



1. INTRODUCCIÓN A NOSQL

1. INTRODUCCIÓN A NOSQL.....	1
1.1. El surgimiento del término NoSQL.....	3
1.2. ¿Qué es NoSQL ?.....	3
1.3. ¿Qué no es NoSQL ?.....	4
1.4. Tipos de bases de datos no relacionales.....	5
1.5. Nuevos requerimientos de negocio.....	5
1.5.1. Volumen.....	6
1.5.2. Variedad.....	6
Datos Estructurados.....	6
Datos Semi-estructurados.....	6
Datos No estructurados.....	6
1.5.3. Velocidad y agilidad.....	7
1.5.4. Valor.....	8
1.6. Casos de estudio NoSQL.....	8
1.6.1. LiveJournal's Memcache.....	8
1.6.2. Google MapReduce.....	9
1.6.3. Google BigTable.....	9
1.6.4. Amazon Dynamo.....	10
1.6.5. MarkLogic.....	10
Tarea.....	10
1.7. Comparación RDBMS Vs Sistemas NoSQL.....	10
1.8. Conceptos Básicos.....	14
1.8.1. Clusters.....	14
1.8.2. Consistent Hashing.....	15
1.8.3. ACID Vs BASE.....	16
1.8.3.1. Transacción.....	16
1.8.3.2. Propiedades ACID de las transacciones en RDBMS.....	16
Atomicidad.....	17
Consistencia.....	17
1.8.3.3. BASE en sistemas NoSQL.....	17
Basic Availability.....	18
1.8.3.4. Database sharding.....	19

1.8.3.5. El teorema de CAP.....	20
1.8.4. Selección de Sistema de bases de datos.....	21
1.8.4.1. Consistencia y tolerancia a la partición.....	22
1.8.4.2. Disponibilidad y tolerancia a la partición.....	22
1.8.4.3. Sistemas NewSQL.....	22
Tarea.....	23

1.1. El surgimiento del término NoSQL

- El término NoSQL apareció por primera vez al final de los 90s como un nombre propuesto para una base de datos relacional: *Strozzi NoSQL*. Su creador *Carlo Strozzi*.
 - Esta base de datos almacena sus tablas en archivos ASCII, cada tupla representa un renglón del archivo con columnas separadas por tabulaciones.
 - El nombre se debe a que no hace uso del lenguaje SQL, utiliza shell scripts combinados con el uso de UNIX pipelines.
- Posteriormente el término NoSQL fue propuesto para darle nombre a una conferencia organizada por *Johan Oskarsson* que hablaría de las nuevas bases de datos de ese momento: BigTable (Google) y Dynamo (Amazon).
 - El nombre fue elegido principalmente por ser atractivo, fácil para hacer un Twitter hashtag, aunque realmente no describía a estos sistemas, y sin pensar que este término sería empleado para nombrar a toda una *tendencia tecnológica*.
 - El nombre adecuado de esta conferencia sería: *Bases de datos open source, distribuidas y no relacionales*.



1.2. ¿Qué es NoSQL ?

- Definir NoSQL representa en sí un reto. El término en realidad no describe los principales conceptos del movimiento NoSQL, sin embargo es utilizado ampliamente principalmente para diferenciar a un producto o solución de los sistemas de bases de datos tradicionales (RDBMS).
- La definición que podría considerarse adecuada con base a los objetivos del curso es:

NoSQL representa a un conjunto de conceptos que permite el procesamiento rápido y eficiente de grandes conjuntos de datos con un enfoque constante en desempeño, confiabilidad y agilidad.

La definición anterior es un tanto genérica. Notar que los RDBMs pueden cumplir con esta definición, lo cual no representa problema alguno. Lo importante es tener en cuenta las siguientes características o conceptos.

- *Va más allá de registros en tablas.* Sistemas NoSQL obtienen datos almacenados en diversos formatos:
 - key-value
 - graph
 - column-family
 - document stores
 - Incluso, registros en tablas.

- *Libre de operaciones Join.* Es posible recuperar datos empleando interfaces de acceso sencillas que evitan el uso de operaciones join.
- *Libre de esquemas (schema-free).* Permite prácticamente poner datos donde sea sin contar con una estructura predefinida así como su consulta sin tener que crear un *Modelo Entidad Relación* (MER) ni un *Modelo Relacional*.
- *Funciona con múltiples procesadores.* Se beneficia del uso de hardware de multiprocesamiento manteniendo un alto desempeño.
- *Funciona en múltiples máquinas simples y de bajo costo* cuyo único recurso compartido es una red de datos (clusters).
 - Las Bases de datos relacionales también corren en clusters, por ejemplo, Oracle RAC, sin embargo, esta solución se basa en la existencia de una BD común (storage compartido) que puede representar el famoso *single point of failure*.
 - Es posible que una BD relacional corra en máquinas completamente independientes (Bases de datos Distribuidas - Sharding). Para este caso, la aplicación tiene que controlar la naturaleza de distribución de datos, además de perder restricciones de referencia, transacciones locales, y algunas propiedades ACID.
- *Permite una escalabilidad lineal.* Si se agrega más capacidad de cómputo se obtiene un incremento consistente en desempeño.
- *Innovador.* Ofrece opciones para realizar un almacenamiento, recuperación y manipulación simple de los datos, incluyendo las soluciones con SQL. De aquí que el término NoSQL se refiere a “**Not Only SQL**”.

1.3. ¿Qué no es NoSQL ?

- *NoSQL no significa la ausencia de SQL.* SQL también puede ser empleado en bases de datos NoSQL
- *No solamente es Open Source.* La mayoría de las soluciones NoSQL son open-source. Existen soluciones comerciales.
- *NoSQL no solo se limita a Big Data.* Volumen y Velocidad no son los únicos focos de atención para las BDs NoSQL. También se consideran variabilidad y agilidad.
- *No solo se limita al Cloud.* Los sistemas NoSQL pueden correr o funcionar en los centros de cómputo de los clientes.

- No solo se limita al uso de la RAM o de los SSDs. Sistemas NoSQL también pueden correr en discos tradicionales.
- No representa a una suite elite de productos.

