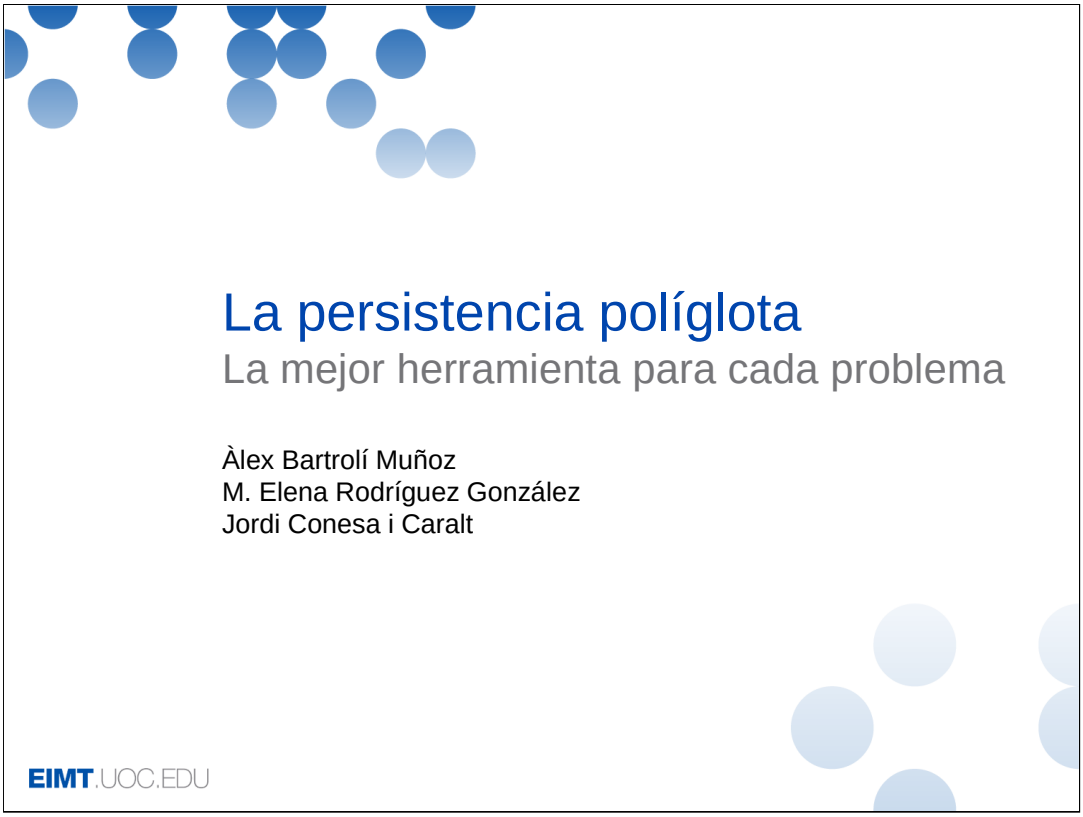


Bienvenidos a esta presentación sobre persistencia políglota. La persistencia políglota consiste en la utilización conjunta de diferentes bases de datos (quizá basadas en diferentes modelos de datos) en función del problema a tratar.

Persistencia políglota

- ¿Qué es la persistencia políglota?
- La estrategia *one size fits all*
- Escenarios donde no funciona un SGBD relacional
- Inconvenientes de las bases de datos NoSQL

EIMT.UOC.EDU

Esta presentación se divide en 4 secciones:

En la primera definiremos qué es la persistencia políglota.

En la segunda sección veremos la estrategia que se solía utilizar antes de la irrupción de la persistencia políglota, la estrategia *one size fits all.*

En la tercera veremos algunos escenarios donde los sistemas gestores de bases de datos relacionales no son capaces de dar una respuesta adecuada para el problema planteado.

Finalizaremos la presentación viendo algunos inconvenientes de las bases de datos NoSQL.

Persistencia políglota

- ¿Qué es la persistencia políglota?
- La estrategia *one size fits all*
- Escenarios donde no funciona un SGBD relacional
- Inconvenientes de las bases de datos NoSQL

EIMT.UOC.EDU

En primer lugar vamos a definir qué se entiende por persistencia políglota.

