# Introducción al business intelligence y al big data

# Bases de datos NoSQL: Características

Hola, bienvenidos a una nueva presentación
dentro del curso de Introducción

al business intelligence,
en la que vamos a profundizar en las características

de las bases de datos NoSQL. Esta presentación
la hemos efectuado Jordi Conesa

y yo misma, Elena Rodríguez.
Ambos somos profesores

de los Estudios de Informática,
Multimedia y Telecomunicación

de la Universitat Oberta de Catalunya.

Esta transparencia resume
las principales características

de las bases de datos NoSQL
y que hemos visto un poco en un vídeo anterior.

Principalmente la primera
de estas características es que no hay

un modelo de datos único.

Las bases de datos NoSQL proporcionan

un esquema de datos flexible. En general,
no existe un lenguaje estándar,

como sería el caso de SQL en el caso
de bases de datos relacionales.

En general, se trata de bases de datos
que son distribuidas; acostumbran,

a efectos de gestión y transacciones,
a no garantizar las propiedades

del modelo ACID, y frecuentemente
son de código abierto. Bajo el paraguas

de NoSQL subyacen principalmente
dos familias de modelos de datos:

los modelos de agregación, que a su vez
se subdividen en el modelo clave-valor,

el orientado a columnas y el orientado
a documentos; y el modelo de datos en grafo.

Desde un punto de vista de expresividad semántica,
el modelo más sencillo es el clave-valor

y el más potente, desde un punto
de vista semántico, es el modelo de datos

en grafo.

En otras palabras, el modelo de datos en grafo
es el que ofrece más constructores

para poder definir de forma más precisa

la semántica del esquema de la base de datos.

Los modelos de datos en grafo
son útiles para tratar dominios

donde se establecen múltiples y complejas
interrogaciones entre los objetos

del mundo real que deseamos tratar.

Para representar los datos asociados utiliza
estructuras en grafo, es decir, vértices y aristas

que se comunican entre sí,
para representar estas interrelaciones.

Por su parte, los modelos de agregación
son útiles para representar tanto datos

no estructurados como semiestructurados
y se basan en el concepto de agregado.

Un agregado no es más que un conjunto
de objetos del mundo real

que están relacionados entre sí
y que deseamos tratar como una unidad indivisible

a efectos, en primer lugar,
de acceso y manipulación.

Esto significa
que la unidad mínima de intercambio

entre los programas de aplicación
y el gestor de la base de datos es el agregado.

En segundo lugar, el agregado

es también la unidad mínima

a efectos de control
de concurrencia y de integridad.

Esto significa que el gestor de la base
de datos garantiza que cualquier

cambio efectuado sobre el agregado
será definitivo sobre la base de datos.

Y, finalmente, si la base de datos es distribuida,
el agregado es la unidad mínima

a efectos de distribución de los datos.
Cada agregado, en estos modelos,

se identifica por una clave.
Si entramos un poco más en detalle en cada uno

de los modelos, el modelo clave-valor
ve al agregado como un objeto opaco.

Esto significa que,
si existe una estructura para el objeto,

para el agregado, esta estructura es desconocida
por el gestor de la base de datos,

es decir, la estructura únicamente
será conocida por los programas de aplicación

que acceden a la base de datos.
En el caso de los modelos de agregación

orientados a documentos, el agregado
se estructura en forma de documento

utilizando para ello XML o JSON.
En este caso,

sí que existe una estructuración
del contenido del documento, es decir,

podemos definir atributos y el gestor
de la base de datos es consciente

de esta estructuración que estamos definiendo,
de tal manera que se puede acceder

a partes del documento, se pueden definir

índices, por ejemplo, sobre algunos
de estos atributos, etcétera.

Por último, el modelo de agregación
orientado a columnas

estructura los agregados en primera instancia
en forma de filas y luego a través de columnas

o conjuntos de columnas, que se denominan
familias de columnas.

