[Visión General NoSQL](#_y0wt66o35wdr)

[NoSQL](#_ft38oyle2op2)

[Características de las bases de datos NoSQL](#_uedugcz6pbco)

[Diferencias con las bases de datos relacionales](#_354vc2vjok1s)

[Motivaciones para elegir una base de datos Nosql](#_ytzgj84oefab)

[Escalabilidad](#_rnaol4at4noe)

[Costo](#_m31wnpivfzr0)

[Flexibilidad](#_y2bnyjqvvj51)

[Introducción a NoSQL](#_v67wbrke710p)

[Tipos de bases de datos NOSQL - Taxonomía de soluciones](#_pyi0s3c1g5iq)

[Persistencia Políglota](#_pjfffl3ih9sr)

[Modelo de Distribución de datos](#_784d9jb6prhh)

[Replicación](#_h2461rvs2k00)

[Sharding](#_b3i64c4c42u5)

[Clasificación de las DB según el teorema CAP](#_42g187makvii)

[Consistencia o coherencia (consistency)](#_bw85p1gb4oom)

[La disponibilidad (availability)](#_kkpvd063xded)

[Tolerancia al particionado (partition tolerance)](#_krbkjbef6d36)

[ACID versus BASE](#_815c6f800cym)

[Fuentes](#_i42iqx2wywn9)

# **Visión General NoSQL**

Sin lugar a dudas el paradigma que domina en el mundo de las bases de datos es el modelo o paradigma relacional. Actualmente la mayoría de los sistemas y aplicaciones utilizan algún RDBMS (libre o comercial). Todos coincidimos en que dichos RDBMS son muy sólidos, robustos y se han consolidado a lo largo de los años en el mercado, entre otras cosas, debido a que utilizan un lenguaje estándar de consultas SQL. A pesar de todas las ventajas conocidas en el paradigma relacional, este también presenta limitaciones propias de su arquitectura. Es por ello que Amazon y Google desarrollaron un sistema de almacenamiento basado en el uso de clusters, obligados por sus necesidades de negocio: Big tables de google y Dynamo de Amazon. Estas soluciones permitían gestionar y procesar enormes volúmenes de información en sistemas distribuidos. Dio el origen entonces a la aparición de las bases de datos NoSQL.

El surgimiento de [Big Data](https://es.wikipedia.org/wiki/Big_data) generó también nuevas necesidades y formas de procesamiento y almacenamiento de la información. En este nuevo contexto, lo más frecuente es tratar con datos no estructurados o semiestructurados, en grandes cantidades (imaginar muy grandes cantidades, como la información que maneja una red social, o un portal de streaming de video, etc.), en donde el procesamiento rápido es fundamental, y donde cuestiones tales como la consistencia de los datos deja de ser un requisito y se puede flexibilizar como así también la cuestión de un esquema fijo. Estos temas fundamentales en un RDBMS dejan de ser esenciales. El procesamiento distribuido se convierte ahora en un requisito de primer orden y en consecuencia surgen términos tales como *clusters*, *alta disponibilidad*, *replicación de datos*, *escalamiento horizontalmente*, entre otros….todos éstos temas los desarrollaremos a lo largo del curso. Si bien es cierto que –de alguna manera– el paradigma relacional puede adaptarse –en mayor o menor grado- a estos nuevos requisitos, no es la mejor solución ni fueron pensados para ese propósito. En este escenario es que surgen las bases de datos NoSQL. Este tipo de base de datos no es homogéneo y es posible encontrar distintas aproximaciones a los mismos problemas. Las bases de datos NoSQL no han surgido para reemplazar a las bases de datos relacionales, sino que las complementan. Dan soluciones efectivas, donde el paradigma relacional ya no podía debido sus características propias inherentes o resulta muy costoso hacerlo.

| Lectura complementaria: [Ranking y tendencia de base de datos](http://db-engines.com/en/ranking_trend) |
| --- |

| Lectura complementaria: [Infografía del uso de internet en 60 segundos](https://blogger.googleusercontent.com/img/a/AVvXsEg2lwGZ02-7k2UgbjFemGWCC53qbj9pe-5t9uB67BpEiDiH4HFsbw_qSNd42ne1_DIB31gCnka4scUoF4s1faV7KSnS9xxlT30mE4XmYzzZkO1iqGnuvHYluVyMhfie4ib-DX27nH6--eGMxNayV5iQ_1DXZD11b0UsrM4Mqw2ZK2gKQKIh6cpQ5XI2=s650) |
| --- |

# **NoSQL**

Son muchas las aplicaciones web que utilizan algún tipo de bases de datos para funcionar. Hasta ahora estábamos acostumbrados a utilizar bases de datos SQL tales como MySQL[[1]](#footnote-0), MariaDB[[2]](#footnote-1), PostgreSQL[[3]](#footnote-2), pero desde hace ya algún tiempo han aparecido otras que reciben el nombre de [NoSQL](http://nosql-database.org/)(Not only SQL – No sólo SQL, existen otras opciones diferentes a las SQL) y que han llegado con la intención de complementar a las bases relacionales utilizadas por la mayoría de los usuarios. No han surgido para reemplazar (a las BD relacionales), sino que surgieron como una alternativa donde las bases de datos relacionales no podían responder ni adaptarse debido a las limitaciones propias de su arquitectura.