¿Qué es la persistencia políglota?

- La persistencia políglota consiste en el uso de diferentes tecnologías de almacenamiento para dar respuesta a las diferentes necesidades de almacenamiento.
- La persistencia políglota no consiste en substituir una tecnología de almacenamiento de datos por otra, sino en utilizar, dentro de un mismo proyecto o una misma empresa, la tecnología de almacenamiento de datos más apropiada para cada necesidad y tarea.

EIMT.UOC.EDU

La persistencia políglota consiste en el uso de diferentes tecnologías de almacenamiento de datos para dar respuesta a las diferentes necesidades de almacenamiento. No trata de substituir una tecnología por otra, sino utilizar, dentro de un mismo proyecto o una misma empresa, la tecnología de almacenamiento de datos más apropiada para cada necesidad y tarea.

Con persistencia políglota definimos la utilización de diferentes tecnologías de bases de datos (relacionales, clave-valor, orientadas a grafos, etc.) dentro de un mismo proyecto, utilizando la tecnología que mejor se adapte a cada necesidad y permitiendo la comunicación entre ellas y su acceso por una aplicación externa.

Persistencia Políglota

- ¿Qué es la persistencia políglota?
- La estrategia *one size fits all*
- Escenarios donde no funciona un SGBD relacional
- Inconvenientes de las bases de datos NoSQL

EIMT.UOC.EDU

Ahora que ya sabemos en que consiste la persistencia políglota, vamos a ver la estrategia *one size fits all*, que ha sido utilizada frecuentemente por los vendedores de soluciones de almacenamiento de datos hasta principios del siglo XXI.

¿Qué es la estrategia *one size fits all?*

- La estrategia *one size fits all* consiste en utilizar un único producto de almacenamiento de datos (SGBD relacionales) como solución única y suficiente para cualquier necesidad de almacenamiento de una empresa, en vez de utilizar soluciones adaptadas a cada necesidad.

EIMT.UOC.EDU

La estrategia *one size fits all* consiste en utilizar un único producto de almacenamiento de datos (normalmente han sido sistemas gestores de bases de datos relacionales) como solución única y suficiente para cualquier necesidad de almacenamiento de una empresa, en vez de utilizar soluciones adaptadas a cada necesidad.

Esta estrategia se llevó a cabo para simplificar el trabajo de los diferentes agentes que intervenían en la compra-venta de la solución de almacenamiento. Las empresas que contrataban un sistema de gestión de bases de datos relacional necesitaban formar a sus trabajadores sobre un único producto, y las empresas que lo vendían se evitaban problemas de costes, compatibilidad, ventas y *marketing* que detallaremos a continuación.

¿Qué pretende evitar la estrategia *one size fits all*?

- **Problemas de costes** derivados del mantenimiento de diferentes productos
- **Problemas de compatibilidad**: las aplicaciones deberían funcionar en los diferentes productos de almacenamiento de datos disponibles.
- **Problemas de ventas**: los comerciales deberían saber qué producto vender a cada cliente.
- **Problemas de *marketing***: diferentes productos se deben posicionar en los mercados apropiados.

EIMT.UOC.EDU

Con la estrategia *one size fits all* se pretendía evitar:

En primer lugar, los problemas de costes derivados del mantenimiento de diferentes productos. Al mantener un solo producto que diera cabida a las necesidades de las empresas, los costes de creación y mantenimiento eran más reducidos que si hubieran creado varios productos adaptados a cada empresa y/o cada necesidad.

En segundo lugar, los problemas de compatibilidad tendrían lugar porque las aplicaciones tendrían que interactuar con todos los productos de almacenamiento adquiridos. Desde la perspectiva de la empresa vendedora, sus productos de almacenamiento deberían garantizar dicha compatibilidad. Al ofrecer una única solución, se simplificarían los problemas.

En tercer lugar, tendríamos los problemas de ventas, que estarían ligados al hecho de que los comerciales deberían saber en todo momento qué producto concreto sería necesario vender a cada cliente. Ofreciendo un único producto, los encargados de ventas, sólo tendrían que mostrar al cliente que el producto daría solución a todas sus necesidades de almacenamiento.