Esto puede parecer similar a una base
de datos relacional, pero aquí se acaban

las coincidencias. Es cierto que permiten
ver una visión matricial de los datos,

pero cada uno de los agregados, es decir,
cada una de las filas, puede definir

sus propias columnas o familias de columnas
y, además, también se permite o se facilita

la existencia de versiones para parte
de estas columnas o familias de columnas.

Los diferentes modelos de datos
que acabamos de ver frecuentemente

se utilizan para clasificar las diferentes
implementaciones comerciales

de bases de datos NoSQL.

BerkeleyDB, actualmente propiedad de Oracle,
Riak o Redis, inicialmente

utilizada por Instagram, son ejemplos
de bases de datos NoSQL clave-valor.

Como ejemplos de bases de datos NoSQL
orientadas a columnas, podemos citar

Cassandra, HBase o Amazon SimpleDB.
Entre las bases de datos NoSQL

orientadas a documentos más conocidas,
están MongoDB, Couchbase o MarkLogic

y, finalmente, entre las bases de datos

NoSQL en grafo podríamos citar
Neo4J, InfiniteGraph o Sparksee,

siendo esta última una base de datos
NoSQL en grafo desarrollada

por una empresa española. En este punto
es importante resaltar que en ocasiones

y dependiendo de las fuentes que consultemos,

podemos ver que una misma base

de datos NoSQL se clasifica
en más de un grupo; esto es debido

a que los desarrolladores de estos productos
van incorporando poco a poco

nuevas funcionalidades de tal manera que,
inicialmente, podrían estar clasificadas

en una de las familias
y a lo largo del tiempo podría variarse

la clasificación.

Este sería el caso, por ejemplo,
de DynamoDB, desarrollada por Amazon,

que inicialmente

fue concebida como una base de datos NoSQL
clave-valor, pero que a día de hoy

permite estructurar el contenido
de los agregados,

de tal manera que se considera orientada a documentos.
En otros casos, directamente, el fabricante

o el desarrollador define su base de datos
NoSQL como multimodelo.

Este sería el caso, a modo de ejemplo,
de ArangoDB o OrientDB.

ArangoDB permite guardar agregados
de tipo clave-valor, documentos

y también modelos en grafo.
Hemos dicho que las bases de datos NoSQL

presentan un esquema de datos flexible.
Esta característica se conoce en inglés

como schemaless, lo que significa
que la base de datos no tiene un esquema

de datos predefinido a diferencia
de una base de datos relacional.

En el caso de una base de datos relacional,
para definir el esquema de la base de datos,

en primer lugar, definimos las tablas
y, luego, insertamos los datos que tienen

que ir en cada una de estas tablas.
En el caso de una base de datos NoSQL

(esto es especialmente cierto

en todas las bases de datos NoSQL,
excepto las clave-valor),

la estructuración de los datos se puede definir
a la vez que insertamos los datos

y, además, el esquema
puede variar para instancias de datos

que pertenecen a una misma entidad.
En algunas ocasiones, por ejemplo, en el caso

de los modelos clave-valor,
el gestor de la base de datos no es consciente

del esquema de la base de datos. Si existe,
solamente es conocido por los programas

de aplicación que están accediendo
a la base de datos. Las estructuras de datos

que se utilizan en las bases de datos
NoSQL son bastante próximas

a las estructuras de datos que utilizan
los programas de aplicación,

los lenguajes de programación.

Esto permite reducir los problemas
de concordancia, impedance mismatch,

que en ocasiones nos encontramos
cuando accedemos a bases de datos relacionales

desde diferentes programas de aplicación.

Otra de las cuestiones es que las bases
de datos NoSQL pueden definir

objetos tan complejos como sea necesario;

en un mismo agregado se pueden englobar
diferentes objetos del mundo real

que están relacionados entre sí.

En definitiva,
las bases de datos NoSQL

basadas en modelos de agregación

están aplicando técnicas de desnormalización

de los datos,

a diferencia de las bases de datos relacionales.
Con esto lo que se intenta

es evitar la ejecución de operaciones
de join.

¿Cuáles son las principales ventajas
de trabajar con un esquema de datos flexible?