Teniendo en cuenta la evolución de las páginas Web.

| **Web 1.0** | **Web 2.0** | **Web 3.0** |
| --- | --- | --- |
| La web primitiva, la del siglo 20, era aquella que se caracteriza principalmente por ser unidireccional y realizada sobre contenidos estáticos. Las primeras páginas que vimos en Internet publicaban contenidos de texto que, una vez publicados, no se actualizaban salvo que el "webmaster" modificarse dichos contenidos y volviese a subir la web de nuevo a internet.  La web 1.0 tenía un carácter principalmente divulgativo, y empezaron a subir a internet documentos e información principalmente cultural. Poco a poco las empresas empezaron a tomar parte y las primeras webs de empresa surgieron, con diseños muy pobres (no había herramientas, ni tecnología, ni conexión suficiente como para hacerlo mejor) y contenidos que rápidamente quedaban anticuados al ser complejo actualizarlos. | La web 2.0 se asentó a mediados de la primera década de este siglo. Sustentada bajo unas conexiones a internet evolucionadas (ya teníamos ADSL), y mejores herramientas para desarrollar web, mejores servidores, etc., la web 2.0, también denominada "la red social", llena Internet de blogs, wikis, foros y finalmente, redes sociales.  El objetivo de la web 2.0 es la *compartición* del conocimiento, es la web *colaborativa* y ha sido uno de los atractivos principales para atraer a usuarios (basta ver los usuarios de facebook que, hasta facebook, no eran usuarios frecuentes de computadoras. | La web 3.0 es la web semántica, la web de la nube, la web de las aplicaciones y la web multidispositivo. Hoy en día ya no solamente utilizamos ordenadores para conectarnos a Internet.  Las tablets, los smartphones, e incluso los mapas interactivos, algunas tiendas, y en un futuro no lejano la automoción estará consumiendo Internet.  La web 3.0 se presenta como una web inteligente (aunque creemos que para esto aun falta bastante), y principalmente aprovecha la nube para prestar servicios al usuario y eliminar su necesidad de disponer de sistemas operativos complejos y grandes discos duros para almacenar su información. |

Se puede decir que la aparición del término *NoSQL* surge con la llegada de la *web 2.0* ya que hasta ese momento sólo subían contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter, Instagram o Youtube, cualquier usuario puede subir contenido (casi) ilimitadamente, provocando así un crecimiento exponencial de los datos (hay un cambio de paradigma). Inclusive el avance del hardware, procesadores más veloces, multiprocesamiento, redes más rápidas, contribuyeron al desarrollo de las base de datos NoSQL sumado a que las bases de datos relacionales no pueden dar la respuesta que se espera a esta nueva necesidad de negocio.

En muchos casos es posible almacenar enormes volúmenes de información en las bases de datos relaciones. Pero es, en ese momento, cuando se quiere gestionar dicha información, que empiezan a aparecer los primeros problemas. En un principio, para solucionar estos problemas de accesibilidad, las empresas optaron por utilizar un mayor número de máquinas pero pronto se dieron cuenta de que esto no solucionaba el problema, además de ser una solución muy cara y costosa de implementar.

Por lo tanto referirse a bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales presentan ciertas dificultades y limitaciones debido principalmente a problemas de escalabilidad y rendimiento, donde se produce el acceso de miles de usuarios concurrentes con millones de consultas diarias. Además de lo comentado anteriormente, las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad–relación.

Los diseñadores de bases de datos recurrieron a los sistemas NoSQL cuando los RDBMS existentes no satisfacían sus necesidades de negocio.

La escalabilidad, el costo, la flexibilidad y la disponibilidad son preocupaciones cada vez más importantes para los desarrolladores de aplicaciones, y su elección de los sistemas de administración de bases de datos refleja esto. Por lo tanto, podemos decir que los sistemas de administración de datos han evolucionado para cumplir con los requisitos cambiantes de las aplicaciones y los negocios, sujeto a las restricciones de las tecnologías de computación y almacenamiento existentes en su momento.

A pesar del uso generalizado y exitoso de las bases de datos relacionales, el crecimiento exponencial del comercio electrónico y las redes sociales llevó a la necesidad de contar con otro sistemas de gestión de datos: Escalable, de bajo costo, flexible y altamente disponible. Lograr algunos de estos objetivos con bases de datos relacionales es posible en algunos casos, pero a menudo con dificultades y costos potencialmente altos.