Finalmente, en cuarto lugar, se simplificarían los problemas de *marketing* a la hora de posicionar diferentes productos en los mercados apropiados. Ofreciendo un único producto, se eliminaría la necesidad de segmentar el mercado ofreciendo en cada uno de ellos el producto que mejor se adaptase.

¿Es necesario abandonar la estrategia *one size fits all*?

- La estrategia *one size fits all* se ha utilizado hasta el inicio del siglo XXI. Por ejemplo, con la explosión de Internet y de los teléfonos inteligentes, las bases de datos relacionales no siempre pueden dar respuesta a proyectos que requieren el almacenamiento de un gran volumen de datos (posiblemente distribuidos) y su tratamiento en tiempo real.

EIMT.UOC.EDU

La estrategia *one size fits all* se ha utilizado hasta el inicio del siglo XXI. Con la explosión de Internet y de los teléfonos inteligentes, entre otros, las bases de datos relacionales no siempre pueden dar respuesta a proyectos que requieren el almacenamiento de un gran volumen de datos (por lo general distribuidos) y su tratamiento en tiempo real.

Estas nuevas tecnologías necesitan almacenar diferentes tipologías de datos para los que los sistemas gestores de bases de datos relacionales no son necesariamente óptimos.

A continuación veremos algunos aspectos por los que los sistemas gestores de bases de datos relacionales no pueden solucionar todas las necesidades de almacenamiento para alguno de los entornos de aplicación que las nuevas tecnologías han propiciado.

La *estrategia one size fits all* (y en consecuencia los sistemas gestores de bases de datos relacionales) no pueden dar respuesta a todas las necesidades de almacenamiento por diversas razones.

En primer lugar, los sistemas gestores de bases de datos relacionales están pensados para guardar datos estructurados, pero las nuevas necesidades de almacenamiento son más diversas e incluyen:

- Datos semi-estructurados
- Datos sin estructura
- Flujos de datos
- Datos provenientes de sensores
- Información incierta
- Información en forma de grafos
- Estructuras de datos complejas

En cambio, las bases de datos NoSQL están diseñadas para almacenar datos sin estructura y otros tipos de objetos complejos como documentos, *feeds* de datos y grafos, cubriendo las expectativas de representación de estos tipos de datos de forma más eficiente que un sistema gestor de bases de datos relacional.

¿Por qué no siempre funciona la estrategia *one size fits all*?

- La necesidad de almacenamiento de datos de forma distribuida
- Buena parte de las aplicaciones actuales necesitan realizar operaciones simples sobre modelos de datos complejos.

EIMT.UOC.EDU

Pero también existen otros motivos, como serían:

La necesidad de almacenamiento de datos de forma distribuida: los sistemas gestores de bases de datos relacionales escalan mejor verticalmente que horizontalmente.

Recordemos que la escalabilidad vertical consiste en aumentar la capacidad de cálculo y de almacenamiento (ya sea en términos de memoria principal o del dispositivo de almacenamiento externo) en un mismo ordenador.

Por su parte, la escalabilidad horizontal consiste en aumentar la capacidad de cálculo y de almacenamiento requeridos en base a añadir nuevos ordenadores que estén comunicados entre ellos.

Además, y relacionado con este punto, los sistemas gestores de bases de datos relacionales priman la consistencia sobre la disponibilidad de los datos y ofrecen una visión integrada de los datos (como si éstos estuvieran almacenados en un único lugar). Esto puede provocar que los sistemas gestores de bases de datos relacionales no den un tiempo de respuesta lo suficientemente rápido.

En cambio, las bases de datos NoSQL escalan mejor horizontalmente. Esto causa que se puedan adaptar mejor al almacenamiento de grandes volúmenes de datos que se encuentran distribuidos y replicados.

Otro motivo respecto a porque la estrategia *one size fits all* no siempre funciona tiene que ver con el hecho que las aplicaciones actuales, en general, necesitan realizar operaciones simples sobre modelos de datos complejos. Las bases de datos NoSQL tratan de responder a estas necesidades mediante:

- Implementaciones nuevas y especializadas,
- Operaciones simples y
- Modelos de datos que comparten la idea de ser flexibles

¿Por qué no siempre funciona la estrategia *one size fits all*?