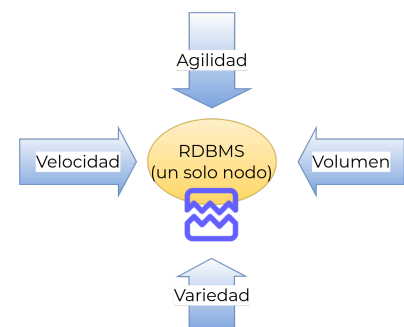
1.4. Tipos de bases de datos no relacionales

La siguiente tabla muestra las principales categorías de bases de datos NoSQL así como algunos ejemplos de cada categoría a los cuales se les conoce como **Patrones de arquitectura NoSQL**. Cada uno de estos patrones será revisado durante el curso.

Tipo	Uso típico	Ejemplos
<p><i>Key -Value store</i></p> <p>Emplea una llave para acceder a un valor</p>	<ul style="list-style-type: none"> • Almacenamiento multimedia • Sistemas de archivos basados en clave/valor • Caché de objetos 	<ul style="list-style-type: none"> • Berkeley DB • Memcache • Redis • Riak • DynamoDB
<p><i>Column family store</i></p> <p>La llave es una matriz que contiene registros y columnas</p>	<ul style="list-style-type: none"> • Rastreadores web • Problemas en Big Data 	<ul style="list-style-type: none"> • Apache HBase • Apache Cassandra • Hypertable • Apache Accumulo
<p><i>Graph store</i></p> <p>Para aplicaciones con una gran cantidad de relaciones.</p>	<ul style="list-style-type: none"> • Redes sociales • Detección de fraudes • Relaciones con volúmenes grandes de datos. 	<ul style="list-style-type: none"> • Neo4j • AllegroGraph • Bigdata (RDF data store) • InfiniteGraph
<p><i>Document store</i></p> <p>Almacena Jerarquías de estructuras de datos de forma directa</p>	<ul style="list-style-type: none"> • Alta variabilidad de los datos • Búsquedas por documento • Centros de investigación • Administración de contenido web 	<p>MongoDB</p> <p>CouchDB</p> <p>Couchbase</p> <p>MarkLogic</p> <p>eXist-db</p> <p>Berkeley DB XML</p>

1.5. Nuevos requerimientos de negocio

Principales requerimientos que obedecen a nuevas necesidades de empresas u organizaciones que motivaron el surgimiento de las soluciones NoSQL



- Volumen
- Velocidad
- Variabilidad
- Valor

1.5.1. Volumen

Las empresas comienzan a captar mayores cantidades de datos para realizar seguimientos detallados a sus procesos de negocio, clientes, productos, etc. Aparecen las redes sociales y la cantidad de datos aumenta exponencialmente.

Grandes cantidades de datos son generados hoy en día, en especial por sistemas, sensores, etc. Un simple avión puede generar 10 TB en 30 min. Imaginar la cantidad total de datos por día de una aerolínea.

Medidas usadas:

- Terabyte 10^{12} bytes
- Petabyte 10^{15} bytes
- Exabyte 10^{18} bytes
- Zettabyte 10^{21} bytes (1 billón de Tera bytes)
- Yottabyte 10^{24} bytes

1.5.2. Variedad

Tradicionalmente el universo de los datos recolectados por las empresas se limitaba a datos empleando las reglas y estructuras del modelo relacional (relación, tupla, columna, dato). Hoy en día, los datos relacionales representan sólo a una de las categorías de datos que se recolectan, es decir, el acervo de los datos de las empresas ahora está formado por datos provenientes de diversas fuentes, organizados de formas diversas: ***persistencia políglota***. Existen 3 principales categorías en la que los datos pueden ser organizados o estructurados:

Datos Estructurados

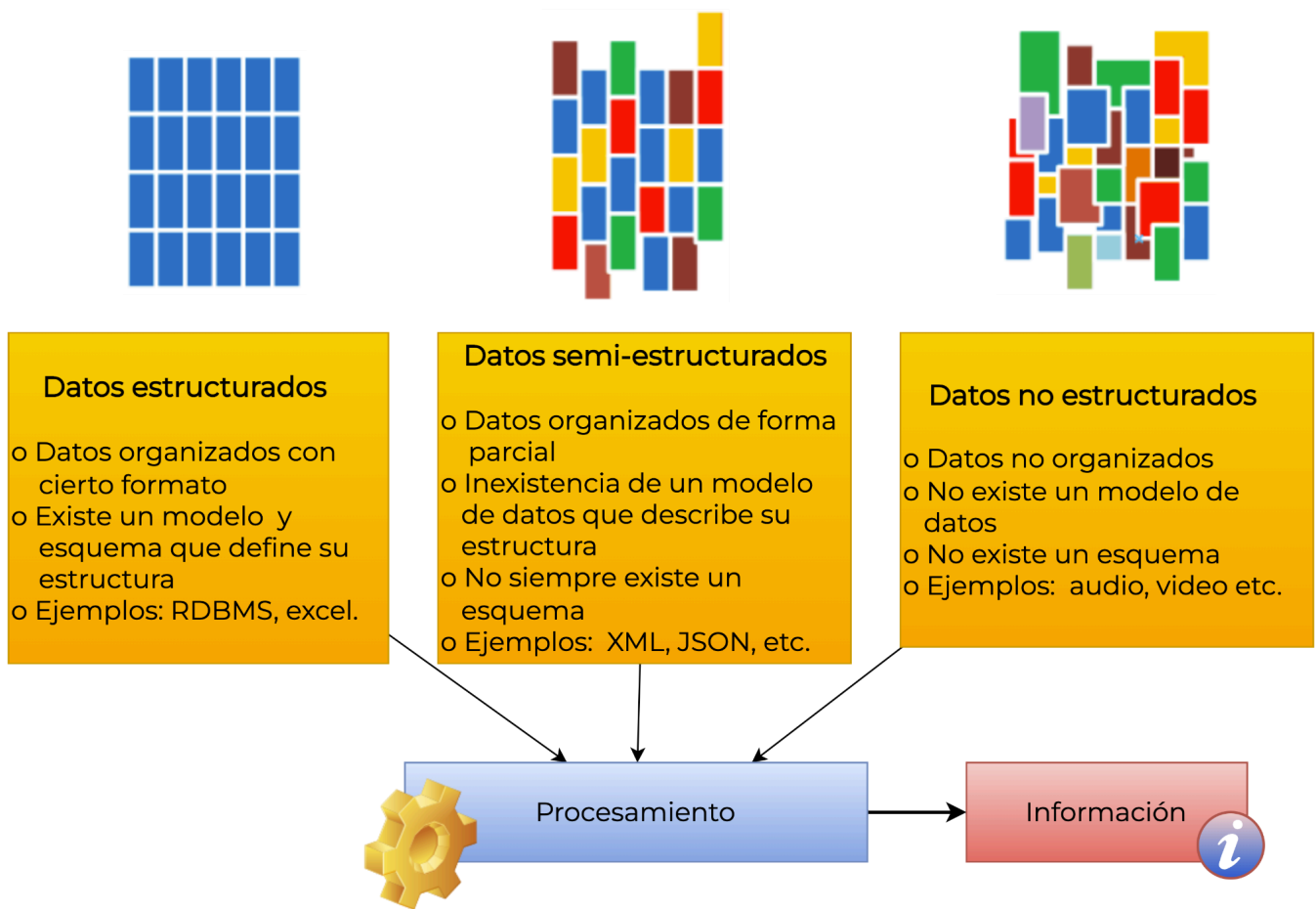
- Tienden a ser relativamente bien definidos a través de un «esquema» (modelo relacional).
- Datos tradicionales generados comúnmente por sistemas transaccionales