Las bases de datos NoSQL evolucionaron para abordar las limitaciones de los sistemas de administración de bases de datos relacionales. Es poco probable que las bases de datos NoSQL desplacen a las bases de datos relacionales.Los RDBMS desplazaron las bases de datos de archivos planos, jerárquicas y de red. Es muy probable que los dos se complementen entre sí y adapten las características entre sí, ya que ambos siguen siendo aplicados a aplicaciones cada vez más complejas y exigentes.

Ventajas que ofrecen las bases de datos NoSQL

* Se ejecutan en máquinas con pocos recursos (en relación a los necesarios para una base de datos relacional - *commodities servers*): Estos sistemas, a diferencia de los sistemas basados en SQL, requieren de apenas computación, por lo que se pueden montar en máquinas de un coste mucho más reducido.
* Mayor velocidad en el desarrollo de aplicaciones: En el desarrollo de aplicaciones utilizando el paradigma relacional se invierte mucho esfuerzo en realizar mapeos entre las estructuras de datos en memoria y el modelo relacional. En este paradigma aparece el concepto de *polimorfismo* o *schemaless*.
* Gestión de grandes volúmenes de datos. Con estas soluciones, empresas y organismos gubernamentales encuentran una alternativa muy eficiente para tener muchos datos y procesarlos rápidamente. Si bien es posible –o podría ser posible de alguna manera con el paradigma relacional– podría resultar excesivamente costoso (en tiempo, esfuerzo e inversión). El motivo de esto, es que las bases de datos fueron concebidas para operar en modo single-instance (una única instancia) y a parte, es más barato procesar muchos datos que están distribuidos en muchos servidores (conformando un cluster). Se puede maximizar el uso de las bases de datos NoSQL si se ejecutan en un cluster.
* No genera cuellos de botella (bottleneck). El principal problema de los sistemas SQL es que necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que constituye un punto de entrada en común, que ante muchas peticiones puede ralentizar el sistema ([SPOF](https://es.wikipedia.org/wiki/Single_point_of_failure)).

# **Características de las bases de datos NoSQL**

* Arquitectura distribuida. Las bases de datos relacionales suelen estar centralizadas en una única máquina o bien en una estructura máster–esclavo, sin embargo en los casos NoSQL la información puede estar compartida en varias máquinas mediante mecanismos de tablas Hash distribuidas.
* Algunas no utilizan SQL como lenguaje de consultas, otras sí. Por lo general no utilizan el esquema relacional ni el lenguaje de consulta SQL.
* Son de código abierto (la gran mayoría).
* Se ejecutan en cluster para maximizar sus prestaciones.
* No garantizan las propiedades ACID (hay que analizar caso por caso). En general, proveen consistencia débil (weak consistency), como p.ej. consistencia eventual,
* Las bases de datos NoSQL no tienen un esquema fijo o rígido. Se dice que las bases de datos NoSQL son *polimórficas*, esto es, no existe un esquema uniforme. Cuando se van a almacenar datos en una base de datos relacional, lo primero que hay que conocer es su esquema, es decir, que tablas hay, cuáles son sus columnas, tipos de datos, etc. Las bases de datos NoSQL operan sin esquema fijo, lo que permite añadir libremente “campos a los registros (filas)” sin tener que re-definir ni conocer su estructura.

Cuando no se usa un esquema fijo, cada registro puede contener exclusivamente sólo aquello que necesita. En un esquema fijo además es necesario conocer previamente el esquema, lo que muchas veces es difícil y complejo y en otras ocasiones hasta puede no conocerse. Cuando no se tiene que cumplir un esquema, se puede almacenar fácilmente solo lo que se necesita, permitiendo cambiar acorde a las necesidades puntuales del momento. Y si hay información que ya no es más necesaria, no se requiere seguir almacenándola, por lo que, se puede dejar de almacenar sin preocuparse por perder datos, como si se borrasen columnas que ya no se utilizan en el paradigma relacional. Las bases de datos sin esquemas, cambian el esquema dentro del código de la aplicación que lo accede. Esto puede ser un problema si existen múltiples aplicaciones desarrolladas por diferentes personas que acceden a la misma base de datos.

# **Diferencias con las bases de datos relacionales**

Algunas de las diferencias más destacables que nos podemos encontrar entre los sistemas NoSQL y los sistemas SQL están:

* **Lenguaje de Consultas:** Las bases de datos SQL utilizan lenguaje de consulta estructurado (SQL) para definir y manipular datos. Por un lado, esto es extremadamente poderoso: el lenguaje SQL es una de las opciones más versátiles y ampliamente utilizadas disponibles en el mercado, lo que resulta en una opción segura e ideal para consultas complejas. Pero por el otro, puede ser altamente restrictivo. Las bases de datos SQL requieren que use esquemas predefinidos para determinar la estructura de sus datos antes de trabajar con ellos.