- Limitaciones de SQL para realizar ciertos tipo de consultas
- La dificultad en la modificación del esquema de una base de datos relacional
- Las propiedades ACID no son siempre necesarias y pueden ralentizar el sistema.

EIMT.UOC.EDU

Finalmente, otros motivos de por qué la estrategia *one size fits all* no siempre funciona serían:

Las limitaciones de SQL para realizar ciertos tipo de consultas. Las bases de datos relacionales requieren el uso de operaciones de *join*. Estas operaciones son poco apropiadas para objetos complejos o para datos sin ninguna estructura.

La dificultad en la modificación del esquema de una base de datos relacional. El esquema de una base de datos relacional está fuertemente definido, lo que dificulta su alteración en un sistema en funcionamiento. Incluso los pequeños cambios se tienen que tratar con mucho cuidado, y pueden requerir un tiempo de inactividad o una reducción de los servicios ofrecidos. Las restricciones en las bases de datos NoSQL, en relación a este punto, suelen ser menores e incluso inexistentes.

Para acabar las propiedades ACID no son siempre necesarias y pueden ralentizar el sistema**.** Las propiedades ACID son indispensables en algunas aplicaciones (ventas, sector bancario y financiero) donde prima la consistencia a la velocidad. En cambio, el cumplimiento de las propiedades ACID puede no ser importante para un motor de búsqueda que puede devolver resultados diferentes a dos usuarios simultáneamente, o en sitios web como Amazon, cuando se devuelven diferentes *reviews* de un mismo producto a dos usuarios. En este tipo de aplicaciones, el rendimiento y la disponibilidad priman sobre la consistencia de los resultados.

Persistencia políglota

- ¿Qué es la persistencia políglota?
- La estrategia *one size fits all*
- Escenarios donde no funciona un SGBD relacional
- Inconvenientes de las bases de datos NoSQL

EIMT.UOC.EDU

Una vez visto por qué no siempre funciona la estrategia *one size fits all*, vamos a ver algunos ejemplos concretos de escenarios donde un sistema gestor de bases de datos relacional no funciona de forma suficientemente eficiente.

Escenarios donde no funciona un SGBD relacional

- *Data warehouse*
- Flujos de datos financieros en tiempo real
- Redes sociales y motores de búsqueda
- Sistemas de recomendación de películas
- Plataformas de juegos en línea
- Geolocalización móvil y sensores

EIMT.UOC.EDU

El advenimiento de Internet, los dispositivos móviles y los avances tecnológicos en algunas ramas científicas han permitido el desarrollo de aplicaciones y herramientas con necesidades de almacenamiento y de escalabilidad de datos diferentes a las ofrecidas por los sistemas gestores de bases de datos relacionales. Algunos ejemplos serían:

Data Warehouse: el perfil de los usuarios de los *data warehouse* es muy diferente al perfil de los usuarios de los sistemas gestores de bases de datos relacionales, así como el tipo operaciones que realizan.

