ejemplo, para guardar el histórico de facturas de un cliente podemos crear una familia de columnas de facturación, y almacenar el importe de cada nueva factura en una columna, usando la fecha de la factura como clave-nombre de columna.

Row Id	Column Family : Client Data					Column Family : Contact Data			Column Family : Accounts		
Client Id	Client	First Name	Last Name	Age	Sex	Contact	Address	Postcode	Position	Deposit	Loan
010		Teresa	Roma	35	Female		Av.Marina 10	08910		010032	010960
										3500 €	-1300 €
Client Id	Client	First Name	Last Name	Sex		Contact	Phone No	Postcode	PO Box	Position	Loan
203		Roger	Garcia	Male			555-0897	27080	5025		203550
											-485 €

Familias de columnas, con columnas variables

Podemos ver este modelo de datos como una especie de tablas con varios niveles anidados y flexibles. Las bases de datos de familias de columnas pueden tener un esquema de alto nivel para los datos que almacenan, indicando las familias de columnas, y qué tipo de datos pueden almacenar las columnas de una familia. En estos casos, seguiremos teniendo la flexibilidad de poder añadir columnas arbitrariamente a registros concretos, sin afectar a la estructura de alto nivel ni al resto de filas de la tabla.

Dos de las principales bases de datos con modelo de columnas extendido son HBase (<https://hbase.apache.org/>) y Cassandra (<http://cassandra.apache.org/>).

3.4. GRAFOS

Las bases de datos de grafos son un tipo aparte en NoSQL, con bastantes diferencias y mucho más especializado respecto al resto de tipos que hemos visto.

Para refrescar la memoria, un grafo es una forma de representar un conjunto de objetos que denominamos vértices o nodos, y cómo están relacionados entre sí, representando las uniones o relaciones entre elementos mediante enlaces o arcos.

Piensa en un grupo de personas y las relaciones que tienen entre ellos: familiares, amigos, compañeros de trabajo... Esa *red social* la podemos trasladar a un grafo para estudiar mejor sus características y cómo se *interrelacionan* entre sí los miembros de la red.



Ilustración de un grafo

Trasladar un grafo a un modelo de tablas relacional o a alguno de los tipos NoSQL que acabamos de ver es posible, pero a la larga no es muy práctico. Aunque las bases de datos relacionales permitirían representar las relaciones entre nodos, las operaciones básicas para recorrer y explorar grafos son complicadas de implementar (encadenando JOINS). Y conforme el tamaño del grafo aumenta (número de vértices y enlaces entre ellos), cada vez es más complejo e ineficiente.

El objetivo de las bases de datos de grafos es justamente poder almacenar y manejar de manera eficiente información de grafos de gran tamaño. Estas bases de datos ofrecen todas las operaciones de exploración y cálculo avanzado con grafos de forma optimizada para asegurar un buen rendimiento.

Dentro de las bases de datos orientadas a grafos, probablemente la más conocida es Neo4j (<https://neo4j.com/>), pero hay más, como Infinite Graph:

(<https://www.objectivity.com/products/infinitegraph/>) o InfoGrid (<http://infogrid.org>)

3.5. MULTIMODELO

Hemos visto los principales modelos de datos en los que se agrupan la mayoría de bases de datos NoSQL. Dependiendo de cómo sean los datos con los que tenemos que trabajar y el tipo de consultas y procesos que vayamos a ejecutar, nos convendrá más usar un tipo de base de datos u otro.

No obstante, en los últimos años han surgido también varias bases de datos que permiten trabajar con distintos modelos de datos a la vez. De esta forma el usuario puede almacenar y manejar datos en diferentes formas sin necesitar varios sistemas.