Una base de datos NoSQL tiene un esquema dinámico para datos no estructurados, es decir, no utilizan estructuras fijas para el almacenamiento de los datos (concepto de *polimorfismo*). Los datos se almacenan de muchas maneras, lo que significa que pueden estar orientados a documentos, a columnas, a gráficos u organizados como un almacén KeyValue. Esta flexibilidad significa que los documentos se pueden crear sin tener una estructura definida primero. Además, cada documento puede tener su propia estructura única. La sintaxis varía de una base de datos a otra, y se puede agregar “columnas” en la medida que se necesite.

No suelen permitir operaciones JOIN. Al disponer de un volumen de datos tan extremadamente grande (y distribuido entre los nodos que componen el cluster) suele resultar deseable evitar los JOIN. Esto se debe a que, cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa. Las soluciones más directas consisten en desnormalizar los datos, o bien realizar el JOIN mediante software, en la capa de aplicación.

* **Escalabilidad:** En casi todas las situaciones, las bases de datos SQL son escalables verticalmente. Esto significa que podemos mejor las prestaciones de un servidor de base de datos, agregando más memoria, procesamiento (CPU) o SSD (discos sólidos).

Las bases de datos NoSQL están diseñadas para escalar horizontalmente con mucha facilidad y sin límite, es decir podemos agregar más servidores muy fácilmente en su infraestructura. Las bases de datos relacionales también pueden escalar horizontalmente pero con muchísimas limitaciones.

| ***Paradigma relacional vs NoSQL***key_icon.jpg  *Ambos pueden escalar horizontalmente y verticalmente.*  *¿Cuáles son las limitaciones y restricciones de cada paradigma?* |
| --- |

* **La Estructura:** Las bases de datos SQL están basadas en tablas, mientras que las bases de datos NoSQL están basadas en documentos, pares de clave-valor, bases de datos de gráficos entre otras. Esto hace que las bases de datos SQL relacionales sean una mejor opción para aplicaciones que manejan transacciones, como por ejemplo un sistema de contabilidad, o para sistemas heredados que se crearon originalmente para una estructura relacional.

# 

# **Motivaciones para elegir una base de datos Nosql**

Las aplicaciones web que deben ofrecer su servicio a miles o tal vez millones de usuarios. Una solución de esta naturaleza es imposible de implementar con base de datos relacionales.

Dan Sullivan[[4]](#footnote-3) propone cuatro características de los sistemas de gestión de base de datos que son particularmente importantes para las tareas de gestión de datos a gran escala. Ésta son:

* **Escalabilidad**
* **Costo**
* **Flexibilidad**
* **Disponibilidad**

Dependiendo de las necesidades y requisitos de la aplicación en particular, algunas de estas características pueden ser más importantes que otras.

## Escalabilidad

La escalabilidad es la capacidad de satisfacer de manera eficiente las necesidades de diferentes cargas de trabajo. Por ejemplo, si hay un aumento en el tráfico de un sitio web, se pueden conectar servidores adicionales para manejar la carga adicional. Cuando el pico cede y el tráfico vuelve a la normalidad, algunos de esos servidores adicionales pueden cerrarse o liberarse. Agregar servidores según sea necesario se llama escalar horizontalmente (scale out).

Cuando se trabaja con bases de datos relacionales, a menudo es difícil escalar horizontalmente, aunque no imposible. Es posible que se necesite software de base de datos adicional para administrar múltiples servidores que funcionan como un solo sistema de base de datos. Oracle, por ejemplo, ofrece Oracle Real Applications Clusters (RAC) para bases de datos basadas en clústeres. Los componentes adicionales de la base de datos pueden agregar complejidad y costo a las operaciones.

Alternativamente, los administradores de bases de datos podrían optar por escalar verticalmente (scale up), que es actualizar un servidor de base de datos existente para agregar procesadores adicionales, memoria, ancho de banda de red u otros recursos que mejorarían el rendimiento en un sistema de administración de bases de datos o reemplazar un servidor existente con uno con más CPU memoria, y así sucesivamente.

La escalabilidad horizontal es más flexible que la escalabilidad vertical ya que los servidores se pueden agregar o quitar según sea necesario. Las bases de datos NoSQL están diseñadas para utilizar servidores disponibles en un clúster con una intervención mínima de los administradores de bases de datos. A medida que se agregan o eliminan nuevos servidores al cluster, el sistema de administración de bases de datos NoSQL se ajusta para usar el nuevo conjunto de servidores disponibles.

Decimos también que es más flexible porque la escalabilidad vertical puede requerir la sustitución de un servidor. Esto implica la migración de los datos por el administrador de base de datos a un nuevo servidor.

Además, escalar verticalmente para agregar recursos (procesadores o memoria por ejemplo) no requeriría una migración, pero probablemente requeriría algún tiempo de inactividad para agregar hardware al servidor de la base de datos (downtime).