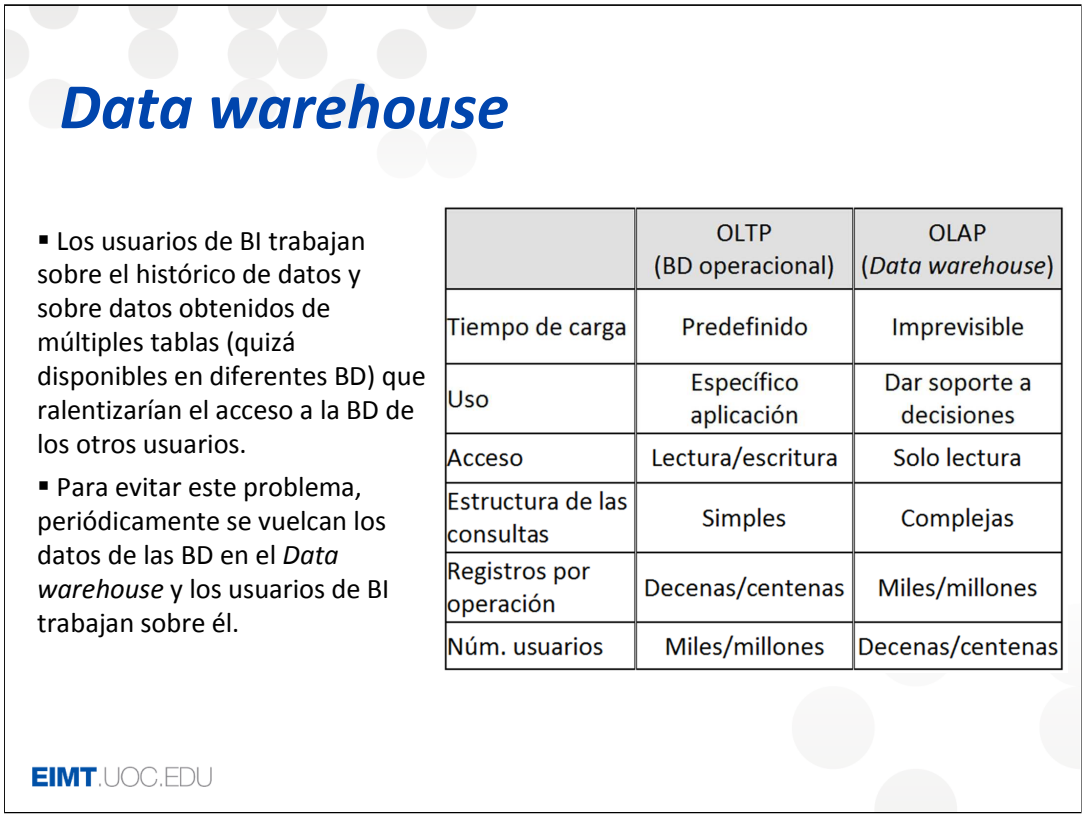
Flujos de datos financieros en tiempo real: los datos para la toma de decisiones en mercados financieros deben estar disponibles en tiempo real y a menudo no es necesario almacenar todo el flujo de datos para tomar las decisiones porque el tiempo necesario para almacenarlo puede suponer un retraso innecesario, que puede resultar crítico en la toma de decisiones.

Redes sociales, motores de búsqueda, sistemas de recomendación de películas y plataformas de juegos *online.* Estos entornos de aplicación necesitan almacenar flujos provenientes de diferentes fuentes de forma concurrente en un mismo archivo. Las escrituras acostumbran a ser *append-only* (es decir, de inserción de datos) y las lecturas secuenciales. Debido a la inmensa cantidad de datos generados, se tienen que utilizar sistemas de almacenamiento económicos que pueden fallar a menudo, por lo que los datos tienen que estar fuertemente replicados. Por estos motivos, una base de datos NoSQL puede ofrecer mejores resultados que una base de datos relacional.

Geolocalización móvil y sensores: los datos provenientes de sensores y de dispositivos móviles almacenan las coordenadas geográficas (latitud y longitud) en tiempo real de personas, vehículos u objetos. Algunos ejemplos de consultas que se deben realizar serían:

- Muestra el recorrido realizado por la persona o el vehículo X en las últimas dos horas.
- Cuántos camiones de una flota se encuentran en una determinada zona.
- Cuáles son las entradas a una ciudad que presentan menor congestión en estos momentos.

Este tipo de consultas no siguen la estructura clásica de SQL, y por lo tanto, un sistema de gestión de bases de datos relacional no es la mejor opción para derivar esta información en el tiempo requerido.



A pesar de que tanto en una base de datos relacional como en un *data warehouse* se utiliza SQL como lenguaje de consulta, el tipo de operaciones que se realizan son bastante diferentes.

Los usuarios de BI trabajan sobre el histórico de datos y sobre datos obtenidos de múltiples tablas que pueden provenir de diferentes bases de datos para realizar el análisis de datos en el *data warehouse*, mientras que el resto de usuarios realizan operaciones transaccionales sobre la base de datos (que estará gestionada por el sistema gestor de la base de datos). Las bases de datos relacionales que nutren al *data warehouse* se conocen también bajo la denominación de bases de datos operacionales.

Si los usuarios del *data warehouse* trabajasen sobre la misma base de datos a la que acceden el resto de usuarios, su acceso se vería ralentizado por las consultas de los usuarios de BI. Para evitar este problema, periódicamente se vuelcan los datos necesarios de las bases de datos en el *data warehouse*.