Básicamente tres: en primer lugar,
facilitar la realización de cambios

en el esquema de la base de datos;

en segundo lugar,

poder trabajar con datos heterogéneos
y, por último, ya ha salido en parte,

intentar mejorar la eficiencia
de las operaciones de consultas

sobre la base de datos evitando
operaciones de join, y esto

es especialmente relevante en el caso
de que la base de datos sea distribuida.

Otra de las características
es que no ofrecen SQL como lenguaje de acceso,

es más, no existe un lenguaje estándar
de acceso, sea SQL

o cualquier otro tipo de lenguaje.

Esto significa que las bases de datos NoSQL ofrecen,

en la mayoría de casos, su propio lenguaje de manipulación y consulta de los datos.

En otros casos, no existe tal lenguaje y el acceso se hace vía API, por ejemplo,

vía API REST. Al igual que las bases de datos relacionales, cada base de datos NoSQL

proporciona drivers de acceso a la base de datos para diferentes lenguajes

de programación.

Aquí tenéis algunos ejemplos de lenguajes de programación para los cuales existen drivers.

Finalmente, de forma frecuente y, en especial, las bases de datos NoSQL

basadas en modelos de agregación, proporcionan operaciones que permiten

la integración con sistemas de computación distribuida,

como sería el caso, por ejemplo, del framework MapReduce.

Si pasamos ya a lo que sería la distribución, las bases de datos

NoSQL

se basan en lo que se conoce
bajo el nombre de escalabilidad horizontal,

a diferencia de las bases de datos relacionales.

Esto significa que en las bases
de datos NoSQL, de nuevo,

las bases de datos basadas en modelos
de agregación (porque las bases de datos

orientadas a grafo,

en principio, están pensadas
para ser utilizadas de forma centralizada

y, por lo tanto, funcionan mejor
bajo escalabilidad vertical);

en las bases de datos NoSQL
basadas en modelos de agregación

el procesamiento y el almacenamiento
de los datos se hace de forma distribuida

a través de una red de ordenadores,
en general, de bajo coste.

Esto es una diferencia
respecto a las relacionales

que, por lo general, trabajan
con ordenadores de élite

que tienen una gran capacidad de procesamiento
y de almacenamiento. Esta escalabilidad horizontal,

además, nos permite
crecer a medida que se necesita,

en función de las necesidades
del sistema distribuido, de tal manera

que es posible añadir nuevos ordenadores,
es decir, nuevos nodos, a nuestra red

de ordenadores sin que por ello tengamos
que parar el sistema; el sistema

continúa estando online. En definitiva,
la adición de nuevos nodos al sistema distribuido

se puede hacer en caliente.

Respecto a la distribución de los datos,
las bases de datos NoSQL, de nuevo,

las basadas en modelos de agregación,
porque las basadas en modelos en grafo están pensadas

para ser centralizadas principalmente,
promueven la fragmentación horizontal

de los datos (en el contexto de NoSQL
esta característica se conoce

como sharding) y la replicación masiva
de los datos mediante la aplicación

de arquitecturas master/slave o peer-to-peer.
La fragmentación y posterior distribución

de los datos se puede realizar
bien aplicando técnicas de hash

o dispersión,

que es el modo de trabajo preferido
de las bases de datos NoSQL clave-valor

que, en esencia, desde un punto de vista

de implementación, no son más que una tabla
de dispersión distribuida,

bien en función del valor que toman
ciertos atributos.

Este es el caso de las bases de datos NoSQL
orientadas a columnas y las basadas en documentos.

En el caso de las bases de datos NoSQL
orientadas a columnas

también se permite aplicar
fragmentación vertical de los datos,

en función de las diferentes columnas o familias
de columnas que se hayan definido.

Las ventajas que se pretenden conseguir
con la distribución de los datos

son fundamentalmente cuatro: en primer lugar,
acercar los datos allá donde se necesitan,

característica que se conoce
en inglés como data locality;

incrementar la disponibilidad de los datos;

mejorar la eficiencia de las operaciones,
especialmente de las operaciones

de consulta y, por último,
incrementar el nivel de paralelismo del sistema.