## Costo

El costo de las licencias de base de datos es una consideración significativa para cualquier empresa u organización (y mucho más si esta es una pyme). Los proveedores de software comercial emplean una variedad de modelos de licenciamiento que incluye el cobro por el tamaño del servidor que ejecuta el RDBMS (cantidad de cores del procesador) o el número de usuarios nombrados autorizados para usar el software. Cada uno de estos modelos presenta desafíos para los usuarios del sistema de base de datos.

Por ejemplo, las aplicaciones web pueden tener picos de demanda que aumentan el número de usuarios que utilizan una base de datos en cualquier momento:

*¿Deberían los usuarios de RDBMS pagar por la cantidad de usuarios pico o la cantidad de usuarios promedio? ¿Cómo se deberían adquirir las licencias RDBMS cuando es difícil saber cuántos usuarios usarán el sistema dentro de seis meses o un año?*

Los usuarios de software de código abierto evitan estos problemas. El software es de uso libre en todos los servidores productivos que se requiera y del tamaño que sea necesario. Afortunadamente para los usuarios de bases de datos NoSQL, las principales bases de datos NoSQL están disponibles como código abierto.

Las compañías de terceros o comerciales brindan servicios de soporte comercial para bases de datos NoSQL de código abierto para que las empresas puedan tener soporte de software como lo hacen con bases de datos comerciales relacionales.

## Flexibilidad

Los sistemas de administración de bases de datos relacionales son muy flexibles en cuanto al rango de problemas que nos permite resolver:

Industrias tan diferentes como la banca, la manufactura, la función pública, el comercio minorista, la energía y la atención médica son ejemplos de negocios que hacen uso de bases de datos relacionales (y muy buen uso).

Sin embargo, hay otro aspecto de las bases de datos relacionales que las hace mucho menos flexibles.

Los diseñadores de bases de datos deben saber al inicio de un proyecto todas las tablas y columnas que serán necesarias para diseñar una aplicación y poder comenzar con el diseño conceptual. Se debe conocer previamente todo el modelo.

Por ejemplo, todos los empleados tienen que tener nombre y apellido y ID. Si bien, luego es posible cambiar la estructura de las tablas, la metodología está orientada a hacer el diseño de la base de datos completo al comienzo del proyecto.

Considere una aplicación de comercio electrónico que utiliza una base de datos relacional para documentar los atributos de sus productos. Los productos informáticos, como las computadoras, tendrían atributos como el tipo de CPU, la cantidad de memoria y el tamaño del disco. Los hornos de microondas tendrían atributos tales como tamaño y potencia.

Un diseñador de base de datos podría crear tablas separadas para cada tipo de producto o definir una tabla con tantos atributos de producto diferentes como ella podría imaginar en el momento en que diseña la base de datos.

A diferencia de las bases de datos relacionales, algunas bases de datos NoSQL no requieren una estructura de tabla fija.

Por ejemplo, en una base de datos de documentos, un programa podría agregar dinámicamente nuevos atributos según sea necesario sin tener que tener un diseñador de base de datos para modificar el diseño de la base de datos (concepto de polimorfismo).

**Disponibilidad**

Muchos de nosotros -en algún momento- tuvimos que esperar a que los sitios web y/o las aplicaciones web estén disponibles cuando queríamos usarlos *¿Te pasó alguna vez con el sistema de inscripción de la facultad?* Si su sitio favorito de redes sociales o comercio electrónico no funcionaba cuando intentó usarlo, es probable que comience a buscar otras opciones, obviamente cuando esto sea posible.

Las bases de datos NoSQL están diseñadas para aprovechar múltiples servidores de bajo costo. Cuando un servidor falla o se pone fuera de servicio por mantenimiento, los otros servidores del clúster pueden asumir toda la carga de trabajo.

El rendimiento puede ser algo menor, y en el caso de verse degradado la aplicación seguirá estando disponible. Ahora, si una base de datos se ejecuta en un solo servidor y este falla, la aplicación dejará de estar disponible a menos que haya un servidor de respaldo o réplica.

Los servidores de réplica mantienen copias de los datos del servidor primario en caso de que falle el servidor primario. Si eso sucede, la copia (la base replicada) puede asumir la carga de trabajo que el servidor primario había estado procesando.

Esta puede ser una configuración ineficiente porque un servidor se mantiene en reserva en caso de una falla pero de lo contrario no está ayudando a procesar la carga de trabajo.

# **Introducción a NoSQL**

## Tipos de bases de datos NOSQL - [Taxonomía de soluciones](http://nosql-database.org/)

Los principales tipos de BBDD de acuerdo con su implementación son los siguientes:

* Base de datos Clave-Valor: Riak, Redis, Dynamo, Voldemort.
* Base de datos orientadas a documentos: MongoDB, CouchDB
* Base de datos basadas en columnas: Cassandra, Hypertable, HBase, SimpleDB
* Base de datos de grafos: Neo4J, Infinite Graph.