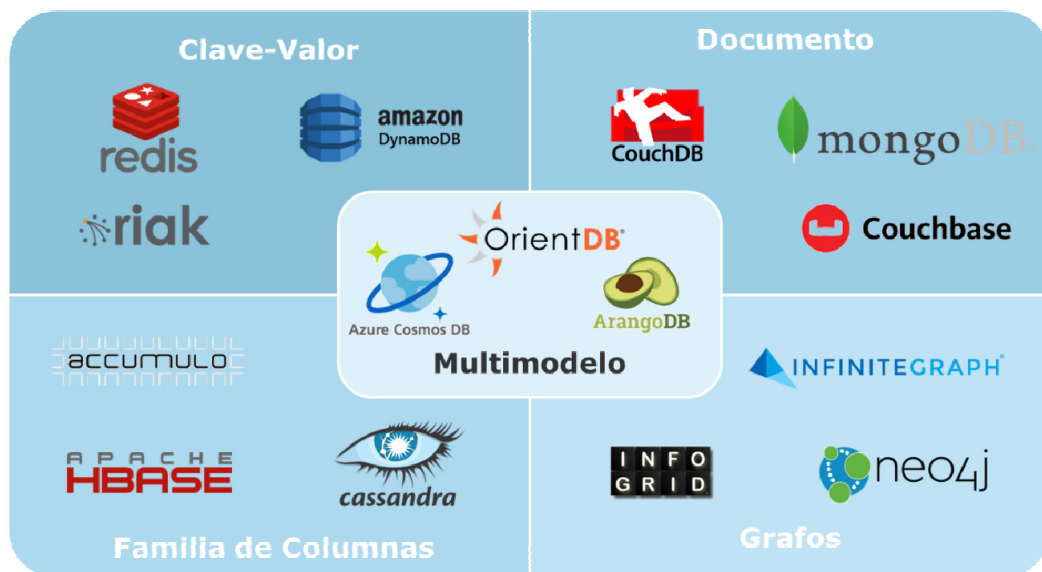
Obviamente, estas bases de datos no son la panacea. Aunque permiten usar distintos modelos de datos según nos convenga, la realidad es que internamente suelen estar optimizadas para trabajar con un modelo en concreto sobre el que se construyen o traducen todos los demás, por lo que la eficacia y rendimiento en estos casos suele resentirse. En cualquier caso suponen una buena solución cuando necesitamos flexibilidad total y no somos muy exigentes con el rendimiento.

Algunas de las principales bases de datos multimodelo son Cosmos DB:

- (<https://azure.microsoft.com/es-es/services/cosmos-db/>)

y OrientDB:

- (<https://orientdb.com/>) o ArangoDB (<https://www.arangodb.com/>).



Resumen de tipos NoSQL y ejemplos de bases de datos

¿QUÉ HAS APRENDIDO?

En esta unidad hemos visto qué son las bases de datos NoSQL. Hemos visto por qué surgieron, como solución a las limitaciones de las bases de datos relacionales para enfrentarse a la avalancha *Big Data*.

Has aprendido también cuáles son las principales características comunes de las bases de datos NoSQL y las propiedades deseables que deben tener. Pero también un teorema fundamental que nos dice que no podemos tener todas esas propiedades a la vez: el teorema CAP.

Finalmente, hemos explorado los distintos tipos de bases de datos NoSQL, en función del modelo de datos que utilizan.

Ahora ya tienes los criterios básicos para decidir qué base de datos utilizar, y que se adecúe mejor a tus necesidades en cada caso.

AUTOCOMPROBACIÓN

1. **¿Cuáles son las limitaciones de utilizar ficheros para almacenar datos?**
 - a) El usuario tiene que controlar el acceso concurrente al contenido.
 - b) En general, buscar un registro concreto requiere recorrer el fichero secuencialmente.
 - c) En general, modificar el contenido de un registro suele requerir reescribir todo el fichero.
 - d) Todas las anteriores *.

2. **¿Qué limitaciones presentaban las bases de datos relacionales para su uso en Big Data?**
 - a) Complejidad para escalar horizontalmente, de forma distribuida, y un modelo de datos poco dinámico y flexible *.
 - b) Están diseñadas para trabajar en una sola máquina.
 - c) No permiten almacenar registros con gran variedad de campos distintos entre ellos, o que contengan campos repetidos con múltiples valores.
 - d) Mantener la integridad referencial no es posible en una base de datos distribuida.

3. ¿Qué son las bases de datos NoSQL?

- a) Una familia de bases de datos específicas para su uso en Big Data, junto con Hadoop o Spark.
- b) Bases de datos que no usan ningún lenguaje de consultas, como SQL.
- c) Unas nuevas bases de datos que reemplazan a las bases de datos relacionales.
- d) Bases de datos que no siguen el modelo relacional *.

4. ¿Cuáles son las características comunes de las bases de datos NoSQL?