Datos Semi-estructurados

- Datos generados por máquinas o sensores: bitácoras, Call Detail Records (Datos relativos a llamadas telefónicas).
- Datos sociales: Redes sociales, microblogging (Twitter).
- Típicamente: documentos JSON, XML

Datos No estructurados


- Datos multimedia: imágenes, audio, bitácoras sin la existencia de un esquema.



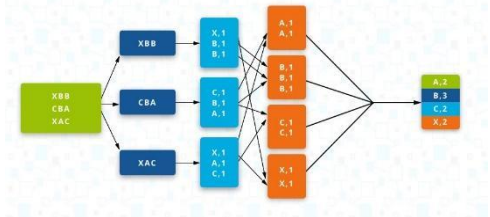


1.5.3. Velocidad y agilidad

El reto ahora es procesar estos 3 universos de datos. De la misma forma en la que se tiene la capacidad de captar grandes cantidades de datos con estructuras diversas, se deberá contar con la capacidad para procesar y analizar de forma ágil y en velocidades en algunas ocasiones ¡en tiempo real!

Los siguientes ejemplos de tecnologías han permitido a las empresas procesar cantidades masivas de datos en múltiples formatos a costos aceptables.

Tecnología	Ejemplos.
Incorporación de modelos y bases de datos NoSQL	Basados en documentos, basadas en clave llave valor, etc.
Frameworks para procesar cantidades masivas de datos distribuidos en clusters	Apache Hadoop, Spark 

Tecnología	Ejemplos.
Sistemas de archivos distribuidos que permitan almacenar cantidades masivas de datos	Hadoop Distributed File System (HDFS) 
Frameworks que permitan comunicar sistemas distribuidos a través de colas de mensajes	Apache Kafka 
Nuevas estrategias y técnicas de programación que permitan el procesamiento de cantidades masivas de datos de forma distribuida.	<p>MapReduce</p> 

1.5.4. Valor

¿Por qué interesa captar y procesar toda esta cantidad de datos?

- El conocimiento que se puede extraer de estas fuentes de datos es de vital importancia para lograr los objetivos de negocio de cualquier empresa: Más datos, mayor conocimiento, mayores beneficios.
- Las empresas deben actualizar su infraestructura tecnológica para poder generar valor sobre estas nuevas formas de hacer análisis.
- El análisis de datos basados en herramientas y sistemas OLAP tradicionales que operan solo con datos relacionales son muy útiles y exitosos los cuales deberán unir resultados con las nuevas estructuras.

1.6. Casos de estudio NoSQL

Los siguientes casos de estudio muestran la forma en la que se han resuelto requerimientos de negocio que diversas empresas han tenido que resolver para mantener o alcanzar sus objetivos ante esta nueva era de revolución tecnológica y competitiva.