Como podemos ver, existen varias tecnologías nosql y [actualmente existen más de 200](http://nosql-database.org/) bajo el término NoSQL.

# **Persistencia Políglota**

Para entender mejor a que se refiere con este término podríamos definir la palabra políglota. Su definición es: *“alguien que habla o escribe varios idiomas”*. Entonces basado en esa definición pero llevado a base datos, podemos definir persistencia políglota como un conjunto de aplicaciones que utilizan varias tecnologías básicas de base de datos que se usan para resolver problemas complejos de almacenamiento.

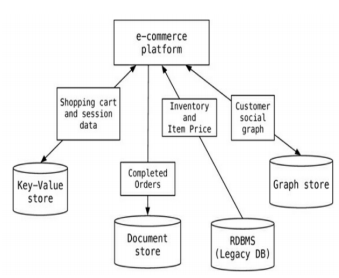
¿Qué modelo o paradigma de base de datos debemos adoptar para nuestra próxima aplicación? La pregunta que podemos hacernos es: ¿por qué elegir solo uno? ¿deberíamos? ¿No sería conveniente utilizar diferentes estrategias para almacenar y recuperar la información para aprovechar las ventajas que cada una ofrece?

A esta idea se la llama **persistencia políglota**, que ya estaba presente desde el surgimiento de las Bases OLAP que se utilizaban para explotar los datos y hacer análisis de alto nivel (Business Intelligence). Ahora debemos sumar a las bases NoSQL como alternativa para el manejo de la persistencia de nuestra solución.

Diferentes modelos de bases de datos están diseñadas para resolver diferentes problemas, es decir, cada una tiene propósito. El uso de un único motor de base de datos para cubrir todos los requisitos de una aplicación suele conducir a soluciones no eficientes. Por lo tanto, estos problemas complejos se fragmentan en partes más pequeñas y entonces se aplican diferentes modelos de base de datos según las necesidades específicas de cada requerimiento del problema. Esta solución también trae inconvenientes pues es necesario una nueva aplicación o un servicio maneje la comunicación con los diferentes tipos de almacenamientos y análisis de datos.

Debido a que persistencia políglota es una idea y forma de trabajo, no una aplicación ni una tecnología específica, esta no tiene un origen claro. En 2006, Neal Ford (Director, Software Architect, Meme Wrangler en ThoughtWorks)[[5]](#footnote-4) acuñó el término *programación políglota*, para expresar la idea de que las aplicaciones deberían ser escritas en una mezcla de idiomas para aprovechar el hecho de que diferentes lenguajes son adecuados para abordar diferentes problemas. Aplicaciones complejas combinan diferentes tipos de problemas, por lo que escoger el lenguaje adecuado para cada trabajo puede ser más productivo que tratar de encajar todos los aspectos en un solo lenguaje. Con el paso del tiempo, se ha aplicado este método de trabajo al área de base de datos y este ha ido evolucionando en este campo, manteniendo la idea original la cual es trabajar con distintos tipos de tecnologías pero ahora en vez de lenguajes de programación, se utilizan distintas base de datos dentro de una misma aplicación, buscando guardar los datos de la forma más óptima (soluciones híbridas).

Ejemplo para una plataforma de e-commerce:



# **Modelo de Distribución de datos**

En el paradigma relacional es posible escalar verticalmente (scale up). Esto significa que si la cantidad de datos que necesito procesar crece exponencialmente podría escalar comprando un servidor mucho más potente. Ahora debemos ser conscientes que no es donde el paradigma relacional se siente más cómodo o donde puede lucir todas sus mayores características. Una solución mejor consiste en ejecutar la base de datos en un cluster. Por este motivo, las bases de datos NoSQL son muy interesantes y constituyen la opción ideal para ejecutarse sobre clústeres de grandes dimensiones.

Existen dos modelos de distribución de datos:

## Replicación

Consiste en tomar los datos y copiarlos, en tiempo real, a otros servidores. Existen varios enfoques de replicación. Maestro-Esclavo o Peer-to-peer. En el caso del Maestro-Esclavo un único nodo denominado Maestro es sobre el que se realizan las actualizaciones y este “replica” “copia” en los esclavos. Hay un único Maestro a la vez y es el único autorizado a recibir cambios desde la aplicación.

En el caso de la replicación peer-to-peer se permite la escritura sobre cualquier nodo, de manera que los nodos se coordinan para sincronizar las copias de los datos.