Existen otras diferencias importantes entre un *data warehouse* y una base de datos operacional.

Mientras que el tiempo de carga de los datos en una base de datos está predefinido y es (en principio) de corta duración, en un *data warehouse* no podemos preveer el tiempo de carga de todos los datos necesarios. Este tiempo de carga, además, puede ser bastante elevado.

Una base de datos se diseña para cubrir las necesidades específicas de cada empresa (por ejemplo, para dar de alta nuevos clientes, introducir una venta realizada, consultar la disponibilidad de un trabajador, etc.). En cambio, un *data warehouse* se utiliza para dar soporte a la toma de decisiones, y esto implica que pueda ser necesario ejecutar consultas imprevistas (o sea, consultas ad doc que no se han especificado a priori).

El acceso en una base de datos es tanto de lectura como de escritura, mientras que en los *data warehouse*, una vez realizada la carga de datos, sólo se realizan operaciones de lectura.

De manera similar, las consultas en una base de datos son simples, mientras que en un *data warehouse* se realizan consultas complejas que involucran a muchas más tablas para obtener información histórica en base a diferentes parámetros.

El número de registros sobre los que se actúa en una base de datos acostumbra a ser de decenas o centenas, mientras que en un *data warehouse* se trabaja con miles o millones de registros.

El número de usuarios que trabajan sobre una base de datos es mucho mayor que en un *data warehouse*, debido a que sobre ella acceden usuarios de múltiples departamentos, mientras que a un *data warehouse* sólo acceden los usuarios de BI.

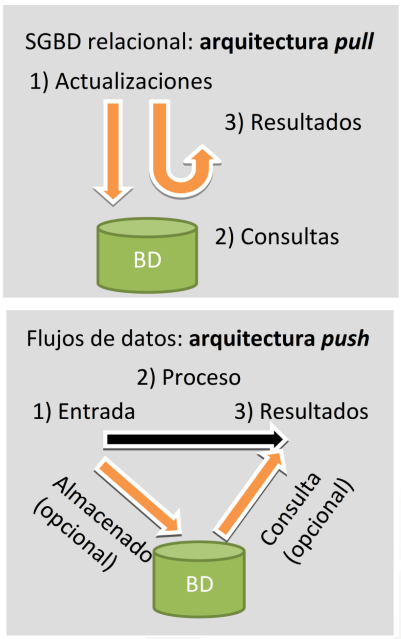
Flujos de datos

- Los SGBD relacionales almacenan los datos y posteriormente permiten realizar las consultas

⇒ a**rquitectura *pull***

- Con flujos de datos en tiempo real, los datos se deben transmitir directamente a destino sin necesidad de ser almacenados previamente ⇒ **arquitectura *push***
- El almacenamiento no es siempre necesario. Se pueden almacenar los datos en memoria y hacer una escritura conjunta más rápida por intervalo de tiempo o por tramos de datos.
- La importancia recae en la calidad de la información extraída y en el cálculo de valores agregados, no en el almacenamiento de una copia exacta.
- Los datos son "infinitos" y la información se calcula por tramos de datos o por intervalos de tiempo.
- La eficiencia y el rendimiento priman sobre la consistencia ⇒ **no es necesario cumplir propiedades ACID**
- Los mecanismos de recuperación también funcionan diferente.

EIMT.UOC.EDU



Otro escenario donde un sistema gestor de bases relacional no da buenos resultados son los flujos de datos. Esto se debe a que los sistemas gestores de bases de datos relacionales utilizan una arquitectura de almacenamiento *pull*. En dicha arquitectura, los datos se deben almacenar en la base de datos antes de poder ser consultados.

Este esquema de almacenamiento es ineficiente en algunos escenarios, por ejemplo en mercados bursátiles, donde un mínimo retraso puede suponer el éxito o fracaso en la toma de una decisión.

Para sistemas que trabajan con flujos de datos, es habitual trabajar con una arquitectura de almacenamiento *push***.** En esta arquitectura, los datos son transmitidos desde el origen al destino sin la necesidad de ser almacenados previamente en la base de datos. Los datos pueden almacenarse en memoria y, con el objetivo de realizar una escritura más rápida, se vuelcan conjuntamente en la base de datos por intervalos de tiempo o tramos de datos.