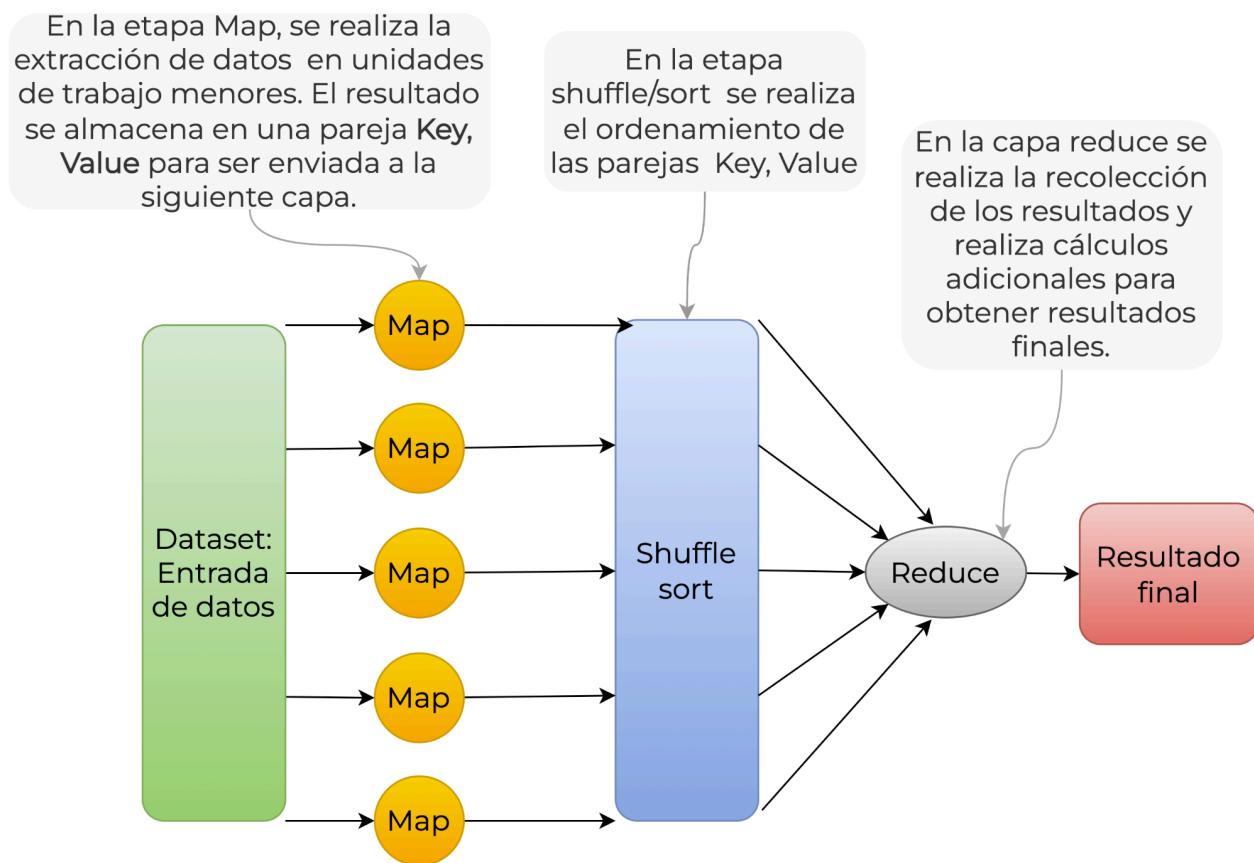
1.6.1. LiveJournal's Memcache

Se trata de un sistema web para realizar blogging. Su popularidad provocó un incremento importante en el número de visitas. La solución para atender esta demanda fue la incorporación de nuevos servidores web (nodos), cada uno con sus propios recursos (RAM, etc.).

- Para mejorar el desempeño, el sistema usa un caché de datos en memoria para almacenar los resultados de consultas SQL.

- El problema es que cada nodo tiene su propia copia, con prácticamente los mismos datos. Lo ideal sería que este caché fuera compartido para evitar retrabajo y copias de datos en caché duplicados.
- Se decidió utilizar un proceso de hashing en el que a cada consulta SQL se le asocia un valor hash. Este valor se envía a otros nodos para verificar si el server tiene este mismo valor en su caché. Por su longitud pequeña no representa un problema en cuanto a cantidad de datos a transmitir entre nodos. Esta solución se estandarizó incluyendo hasta el protocolo de comunicación a la que se le llamó **memcached protocol**.
- Esta herramienta hace uso de una BD en memoria llamada **in-memory-key-value store**.

1.6.2. Google MapReduce



Uno de los principales procesos que realiza Google para atender una búsqueda de forma eficiente, es la transformación de grandes volúmenes de datos obtenidos del Internet a un conjunto de *índices de búsqueda* haciendo uso de máquinas caseras, comunes a bajo costo.

- Esta transformación se realiza a través de 2 etapas empleando funciones **map** y **reduce**.
- Las funciones **map** y **reduce** se emplean para particionar grandes cantidades de datasets en unidades de trabajo de menor tamaño que pueden ser transformadas

de forma independiente y en paralelo. Esta solución es ampliamente escalable empleando múltiples máquinas.

1.6.3. Google BigTable

¡BigTable representa a una tabla con un billón de registros y un millón de columnas!

- La principal motivación de esta solución surge a partir de un requerimiento por parte de los rastreadores web que analizan contenido del Internet (html, imágenes, vídeos, audios). El dataset resultante es tan grande que no podría ser almacenado en una base de datos relacional por lo que Google construyó su propio sistema de almacenamiento empleando hardware de bajo costo.
- A esta solución se le nombró *Distributed storage system*. Esta solución trabaja con datos estructurados.

1.6.4. Amazon Dynamo

Requerimiento: ¡Aceptar órdenes de compra las 24 hrs del día, los 7 días a la semana!

- Este producto fue publicado en un NoSQL paper de Amazon llamado *Amazon's 2007 Dynamo: A Highly Available Key-Value store*.
- Como su nombre lo indica, se trata de un sistema altamente disponible, confiable, escalable, sin ninguna interrupción.
- La solución inicial de Amazon fue la adquisición de miles de licencias de manejadores RDBMS pero rápidamente se detectó la no viabilidad para atender futuros requerimientos.

1.6.5. MarkLogic

Base de datos que permite almacenar largas colecciones de documentos XML de forma distribuida.

- La solución se basa en 2 tipos de nodos:
 - Query nodes: Recibe y coordina las peticiones de ejecución de una consulta para que esta sea ejecutada en todos los nodos que contiene documentos XML indexados. La consulta (query) se envía a los nodos en lugar de extraer el documento para enviarlo a otro nodo para ser procesado.
 - Document nodes: Contiene los documentos XML, se encarga de ejecutar la consulta.



Tarea

Investigar otro caso de estudio (diferente a los mostrados anteriormente) para cada una de las categorías de sistemas NoSQL:

- Key Value
- Documents
- Graph

1.7. Comparación RDBMS Vs Sistemas NoSQL

RDBMS

Concepto	Ventajas - RDBMS
Concurrencia	Aplicaciones que soportan a una gran cantidad de usuarios pueden consultar e inclusive intentar modificar el mismo conjunto de datos de forma simultánea. Esta condición deberá tratarse adecuadamente. El manejo de programación concurrente es complicada y propensa a errores. Los modelos relacionales permiten resolver estos conflictos a través del uso de transacciones, propiedades ACID y sus niveles de aislamiento.
Transacciones	Sin lugar a duda, las transacciones son indispensables para garantizar consistencia de datos, en especial cuando los cambios se realizan de forma parcial debido a la ocurrencia de errores. Para ello se emplean las instrucciones commit , rollback , savepoint .
Integración	Soluciones empresariales pueden estar formadas por múltiples aplicaciones. Almacenar y administrar sus datos en una misma base de datos permite compartirlos y utilizarlos de forma integrada.
Seguridad	Los mecanismos de seguridad pueden ser aplicados a nivel de registro y columna. El uso de vistas previene cambios de usuarios no autorizados
Portabilidad	Código SQL portable respecto otras bases de datos SQL
Restricciones y tipos de datos	Tipos de datos personalizados y el uso de restricciones permiten validar los datos antes de ser almacenados. Esto incrementa la calidad de los datos.
Conceptos y tecnología conocida	El equipo de desarrollo está familiarizado con los conceptos de modelado ER y SQL