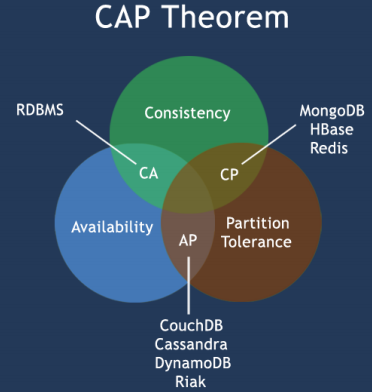
## Sharding

Este concepto, se podría traducir como particionado. Se emplea esta técnica para gestionar (balancear de alguna manera) la carga de los servidores. Consiste en distribuir los datos entre distintos shards (conjuntos de servidores que almacenan parte de los datos), para que se reparta la carga a la hora de realizar consultas e inserciones. Dicho de otra manera, los datos están distribuidos, repartidos entre los distintos shards. Cada servidor o nodo, actúa como un único repositorio de dichos datos.

# **Clasificación de las DB según el teorema CAP**

Las bases de datos NoSQL están pensadas para ser escalables y distribuidas. Y para ser distribuidas tendremos que tener en cuenta el teorema CAP. El teorema de CAP establece que, si se consideran las propiedades de **disponibilidad**, **consistencia** (coherencia) y **tolerancia a la partición** en un sistema, solo es posible conseguir dos de ellas a la vez.

*“You can have it good,you can have it fast, you can have it cheap: pick two.”*



## Consistencia o coherencia (consistency)

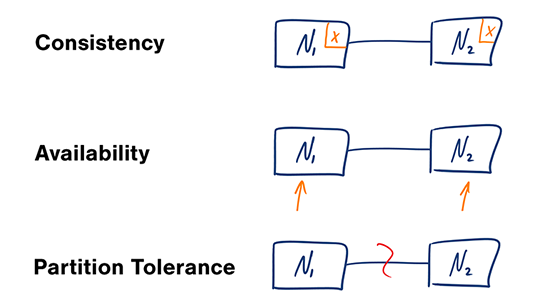
En sistemas distribuidos, habitualmente se dice que se encuentra en un estado *consistente* si, después de una operación de escritura, todas las operaciones de lectura posteriores son capaces de ver las actualizaciones desde la parte del sistema desde la que están leyendo. Dicho de otra manera, que todos los usuarios vean la misma información al mismo tiempo. Por lo tanto con la consistencia se garantiza que una lectura devolverá la escritura más reciente para un cliente determinado.

## La disponibilidad (availability)

La alta disponibilidad se logra cuando el sistema ha sido diseñado e implementado de modo que se pueda continuar operando (lecturas, escrituras), incluso después de que algún nodo haya sufrido algún inconveniente, tanto de hardware como de software.

## Tolerancia al particionado (partition tolerance)

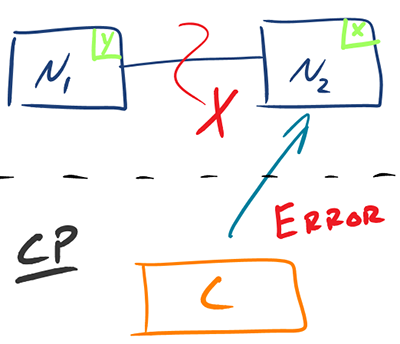
Los sistemas distribuidos pueden estar divididos en particiones (generalmente de forma geográfica). Así que esta condición implica, que el sistema tiene que seguir funcionando aunque existan fallos o caídas parciales que dividan el sistema. El sistema tiene que seguir funcionando aunque existan fallos en la comunicación de dichas particiones (las particiones están activas pero no pueden comunicarse entre sí).



Fuente: <https://robertgreiner.com/cap-theorem-revisited/>

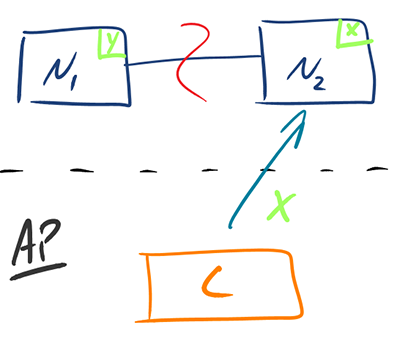
El teorema CAP se extendió en base a la idea original que solo puede elegirse 2 de las 3 variables para un sistema de cómputo distribuido. Aunque la frase es poco feliz como el mismo autor [sugirió](https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed) años después, sí es cierto que tenemos al menos dos estrategias posibles para manejar la información sobre una red:

* Si no queremos particionar la información, y queremos mantenerla centralizada, no hay necesidad de hablar de consistencia ni de disponibilidad. Este es el caso de las bases relacionales y las de grafos (**CA**).
* Si por el otro lado queremos trabajar con un esquema de particionado, las opciones son priorizar Consistencia o Disponibilidad.
* **CP:** Mongodb, HBase, Redis, MemcacheDB. La consecuencia de esta elección es que las actualizaciones y las consultas deben ser atómicas y pueden devolver un error de timeout (se penaliza la disponibilidad).