Cuando acometemos el desarrollo
de un sistema distribuido

también tenemos que tomar en consideración
el teorema CAP. Este teorema, inicialmente

presentado en forma de conjetura
en el año 2000 por el profesor Brewer

en una conferencia invitada en un congreso
de computación distribuida,

fue probado finalmente dos años más tarde
por dos profesores del MIT

y trata sobre las propiedades
deseables que un sistema distribuido

debería cumplir. Estas propiedades
son la consistencia, la disponibilidad

y la tolerancia a particiones.
La propiedad de consistencia

trata de la existencia réplicas de unos mismos datos
e implica que los usuarios

del sistema tienen que poder recuperar
siempre los mismos valores

para unos mismos datos,

en concreto, los valores más recientemente confirmados

en un mismo instante de tiempo.

Por su parte, la propiedad de disponibilidad

implica que las peticiones de servicio enviadas por los usuarios a un nodo que está disponible

dentro de nuestro sistema distribuido

deben obtener una respuesta.

Finalmente, la propiedad de tolerancia a particiones

implica que el sistema debe proporcionar servicio a los usuarios, a pesar de que se puedan producir

situaciones de avería (un corte en las radiocomunicaciones, pérdidas de mensajes…)

que causen que el sistema quede particionado en diferentes

componentes.

Pues bien, lo que dice el teorema CAP es que es imposible acometer

el desarrollo de un sistema distribuido que simultáneamente cumpla las tres propiedades.

Desde la perspectiva de las bases de datos NoSQL, en ocasiones,

de forma un tanto confusa,

este teorema se acaba reformulando diciendo que hay que elegir

dos de las tres propiedades.

Las bases de datos NoSQL, como ya hemos dicho, están pensadas

para ser utilizadas en sistemas altamente distribuidos que necesitan

estar siempre disponibles, el sistema distribuido visto como un todo

siempre tiene que estar online. Por lo tanto, un requisito fundamental es que el sistema

pueda proveer servicio aunque se puedan producir particiones; por lo tanto,

siempre se desea garantizar la propiedad P del teorema CAP. En esencia, por lo tanto,

puesto que solamente se pueden cumplir dos de las tres propiedades,

hay que elegir, y tenemos que decidir si deseamos priorizar

la disponibilidad de los datos o la consistencia de los datos.

Si elegimos la disponibilidad y trabajamos con un sistema AP,

en esencia, el sistema distribuido
siempre estará disponible,

aunque temporalmente pueda mostrar datos
inconsistentes en presencia de particiones.

Ejemplos de bases de datos NoSQL
que trabajan de forma AP

serían Riak, DynamoDB o Cassandra.
Por otra parte, si lo que queremos

es tener siempre una visión consistente
de los datos y, en consecuencia, trabajamos

con un sistema CP, lo que sucederá
es que el sistema siempre contendrá una visión

consistente de los datos, aunque no esté
totalmente disponible en presencia

de particiones. Ejemplos de bases
de datos NoSQL que trabajan

bajo la forma CP serían MongoDB, HBase
o Redis. Destaca en este punto

que los ejemplos representan la modalidad
por defecto del producto,

de la implementación, de la base de datos
NoSQL. En ocasiones,

una misma base de datos NoSQL es capaz
de trabajar de forma AP

o de forma CP, pero siempre hay
una configuración por defecto o preferida.

Para acabar la parte de distribución

es importante mencionar

la gestión de transacciones.
Las transacciones en una base de datos

relacional se ajustan a un modelo
transaccional que se conoce

como modelo ACID. En una base de datos
relacional, las transacciones

incorporan un conjunto de sentencias SQL
que verifican estas propiedades ACID,

que son: atomicidad, consistencia,
aislamiento y definitividad.

La atomicidad representa el todo o nada
de la transacción.

La transacción es la unidad atómica
de ejecución, nunca queda a medias:

o se completa totalmente
o no se realiza en absoluto.