Concepto	Desventajas - RDBMS
Impedance mismatch	<p>Una de las principales frustraciones de los developers es el problema del mapeo entre 2 modelos:</p> <ul style="list-style-type: none"> • Las estructuras de memoria que manejan las aplicaciones, generalmente representadas por objetos (programación orientada a objetos). • Las estructuras que definen el modelo relacional. <p>A nivel general, el modelo relacional es más restrictivo. Los datos se organizan en relaciones, tuplas y columnas. Todas las operaciones SQL consumen y regresan relaciones (tablas). La limitante radica en que los</p>

Concepto	Desventajas - RDBMS
	<p>valores de una tupla son simples (un solo dato), no pueden contener otras estructuras.</p> <p>Por otro lado, las estructuras de memoria organizadas en objetos permiten el diseño de estructuras complejas y anidadas.</p> <p>Para realizar la persistencia de estas estructuras complejas, los desarrolladores deberán realizar un mapeo o transformación entre modelos que requiere programación y conocimientos sólidos. Entre los principales productos que permiten simplificar este proceso están Hibernate, JPA, etc., llamados a nivel general Object Relational Mappings (ORMs).</p> <p>Otro problema es el desempeño, en especial con volúmenes importantes de datos.</p>
Dependencia de un modelo	Los modelos ER y relacional deben ser concluidos y aprobados antes de iniciar con el desarrollo y pruebas.
Falta de escalabilidad	Pueden existir problemas de escalabilidad cuando se emplean operaciones join
Dificultad para realizar sharding	Es posible realizar sharding empleando múltiples servidores, aunque esta funcionalidad requiere código de la aplicación (los developers deben programar), la solución final puede ser ineficiente.
Problemas con disponibilidad	Puede ser complicado almacenar datos que requieren una alta disponibilidad en tablas.
Problemas al realizar integración de datos	<p>Si bien, una BD relacional (base de datos integradora) permite la integración de datos para ser compartidos entre aplicaciones, con el paso del tiempo, y el crecimiento de los datos, la base de datos integrada se vuelve compleja, mucho más allá de lo que una simple aplicación requiere: Aplicaciones simples tienen que trabajar con una base de datos integrada y compleja afectando de forma negativa al desempeño.</p> <p>Para resolver lo anterior, en lugar de utilizar una base de datos integradora, ahora se tiene una base de datos de aplicación la cual es utilizada únicamente por su respectiva aplicación</p>

Concepto	Desventajas - RDBMS
	<div data-bbox="423 247 1446 558"> </div> <p>La solución anterior permite varias cuestiones:</p> <ul style="list-style-type: none"> • Cada equipo de desarrollo se enfoca a su aplicación facilitando su mantenimiento y evolución (la idea detrás de los microservicios). • Cada base de datos puede ser inclusive de diferente tipo. • Debido a que el mismo equipo de desarrollo controla tanto la aplicación como la base de datos, algunas funcionalidades que típicamente estaban implementadas en la base de datos, ahora se pueden implementar en la aplicación. <ul style="list-style-type: none"> ◦ Ejemplo: Si la base de datos 3 no tiene soporte para manejar cierta concurrencia, esta se puede implementar en la aplicación. Al quitarle responsabilidades a la base de datos, su administración se vuelve simple, y por lo tanto aumenta su desempeño. Esta es una característica de los sistemas NoSQL.
Necesidad de esquemas flexibles	<p>El boom de la arquitectura basada en microservicios ha revolucionado la forma en la que se desarrolla software. Las Aplicaciones se comunican a través de servicios web tipo REST, empleando el protocolo HTTP.</p> <div data-bbox="683 1272 1170 1703"> </div> <p>La comunicación se realiza generalmente a través de mensajes en formato XML o JSON principalmente. Estos formatos semi-estructurados requieren esquemas de almacenamiento mucho más flexible que el que ofrece el modelo relacional ya que se desea enviar estructuras de datos complejas para minimizar el uso de la red.</p>

NoSQL

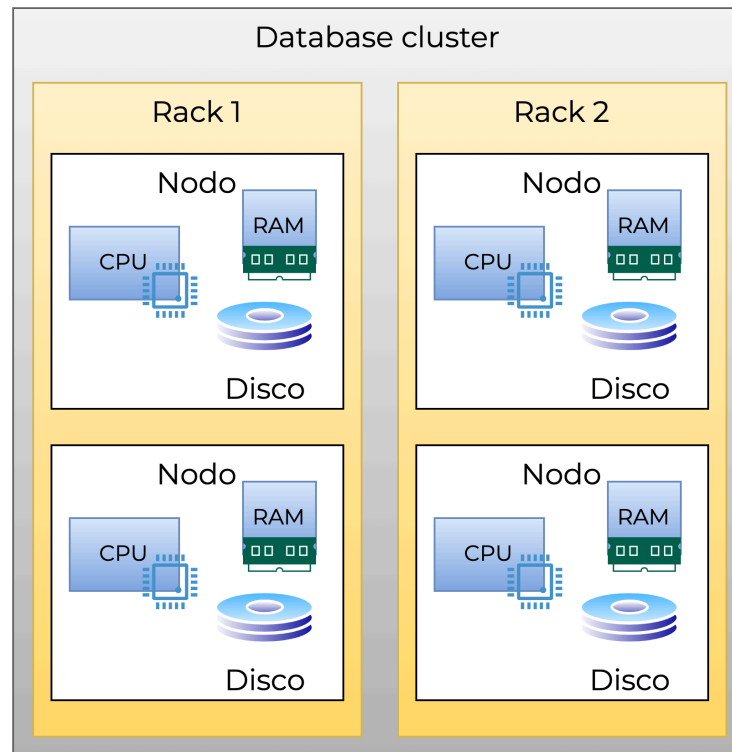
Concepto	Ventajas Sistemas NoSQL
Modelos flexibles	La carga de datos puede realizarse con herramientas simples, no requiere que el modelo ER esté completo
Arquitectura modular	Arquitectura modular permite el reuso de componentes, cada componente es relativamente simple
Sharding como funcionalidad nativa.	Sharding es una característica propia de los sistemas NoSQL, no requiere desarrollo adicional.
Funciones especializadas y nativas	Funciones de búsqueda integradas incrementa la calidad de los resultados
Sin ORMs	No existe capa de mapeo objeto relacional, por lo que el desarrollo se simplifica y mejora en desempeño. Los modelos flexibles permiten almacenar estructuras complejas
Variabilidad de datos	Facilidad para almacenar datos con alta variabilidad.

concepto	Desventajas NoSQL
Transacciones no siempre disponibles	Las transacciones ACID existen pero solo a nivel de documento. Otras transacciones deberán ser implementadas
Seguridad no nivel particular	No existen mecanismos de seguridad para controlar el acceso a elementos individuales dentro de un mismo documento, por ejemplo, documentos JSON.
Curva de aprendizaje	Las tecnologías NoSQL no son tan familiares, y conocidas por los developers
Estándares	No existen estándares como en el caso de SQL.
Incompatibilidad	Algunos tipos de sistemas NoSQL no son compatibles con herramientas OLAP existentes.