En algunos casos, no es necesario almacenar todo el flujo de datos. En su lugar se almacena la información extraída de los datos y los cálculos de valores agregados. Para realizar estos cálculos debemos tener en cuenta que los flujos de datos pueden ser infinitos, por lo que la información se obtendrá por intervalos de tiempo o tramos de datos.

Para finalizar, cabe recordar que en los sistemas que trabajan con flujos de datos, la eficiencia y el rendimiento priman sobre la consistencia, por lo que no es necesario cumplir con las propiedades ACID. De forma similar, los mecanismos de recuperación ante situaciones de fallo también funcionan de manera diferente a un sistema gestor de bases de datos relacional.

Escalabilidad

- Con el advenimiento de las nuevas tecnologías, las redes sociales y los dispositivos móviles, la gran cantidad datos generados promueve el uso de sistemas distribuidos y con réplicas de los datos para garantizar la disponibilidad de los datos y su almacenaje y procesamiento de forma sostenible.
- En estos escenarios es necesario disponer de sistemas de almacenamiento que permitan un crecimiento de acuerdo a las necesidades, sin que ello signifique dejar de proporcionar servicio o dar un servicio de peor calidad.

EIMT.UOC.EDU

En los escenarios donde la escalabilidad es un factor determinante, los sistemas gestores de bases de datos relacionales pueden no ser la mejor opción.

En algunos casos, la gran cantidad de datos a procesar puede requerir almacenar los datos de forma distribuida. El almacenamiento distribuido puede tener ventajas tanto económicas como en disponibilidad de los datos.

Desde el punto de vista económico, y cuando los datos son masivos, puede ser más rentable almacenar los datos en una red de ordenadores de poca capacidad que en un ordenador de élite con gran capacidad de proceso y almacenamiento.

Desde un punto de vista de disponibilidad, tener los datos en un solo ordenador puede entrañar riesgos. Por otro lado, al estar los datos distribuidos en diferentes ordenadores, se incrementa el riesgo de que se produzcan situaciones de avería. Ello motiva la replicación de los datos.

La escalabilidad es importante planteársela no sólo teniendo en cuenta los datos iniciales a almacenar, sino también los datos que se esperan en el futuro. Es decir, en escenarios donde se deba permitir un crecimiento dinámico de acuerdo a diversas necesidades poco previsibles, puede ser aconsejable escoger una arquitectura de bases de datos que permita escalar de forma dinámica para ofrecer el mismo servicio sin perder calidad.

Tanto los sistemas relacionales como los NoSQL permiten distribución, réplicas y escalabilidad. No obstante, sus aproximaciones son diferentes: las bases de datos relacionales priorizan la consistencia a la disponibilidad. En cambio, las tecnologías NoSQL priorizan la disponibilidad a la consistencia, por lo que escalan mejor horizontalmente que los sistemas gestores de bases de datos relacionales. Por lo tanto, los sistemas NoSQL pueden ser más aptos en aquellos proyectos donde la escalabilidad es un factor determinante pero a costa de sacrificar, en cierto grado, la consistencia de los datos.

Persistencia políglota

- ¿Qué es la persistencia políglota?
- La estrategia *one size fits all*
- Escenarios donde no funciona un SGBD relacional
- Inconvenientes de las bases de datos NoSQL

EIMT.UOC.EDU

Ahora que ya hemos visto la estrategia *one size fits all* y algunos escenarios donde no funciona un sistema gestor de bases de datos relacional, pasamos a ver algunos inconvenientes de las bases de datos NoSQL.

Inconvenientes de las BD NoSQL

- **Madurez**: los SGBD relacionales son sistemas estables con múltiples funcionalidades mientras que las BD NoSQL tienen funcionalidades sin implementar.
- **Especialistas**: hay millones de programadores en el mundo familiarizados con los SGBD relacionales, pero no es tan fácil encontrar especialistas en BD NoSQL.
- **Falta de estándares**: no existe un lenguaje de consulta unificado como SQL.
- **Administración**: las BD NoSQL pueden ser más complejas de instalar y mantener que los SGBD relacionales y la curva de aprendizaje puede ser lenta.