Por su parte, la consistencia
implica que la transacción

debe preservar la integridad de la base
de datos, es decir, debe respetar

todas las restricciones de integridad
que se hayan podido definir y, adicionalmente,

si estamos trabajando con una base de datos
distribuida, la consistencia

también significa

que los datos contenidos en réplicas,
si tenemos réplicas de unos mismos datos,

como muy tarde
en el momento de confirmación de la transacción

deben contener exactamente los mismos valores.

Por su parte, la propiedad de aislamiento
implica

que una transacción

no puede ver interferida su ejecución,
no puede recuperar resultados anómalos,

debido a que se esté ejecutando
concurrentemente con otras transacciones

que puedan estar accediendo
a la misma porción de la base de datos.

Finalmente, la definitividad
significa que los resultados

producidos por transacciones
que confirman sus resultados tienen que ser

definitivos en disco, en la base de datos,
no se pueden perder. Soportar un modelo

de transacciones ACID

en bases de datos que almacenan
grandes volúmenes de datos

que están distribuidas y con replicación es complejo
y puede causar serios problemas

de rendimiento;

es por ello por lo que frecuentemente
las bases de datos NoSQL afirman

que no trabajan
con transacciones.

Esto es discutible o inexacto,
si preferís.

En el caso de una base datos NoSQL,
cada operación lanzada contra la base de datos

en sí misma constituye una transacción,
por lo tanto, lo que no se admite

son transacciones que incorporen
un conjunto de sentencias.

Una transacción es una sentencia.

Adicionalmente, y en relación
con el teorema CAP, las bases de datos NoSQL

de tipo AP

trabajan con un modelo de transacciones

alternativo al modelo ACID,
y este modelo de transacciones

recibe el nombre de modelo BASE.
El acrónimo de BASE no tiene tanta gracia

como el modelo ACID;
básicamente intenta ser un juego de palabras,

lo que sería el significado
de estas palabras en inglés:

acid, ácido, contra base,
de la química. Son posicionamientos

contrapuestos de cara a la gestión
de transacciones. La noción de consistencia

en el caso del modelo BASE,
que es lo importante,

es una noción de consistencia diferente
a la del modelo ACID y se refiere

a la existencia de réplicas. Se trata
de una noción de consistencia que se denomina

consistencia final en el tiempo,
en inglés: eventual consistency.

La consistencia final en el tiempo,
cuando la aplicamos a réplicas de unos mismos datos,

significa

que el contenido de las réplicas,
en un instante de tiempo, puede ser diferente

de una réplica con unos mismos datos,
pero se espera que con el paso del tiempo,

tras un cierto tiempo sin cambios
en esos datos, las réplicas de unos mismos

datos acaben convergiendo a unos mismos valores
de tal manera que vuelvan a ser consistentes.

Fijaos que la consistencia final
en el tiempo no coincide al 100 %

con la noción de consistencia del modelo ACID.
Puede parecer un poco abstracto

y un poco teórico hablar de un modelo
de transacciones BASE y hablar

de un teorema, el teorema CAP,
pero es posible que hayamos visto

las consecuencias del modelo BASE
y, por lo tanto, del teorema CAP,

en nuestro día a día.

A modo de ejemplo: cuando escribimos
un post en un blog que tenemos

y lo damos de alta y no vemos el post
que hemos escrito, pero un amigo nos dice

que le ha gustado mucho ese post; o cuando enviamos
un tweet y ese tweet nunca aparece

en nuestro timeline,
es como si se hubiera perdido; o cuando reservamos

una habitación de hotel mediante
una plataforma, a través de internet

o de una aplicación, y llegamos al hotel
y nos ofrecen una habitación superior

a la que hemos encargado por el mismo precio,

lo que estamos viendo son las consecuencias

del modelo BASE de gestión de transacciones.

Bien, pues, hasta aquí esta presentación
dedicada a características de las bases

de datos NoSQL. Esperamos que haya sido
de vuestro interés. Aquí os dejamos

algunas referencias por si deseáis
profundizar en el tema.

Y nada más por nuestra parte.
Que tengáis un buen día.