1.8. Conceptos Básicos

1.8.1.Clusters

Un cluster está formado por un conjunto de máquinas llamadas **nodos**, agrupados en **racks**. Cada nodo se compone de su propio hardware: CPU, memoria, disco y generalmente son máquinas simples, de costo relativamente bajo.



El CPU de cada nodo puede estar formado por un solo procesador, por varios, o por procesadores multi-núcleo. El disco de cada nodo de forma similar puede ser uno solo, o varios discos con controladoras independientes.

Los nodos se agrupan en racks los cuales se comunican entre sí generalmente con conexiones de anchos de banda alta.

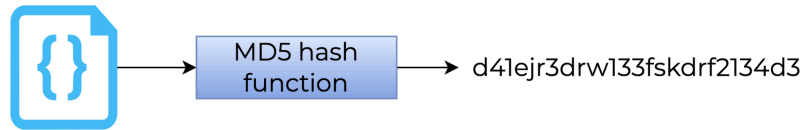
Un conjunto de racks forman un cluster. Múltiples clusters pueden existir en el mismo data center.

En algunos casos, para que una transacción NoSQL se considere exitosa, se deberán realizar operaciones en las bases de datos en nodos ubicados en diferentes data centers geográficamente distantes.

1.8.2. Consistent Hashing

Es una característica de las bases NoSQL que permite administrar y mantener la mayor cantidad de datos y consultas en memoria para mejorar el desempeño: **datos en caché**.

Este concepto le permite conocer si una consulta, un documento, o un dato existe en caché. Esto permite realizar accesos innecesarios a disco, por lo que permite que la base de datos funcione de forma más rápida.



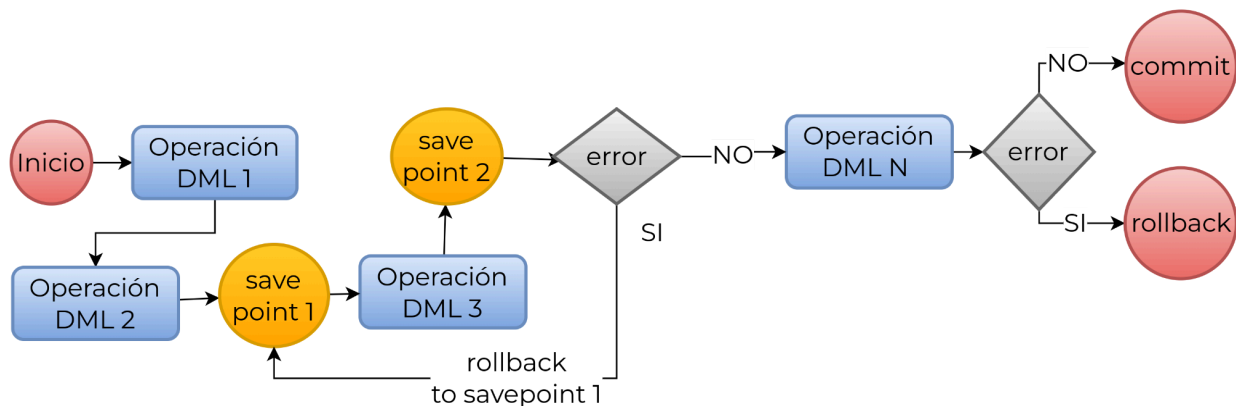
1.8.3. ACID Vs BASE

El control y manejo de transacciones en ambientes distribuidos para garantizar consistencia y buen desempeño son 2 aspectos muy importantes.

- RDBMS hace uso de las propiedades ACID de las transacciones
- Sistemas hacen uso de BASE (Basically Available, Soft state, Eventual consistency)

1.8.3.1. Transacción

Recordando el concepto de transacción



- Conjunto de instrucciones DML. Generalmente, la primera instrucción DML en ejecución provoca la creación de una nueva transacción en caso de no existir.
- La transacción termina su ciclo de vida al ejecutar las instrucciones **commit** o **rollback**.
- En cualquier punto se pueden establecer marcas llamadas *save points*. Un *save point* se emplea para realizar operaciones de rollback parciales: deshacer todas las operaciones realizadas hasta un cierto *save point*. Un rollback realizado hacia un checkpoint no produce el término de una transacción. En la siguiente imagen, se realiza una primera validación de un error. De ocurrir, se aplica un rollback parcial hasta el *save point 1*, es decir la operación DML 3 se revierte. La transacción puede continuar después de realizar un rollback parcial (**rollback to savepoint**).

1.8.3.2. *Propiedades ACID de las transacciones en RDBMS*

Atomicidad

Una transacción se integra por varias instrucciones DML. Este agrupamiento permite al RDBMS considerar a todas las instrucciones como si se tratará de una sola. Por lo tanto, si una instrucción falla, la transacción fallará y podrá revertirse, es decir, o se hace todo bien, o no se hace nada si algo falla. Este comportamiento permite conservar el estado consistente de la base de datos a pesar de la existencia de diversas fallas durante la ejecución de la transacción.

Consistencia

Mientras una transacción T1 se encuentre en ejecución, otros usuarios o transacciones no deberían ver los cambios que T1 está realizando ya que aún no termina. Si otras transacciones pudieran leer los cambios, problemas de inconsistencia de datos podrían aparecer (lecturas sucias). Los cambios de T1 serán visibles (lecturas confirmadas) hasta que sus cambios sean confirmados (**commit**). Los RDBMS se encargan de aislar o detener a otras transacciones que intenten leer o escribir datos que están siendo leídos o modificados por T1.

Aislamiento (Isolation)

Cada operación que forma parte de una transacción se ejecuta de forma totalmente independiente respecto a otras transacciones, es decir, las transacciones no se comunican entre sí, no tienen conocimiento de los cambios que realizan otras transacciones.