EIMT.UOC.EDU

Las bases de datos NoSQL no tienen el mismo estado de madurez que los sistemas de gestión de bases de datos relacionales. Estos últimos tienen múltiples funcionalidades implementadas y ampliamente verificadas desde hace tiempo, mientras que las bases de datos NoSQL tienen todavía funcionalidades sin implementar.

Debido a la "juventud" de las bases de datos NoSQL, es mucho más fácil encontrar profesionales familiarizados en bases de datos relacionales que en NoSQL.

Las bases de datos NoSQL están faltas de estándares y no tienen un lenguaje de consulta unificado como es el caso de SQL.

La administración de las bases de datos NoSQL puede ser más compleja que las de los sistemas de gestión de bases de datos relacionales. La instalación, mantenimiento y la curva de aprendizaje puede resultar ser lenta.

Inconvenientes de las BD NoSQL

- **Soporte**: las empresas especializadas en NoSQL no tienen la envergadura de las empresas que trabajan con SGBD relacionales (Oracle, Microsoft o IBM) y deben garantizar una asistencia técnica rápida y eficiente.
- **Falta de un *benchmark*** definido (todos son los mejores)
- El **esquema de una BD NoSQL puede ser complejo** y no siempre está definido.

EIMT.UOC.EDU

Se debe garantizar un buen servicio de soporte. Las empresas especializadas en NoSQL no tienen la envergadura de las empresas que trabajan con sistemas gestores de bases de datos relacionales (Oracle, Microsoft o IBM) y deben garantizar una asistencia técnica rápida y eficiente ante fallos del sistema.

Las bases de datos NoSQL carecen de un *benchmark* claro y bien definido. Todas las soluciones NoSQL se presentan como la mejor opción. Por ello se echa en falta un *benchmark* que permita comparar los diferentes productos disponibles.

El esquema de una base de datos NoSQL puede ser complejo y, en general, no está definido. Un sistema gestor de bases de datos relacional guarda información sobre el esquema de la base de datos. En el caso de una base de datos NoSQL el esquema está, casi siempre, escondido en las aplicaciones. Este hecho puede dificultar su modificación y la gestión de otros aspectos, por ejemplo, los ligados a la seguridad (derechos de acceso de los usuarios a la base de datos).

Persistencia políglota

- ¿Qué es la persistencia políglota?
- La estrategia *one size fits all*
- Escenarios donde no funciona un SGBD relacional
- Inconvenientes de las bases de datos NoSQL

EIMT.UOC.EDU

Hasta aquí esta presentación dedicada a la persistencia políglota que propugna elegir la mejor solución de almacenamiento, en contraposición a la estrategia *one size fits all* defendida por los fabricantes de sistemas gestores de bases de datos relacionales. Para argumentar esta idea, hemos examinado algunos escenarios donde las bases de datos relacionales no constituyen la mejor solución. Finalmente, también hemos revisado las principales críticas que se realizan sobre las bases de datos NoSQL.

Referencias

P. Atzeni, C.S. Jensen, G. Orsin & S.Ram (2013). "The relational model is dead, SQL is dead, and I don't feel so good myself", *SIGMOD Record* 42(2), pp. 64-68. (http://www.sigmod.org/publications/sigmod-record/1306/pdfs/11.reports.atzeni.pdf)

R. Cattell (2010). Scalable SQL and NoSQL Data Stores. (http://cattell.net/datastores/Datastores.pdf)

L. Liu & M.T. Özsu (Eds.) (2009). *Encyclopedia of Database Systems*. Springer.

P.J. Sadalage & M. Fowler. (2013). *NoSQL Distilled. A brief Guide to the Emerging World of Polyglot Persistence*, Pearson Education.

M. Stonebraker & U. çetintemel (2005). "One Size fits all: An Idea whose time has come and gone", *Proceedings of the International Conference on Data Engineering*, pp 2-11. (http://cs.brown.edu/people/ugur/fits_all.pdf).

Esperamos que hayáis disfrutado y aprendido con este vídeo. A continuación encontraréis algunas referencias que os permitirán profundizar más en los aspectos tratados.

Que tengáis un buen día.