**CASOS DE USO DE CADENAS REDIS**

**Caché de sesión:** muchos sitios web aprovechan Redis Strings para crear un [**caché de sesión**](https://redis.io/topics/data-types-intro) para acelerar la experiencia de su sitio web al almacenar en caché fragmentos o páginas HTML. Dado que los datos se almacenan temporalmente en la RAM, este atributo convierte a Redis en una opción perfecta como caché de sesión. Es capaz de almacenar temporalmente datos específicos del usuario, por ejemplo, artículos almacenados en un carrito de compras en una tienda en línea, lo cual es crucial para que sus usuarios no pierdan sus datos en caso de que cierren sesión o pierdan la conexión.

**Colas:** cualquier aplicación que se ocupe de la congestión del tráfico, la mensajería, la recopilación de datos, la gestión del trabajo o el enrutamiento de empaquetadores debe considerar una [**Cola de Redis**](https://www.infoworld.com/article/3230455/how-to-use-redis-for-real-time-metering-applications.html) , ya que esto puede ayudarlo a administrar el tamaño de la cola según la tasa de llegada y salida para la distribución de recursos.

**Uso y facturación medida:** un caso de uso menos conocido para Redis Strings es la medición en tiempo real para [**los modelos de precios basados**](https://www.infoworld.com/article/3230455/how-to-use-redis-for-real-time-metering-applications.html) en el [**consumo**](https://www.infoworld.com/article/3230455/how-to-use-redis-for-real-time-metering-applications.html) . Esto permite que las plataformas SaaS facturen según el uso real para medir la actividad de sus clientes, como en la industria de las telecomunicaciones, donde pueden cobrar por mensajes de texto o minutos.

**LISTA DE CASOS DE USO DE REDIS**

**Sitios de redes sociales:** las plataformas sociales como Twitter usan las listas de Redis para llenar sus líneas de tiempo o feeds de la página de inicio, y pueden personalizar la parte superior de sus feeds con tweets o historias de tendencia.

**Fuentes RSS:** cree fuentes de noticias a partir de fuentes personalizadas donde puede obtener las últimas actualizaciones y permitir que los seguidores interesados ​​se suscriban a su fuente RSS.

**Tablas de clasificación:** foros como Reddit y otras plataformas de votación aprovechan las listas de Redis para agregar artículos a la tabla de clasificación y ordenar por las entradas más votadas.

**QUÉ ES SQL Y NOSQL**

Empezaremos por definir brevemente estos dos conceptos:

En este caso, al utilizar las siglas SQL estamos haciendo referencia al modelo relacional, que se trata de un paradigma en los modelos de bases de datos basado en las relaciones entre los diferentes conjuntos de datos.

Algunos ejemplos de gestores de BDR: Oracle, PostgreSQL, MySQL, etc.

NoSQL, también conocido como Not Only SQL, consiste en una amplia gama de SGBD cuyas principales características son el uso de un lenguaje alternativo al SQL y que guardan información no estructurada. Es decir, no existen tablas ni columnas, como en el modelo relacional.

**ALGUNOS EJEMPLOS DE GESTORES DE BD NOSQL:** MongoDB, Cassandra, Redis, etc.

**PRINCIPALES CARACTERÍSTICAS DE SQL Y NOSQL**

Ahora que podemos distinguir entre los dos paradigmas, pasamos a ver las diferentes características que las definen:

**BASE DE DATOS RELACIONAL (SQL)**

**Modelo entidad-relación**: éstas siguen el modelo relacional que consiste en el almacenamiento de datos en entidades (tablas) formadas por filas y columnas que necesariamente se relacionan entre ellas.

**Flexibilidad en el esquema:** a diferencia de los sistemas no relacionales, estos esquemas de datos son muy poco flexibles.

**Lenguaje:** se caracterizan por utilizar el lenguaje SQL (que, debido a sus siglas en inglés, significa Structured Query Language). Se trata de los lenguajes más utilizados en los SGBD relacionales más populares, desde su fecha de lanzamiento en 1986. Permiten el acceso a bases de datos y a recuperar la información almacenada.

**ACID:** aplican y garantizan las siguientes propiedades conocidas como ACID,

**Atomicity:** Todo o nada. Si algo falla, toda la operación falla.

**Consistency:** Hace referencia a la integridad de la información. Garantiza que cualquier transacción que se pueda realizar en la base de datos la lleve de un estado válido a otro también válido.

**Isolation:** Con esta propiedad el sistema consigue que las diferentes operaciones que puedan concurrir sean visibles en el momento adecuado, asegurando que una operación no afecte a la otra.

**Durability:** Mediante la persistencia se consigue que, tras la operación, los datos estén asegurados aun cuando el sistema falle, garantizando así que la información se conserve.