Durabilidad

Una vez que los cambios de una transacción han sido confirmados, estos se consideran permanentes, inclusive si ocurre alguna falla posterior a la confirmación. Existen mecanismos que permiten recuperar todas las transacciones confirmadas a pesar de la ocurrencia de fallas (media o instance failures)

Una de las principales técnicas para implementar las propiedades ACID de las transacciones es a través de mecanismos de bloqueo, cierre o reserva de recursos (**locking**).

Los sistemas que hacen uso de las propiedades ACID consideran a la consistencia e integridad como los principales objetivos, incluso por encima del desempeño.

1.8.3.3. *BASE en sistemas NoSQL*

Los sistemas NoSQL utilizan una técnica diferente, no siempre es factible que una transacción deba esperar a que los recursos se liberen o a que una transacción T1 termine para que una transacción T2 comience. Imaginar que un cliente intenta capturar una orden de compra pero esta es bloqueada debido a que otro proceso está realizando cálculos de inventario o inclusive actualizando el inventario de productos de la tienda. ¡E/

cliente se irá con la competencia por no obtener su orden de compra de forma rápida! .
La alternativa es BASE.

Basic Availability

Permite a los sistemas estar temporalmente inconsistentes para permitir que las transacciones operen de forma más flexible, es decir, la disponibilidad para ofrecer un servicio debe ser algo primordial “*basically available*”

Soft state

Se permite la existencia de datos que pueden ser *no adecuados* de forma temporal y que los datos pueden cambiar mientras se estén empleando, esto con la finalidad de reducir el uso de recursos.

Eventual consistency

Se asume que de forma eventual, cuando la ejecución de la lógica de un servicio ha terminado, el sistema puede quedar en un estado de inconsistencia.

Como se puede observar, RDBMS se enfoca en **consistencia** mientras que NoSQL se enfoca en **disponibilidad**.

- Sistemas que hacen uso de BASE favorecen a la captación e inserción de datos inclusive a pesar del riesgo de perder sincronía o consistencia de datos por periodos cortos de tiempo.
- Los Sistemas BASE se consideran *optimistas* en el sentido que eventualmente todas aquellas posibles inconsistencias serán corregidas y recuperar así el estado consistente de la BD.
- Sistemas BASE tienden a ser rápidos y simples ya que no requieren la escritura de código para bloquear y desbloquear recursos.

La siguiente tabla muestra un resumen de estos 2 conceptos. Con base a las reglas de negocio y requerimientos no funcionales de cada sistema, se elige la estrategia adecuada, o inclusive ambas.

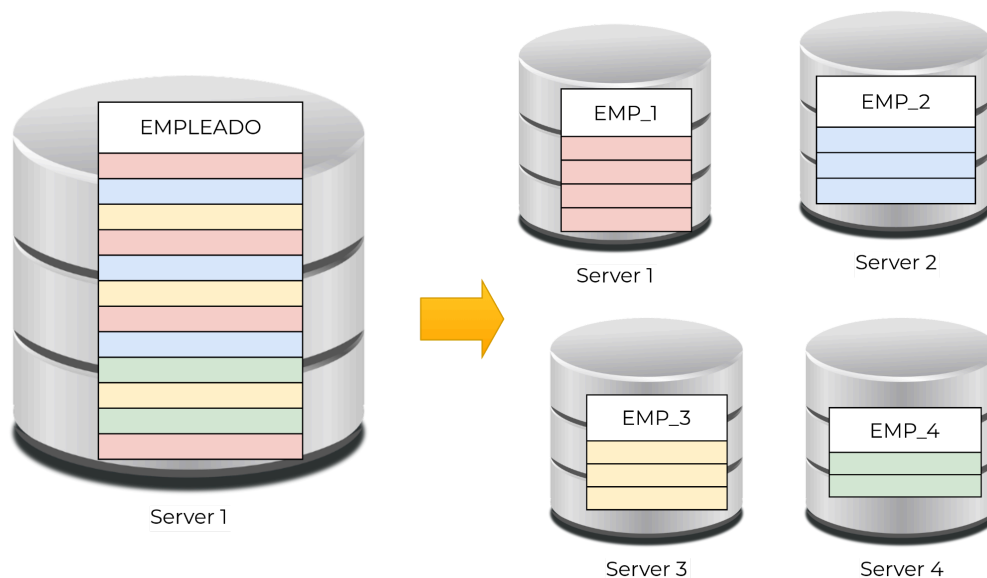
ACID	BASE
Bloquea peticiones de escritura si el dato está siendo modificado por otra transacción	Nunca se realizan bloqueos de escritura
Su enfoque principal es mantener la consistencia	Su enfoque principal es la disponibilidad del servicio: throughput
Su enfoque es ser pesimista. Asume que habrá múltiples peticiones leyendo y escribiendo un mismo dato por lo que aplica bloqueos y validaciones para garantizar consistencia.	Su enfoque es optimista: Asume que pueden ocurrir inconsistencias, pero eventualmente estos problemas se van a solucionar para recuperar el estado consistente.

ACID	BASE
El resultado de aplicar una transacción siempre debe ser correcto: datos consistentes.	Algunos resultados pudieran ser inconsistentes ¡No importa!
Gran cantidad de operaciones lock and unlock	Mantener todo simple sin operaciones lock

1.8.3.4. Database sharding

Conforme la cantidad de datos de una empresa u organización crece de manera importante de tal forma que excede las capacidades del entorno de ejecución (hardware) surge la necesidad de realizar una *distribución* de los datos.

Existe una técnica que permite realizar esta distribución llamada **sharding**.



Sharding significa dividir y distribuir los datos en subconjuntos llamados **shards** (fragmentos).

- Cada subconjunto se define empleando reglas de fragmentación. Por ejemplo, un sistema pudiera almacenar a todos los clientes cuya primera letra de su apellido se encuentre en el rango [A-L] y en otro nodo, los apellidos en el rango [M-Z].
- Las reglas de fragmentación deben ser definidas cuidadosamente para evitar problemas de desempeño. Por ejemplo, mantener datos que se consultan juntos en un solo nodo, etc.
- Los datos se distribuyen en sus respectivos nodos de forma automática conforme se ingresan al sistema.
- Si se agrega un nuevo nodo al cluster, los datos serán migrados al nuevo nodo dependiendo las reglas de fragmentación sin tener que detener el sistema.