Fuente: <https://robertgreiner.com/cap-theorem-revisited/>

* **AP:** Cassandra, Dynamo, CouchDB. La consecuencia de esta elección es que la información recuperada puede no ser la última, a favor de tener una alta disponibilidad de todos los nodos.



Fuente: <https://robertgreiner.com/cap-theorem-revisited/>

# **ACID versus BASE**

El mundo de las bases de datos relacionales está familiarizado con las transacciones ACID. Las transacciones que se producen en el lenguaje SQL, sea cual fuere el sistema gestor de base de datos cumplen siempre las propiedades ACID. Este tipo de transacciones se llaman así porque garantizan la Atomicidad, la Consistencia, el aislamiento (Isolation) y la Durabilidad.

El modelo BASE es un enfoque similar al ACID, aunque perdiendo la consistencia y el aislamiento, a favor de la disponibilidad, la degradación y el rendimiento.

**BASE** es una filosofía de diseño de sistemas de datos que pondera la disponibilidad sobre la consistencia de las operaciones. BASE se desarrolló como una alternativa para producir arquitecturas de datos más escalables y accesibles ofreciendo más opciones a las empresas

El modelo BASE toma su nombre de:

* **Basic Availability.** El sistema funciona incluso cuando alguna parte falla, debido a que el storage sigue los principios de distribución y replicación. BA es básicamente disponible. Esto significa que puede haber una falla parcial en algunas partes del sistema distribuido y el resto del sistema continúa funcionando.

Por ejemplo, si una base de datos NoSQL se ejecuta en un cluster con 10 servidores sin replicar datos y uno de los servidores falla, entonces el 10% de las consultas de los usuarios fallará, pero el 90% lo logrará. Las bases de datos NoSQL a menudo conservan múltiples copias de datos en diferentes servidores. Esto permite que la base de datos responda a las consultas incluso si uno de los servidores ha fallado.

* **Soft State.** Los nodos no tienen porque ser consistentes entre sí todo el tiempo. En las operaciones NoSQL, se refiere al hecho de que los datos pueden sobrescribirse con datos más recientes.
* **Eventual Consistency.** La consistencia se produce de forma eventual. Esto significa que puede haber ocasiones en que la base de datos se encuentre en un estado inconsistente.

Por ejemplo, algunas bases de datos NoSQL mantienen varias copias de datos en varios servidores. Sin embargo, existe la posibilidad de que las copias múltiples no sean consistentes por un corto período de tiempo.

Esto puede ocurrir cuando un usuario o programa actualiza una copia de los datos y otras copias siguen teniendo la versión anterior de los datos.

Eventualmente, el mecanismo de replicación en la base de datos NoSQL actualizará todas las copias, pero mientras tanto, las copias son inconsistentes.

El tiempo que toma actualizar todas las copias depende de varios factores, como la carga en el sistema y la velocidad de la red.

Considere una base de datos que mantiene tres copias de datos. Un usuario realiza una actualización en un servidor. El sistema de gestión de base de datos NoSQL actualiza automáticamente las otras dos copias. Una de las otras copias está en un servidor en la misma red de área local, por lo que la actualización se realiza rápidamente. El otro servidor se encuentra en un centro de datos[[6]](#footnote-5) a miles de kilómetros de distancia, por lo que hay un retraso en la actualización de la tercera copia. Un usuario que ejecuta una consulta en el tercer servidor mientras la actualización está en curso puede obtener el valor anterior del usuario, mientras que alguien que consulta el primer servidor recibe el nuevo valor.

# **Fuentes**

1. Artículo: <https://eamodeorubio.wordpress.com/2010/05/17/nosql-2-no-necesitas-acid/> [Consultado: Agosto 2016]

1. Artículo: <https://docs.mongodb.com/manual/> [Disponible: Agosto 2016]

1. Introducción a las bases de Datos NoSQL usando MongoDB, Antonio Sarasa

1. Artículo: <http://db-engines.com/en/ranking_trend> [Disponible: Febrero 2017]

1. Dan Sulivan (201) - NoSQL for Mere Mortals (Inglés) 1st Edición

1. Artículo: <https://robertgreiner.com/cap-theorem-revisited/> [Disponible: Marzo 2020]

1. <https://www.mysql.com/> [↑](#footnote-ref-0)
2. <https://mariadb.org/> [↑](#footnote-ref-1)
3. <https://www.postgresql.org/> [↑](#footnote-ref-2)
4. <https://www.amazon.com/Dan-Sullivan/e/B001IXSA1Y/ref=dp_byline_cont_book_1> [↑](#footnote-ref-3)
5. N. Ford, “about me,” 2006, [Online; accessed 20-may-2017] [↑](#footnote-ref-4)
6. <https://es.wikipedia.org/wiki/Centro_de_procesamiento_de_datos> [↑](#footnote-ref-5)