- a) Son distribuidas por diseño, utilizan esquemas de datos estáticos basados en modelos no relacionales, y no tienen un lenguaje común de consulta.
- b) Son distribuidas por diseño, utilizan esquemas de datos estáticos con modelos no relacionales, y utilizan lenguajes de programación como Javascript y otros para manejar y consultar los datos.
- c) Son distribuidas por diseño, permiten esquemas de datos flexibles, no siguen el modelo relacional, y no tienen un lenguaje común de consulta *.
- d) Hay multitud de bases de datos NoSQL diferentes, y no tienen características comunes.

5. ¿Qué mecanismo de recuperación utilizan las bases de datos NoSQL si un nodo falla para salvaguardar la información y poder seguir funcionando?

- a) Tener varias réplicas de las particiones o bloques de datos y repartir las copias entre nodos distintos *.
- b) Tener réplicas de todas las particiones o bloques de datos en todos los nodos.
- c) Tener un nodo maestro con una réplica de todas las particiones o bloques de datos.
- d) Tener un backup adicional a los nodos.

6. ¿Qué es la consistencia en el teorema CAP?

- a) Que un usuario obtenga siempre los mismos datos como resultado de una consulta.
- b) Que toda consulta obtenga siempre la última versión disponible de los datos almacenados *.
- c) Que la base de datos responda con la versión mayoritaria de los datos almacenados en los nodos.
- d) Que la base de datos espere a que todos los nodos coincidan con la misma versión de los datos antes de responder a una consulta.

7. ¿Qué dice el teorema CAP?

- a) Que una base de datos distribuida no puede cumplir completamente y al mismo tiempo con las tres propiedades CAP *.
- b) Que una base de datos distribuida no puede cumplir las propiedades CAP.
- c) Que una base de datos distribuida ha de cumplir las tres propiedades CAP.
- d) Que una base de datos distribuida solo puede cumplir completamente una de las propiedades CAP.

8. ¿Cómo se almacena la información en una base de datos de tipo clave-valor?

- a) Mediante pares de una clave o identificador y un valor o dato asociado, ambos pueden repetirse.
- b) Mediante pares de una clave de tipo numérico o textual, y un valor o dato asociado de cualquier tipo.
- c) Mediante pares de una clave o identificador único y un valor o dato asociado *.
- d) Mediante pares de una clave o identificador y el valor o dato asociado, ambos deben ser únicos.

9. ¿Qué ventaja aportan las bases de datos orientadas a documentos frente a las clave-valor?

- a) Ninguna, son más restrictivas, ya que las clave-valor pueden almacenar datos que no sean documentos.
- b) Permiten almacenar documentos sin un esquema interno predefinido.
- c) Permiten realizar búsquedas sobre el contenido de los documentos *.
- d) Ninguna, en realidad son equivalentes.

10. ¿Cuál de las siguientes afirmaciones no es correcta? Las bases de datos multimodelo...

- a) Combinan internamente varios motores de base de datos para manejar los distintos tipos de modelos *.
- b) Almacenan internamente la información usando un único modelo de datos, sobre el que construyen o traducen los demás.
- c) Son una solución adecuada si necesitamos trabajar con distintos modelos de datos.
- d) Son una solución adecuada si no nos importa sacrificar rendimiento en favor de la flexibilidad.

SOLUCIONARIO

1.	d	2.	a	3.	d	4.	c	5.	a
6.	b	7.	a	8.	c	9.	c	10.	a

BIBLIOGRAFÍA

Bibliografía general sobre NoSQL

- NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Sadalage, Pramod; Fowler, Martin. Addison-Wesley, 2012.
- NoSQL for mere mortals. Dan Sullivan. Addison-Wesley, 2015.

Bases de datos NoSQL

- Redis. <https://redis.io/>
- Riak. <http://basho.com/products/riak-kv/>
- MongoDB. <https://www.mongodb.com/>
- CouchDB <http://couchdb.apache.org/>
- Couchbase (<https://www.couchbase.com/>)
- HBase <https://hbase.apache.org/>
- Cassandra <http://cassandra.apache.org/>
- Neo4J <https://neo4j.com/>
- Infinite Graph <https://www.objectivity.com/products/infinitegraph/>
- InfoGrid <http://infogrid.org>

Otros recursos NoSQL

- <http://nosql-database.org/>