- Si un nodo presenta problemas, o si existen problemas de red, la base de datos debe seguir funcionando. A esta capacidad se le conoce como **tolerancia al particionamiento**. Para implementar esta capacidad, se agrega una nueva funcionalidad: **replicación**. Sin embargo, esto incorpora un nuevo reto: mantener las réplicas de datos sincronizadas.

1.8.3.5. El teorema de CAP

Originalmente propuesto por Eric Brewer en el año 2000. Este teorema indica que un sistema de bases de datos distribuido puede contar a lo más con 2 de las siguientes 3 funcionalidades:

Consistencia: No confundir con el concepto de la propiedad de consistencia como parte de las propiedades ACID de las transacciones. En este caso, la consistencia se refiere a que múltiples clientes deberán leer el mismo dato en cualquiera de sus fragmentos replicados, es decir, obtener la misma versión de los datos.

Alta disponibilidad: La base de datos distribuida siempre permitirá a los clientes actualizar datos sin tener que esperar debido a la ocurrencia de una falla. Fallas internas para sincronizar los datos replicados, no debería impedir actualizaciones.

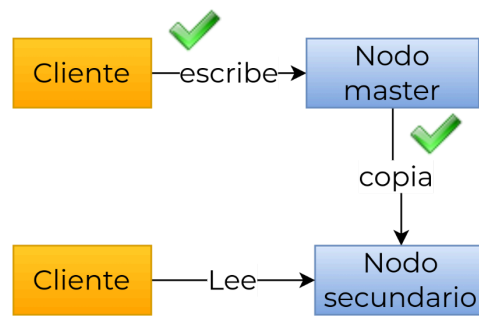
Tolerancia a la partición: Habilidad para responder a pesar de existir fallas de comunicación entre los nodos donde se encuentran los fragmentos de la BD (shards).

El teorema de CAP aplica cuando sistemas distribuidos presentan problemas de comunicación entre nodos de un cluster.

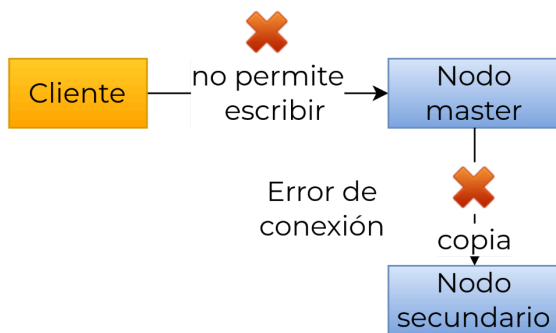
Básicamente el teorema de CAP ayuda a decidir a qué funcionalidad darle mayor prioridad: alta disponibilidad o consistencia cuando ocurra una falla de comunicación entre nodos.

Si la red es muy confiable, no hay fallas, este teorema podría omitirse ya que las 3 funcionalidades estarán soportadas.

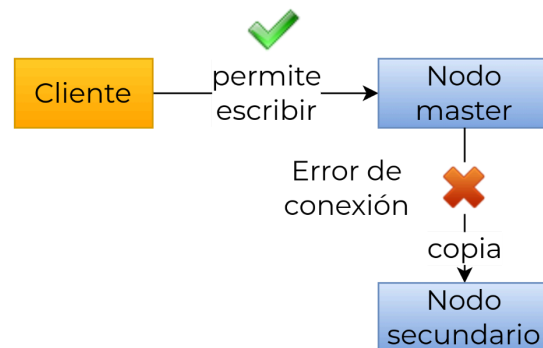
Operación normal



Opción 1: Consistencia



Opción 2: Alta disponibilidad



De la imagen anterior:

- Notar que en el caso de una operación normal, el cliente realiza cambios en un nodo master, su cambio es replicado exitosamente en el nodo secundario.
- Si la conexión con el nodo secundario se pierde, el cliente pudiera decidir entre las 2 opciones mostradas:
 - **Opción 1:** El cliente no podrá escribir en el nodo master hasta que la comunicación con el nodo secundario se restablezca.
 - **Opción 2:** Se acepta hacer cambios en el nodo master tomando como riesgo que si un cliente lee en el nodo secundario, podrían existir inconsistencias ya que no se ha realizado la sincronización por problemas de red.

¿ Qué sucede si ambas características son necesarias?, es decir, se requiere una alta consistencia y una alta disponibilidad ?

R: No distribuir y realizar un escalamiento vertical.



1.8.4. Selección de Sistema de bases de datos

El teorema de CAP permite comprender y ayudar a decidir qué sistemas de base de datos podría ser el adecuado para satisfacer las necesidades de un proyecto u organización

cuyos datos están distribuidos en varios servidores. Notar que se pueden seleccionar distintas bases de datos, no solo una.

1.8.4.1. Consistencia y tolerancia a la partición

Ejemplos de bases de datos enfocadas a proporcionar consistencia y tolerancia a la partición son:

- BigTable
- HBase
- Redis
- MongoDB

Si se presentan problemas de red entre los nodos, las lecturas se permiten, pero las escrituras fallarán hasta que la comunicación se restablezca y poder garantizar consistencia en todos los nodos.

1.8.4.2. Disponibilidad y tolerancia a la partición

Ejemplos de bases de datos enfocadas a proporcionar disponibilidad y tolerancia a la partición son:

- Cassandra
- Dynamo
- KAI
- CouchDB
- Riak

En estos sistemas, una petición de escritura será atendida en alguno de los nodos activos. Al realizar una lectura, el valor obtenido pudiera ser inconsistente debido a que tal vez la última versión en el nodo replicado aún no ha sido sincronizada.

1.8.4.3. Sistemas NewSQL

Representa una versión o evolución de las bases de datos relacionales que incorporan (mezclan) algunas funcionalidades de los sistemas NoSQL, es decir, proporcionar algunos beneficios de los sistemas NoSQL como son escalabilidad y alto rendimiento sin dejar de ser bases de datos relacionales, *¡las propiedades ACID se conservan!*

Por lo anterior, estos sistemas son capaces de **cumplir** con los 3 atributos del teorema de CAP. Algunos ejemplos de estos sistema son:

- ClustrixDB
- NuoDB
- CockroachDB
- TokuDB
- Apache Trafodion

**Tarea**

Seleccionar 3 sistemas **NewSQL** mencionados anteriormente, o investigar otros, investigar sus principales características, y usos prácticos