Escalabilidad: En este caso, hablamos de una escalabilidad vertical. Se aumenta el hardware por uno más potente (disco duro, procesador, etc.).

**REDIS (NOSQL)**

**Consistencia eventual:** al contrario de lo que ocurre en las bases de datos relacionales, en NoSQL no se implementa un mecanismo rígido de consistencia que garantiza la integridad de los datos, sino que consta de un sistema de “consistencia eventual” en el que los cambios realizados se propagan en un intervalo de tiempo no determinado (por lo que no todos los observadores ven la actualización de datos inmediatamente).

**Flexibilidad en el esquema:** hablamos de esquemas de datos dinámicos, organizados en registros (llamados documentos) que pueden almacenar sus atributos en diferentes formas dependiendo de las necesidades que se dan en el momento.

**Escalabilidad horizontal:** se trata de la posibilidad de aumentar el rendimiento del sistema implementando más nodos (servidores).

**Estructura distribuida:** existen dos modelos de distribución de datos, mediante los cuales se distribuyen los datos entre los diferentes nodos que forman el sistema:

**Sharding:** los datos son distribuidos entre múltiples nodos o servidores, cada uno de los cuales corresponde a la única fuente de un subconjunto de datos.

**Réplica:** en este tipo de distribución, los datos son copiados entre múltiples servidores, de modo que la misma información puede encontrarse en diferentes lugares. Existe la réplica master-slave, que consiste en la gestión de la copia por parte de un servidor (master) con el cual el resto se sincroniza; y la peer-to-peer, en la que son todos los servidores que se coordinan para sincronizar sus copias.

Y bien, con lo que hemos visto hasta ahora ya podemos diferenciar claramente entre estos dos modelos de gestión de datos. En este post profundizaremos sobre cómo, cuándo y por qué utilizar NoSQL teniendo en cuenta sus pros y contras. De esta manera podrás decidir, de forma eficiente, qué paradigma se adecua más a tu proyecto.

**ENSAYO DE LA NORMALIZACIÓN NO SEA TAL VEZ NORMAL**

la escalabilidad no es un problema, hacer que la gente se preocupe por eso si . Por lo tanto, cuando se trata del diseño de bases de datos, mida el rendimiento, pero intente errar fuertemente en el lado del diseño sano y simple . Elija el esquema de base de datos que crea que es más fácil de entender y con el que trabaje a diario. No tiene que ser todo o nada como lo he visto arriba; puede desnormalizar parcialmente donde tiene sentido hacerlo, y mantenerse completamente normalizado en otras áreas donde no lo tiene.

Dare Obasanjo tuvo una excelente publicación Cuándo no normalizar su base de datos SQL en la que proporciona un esquema de base de datos de muestra para un sitio de redes sociales genérico.

La normalización sin duda cumple en términos de limitar la duplicación. Cada entidad está representada una vez, y solo una vez, por lo que casi no hay riesgo de inconsistencias en los datos. Pero ese diseño también requería de seis combinaciones para recuperar la información de un solo usuario.

Esas seis uniones tampoco están haciendo nada para ayudar al rendimiento del sistema. La normalización en toda regla no es simplemente difícil de entender y difícil de trabajar, también puede ser bastante lenta.Como señala Dare, la solución obvia es desnormalizar : colapsar una gran cantidad de datos en una sola tabla de Usuarios.

Esto funciona: las consultas ahora son increíblemente simples probablemente también muy rápidas. Pero tendrá un montón de huecos vacíos en sus datos, junto con una gran cantidad de matrices de campos con nombres incómodos. Y todos esos problemas molestos de integridad de datos que la base de datos usó para usted

Ambas soluciones tienen sus pros y sus contras. Entonces: ¿cuál es mejor: una base de datos normalizada o una base de datos desnormalizada

La respuesta es que no importa! Hasta que tengas millones y millones de filas de datos, eso es. Todo es rápido para los pequeños . Incluso una PC modesta para los estándares actuales, digamos una caja de doble núcleo con 4 gigabytes de memoria, le dará un rendimiento casi idéntico en cualquier caso para cualquier cosa que no sea la base de datos más grande.

**ENSAYO DE PROCEDIMIENTOS ALMACENADOS**

Es gratificante hablar que los procedimientos almacenados dan como resultado un mejor rendimiento y seguridad pero son mitos ya que se pueden presentar diversos escenarios donde se puede ver afectado el rendimiento la seguridad y la escalabilidad de nuestras bases de datos incluso se pueden presentar problemas entre versiones diferentes.

Para las bases de datos modernas y los escenarios de uso del mundo real,una arquitectura de Procedimiento almacenado tiene serias desventajas y pocos beneficios prácticos.

Los procedimientos almacenados se deben considerar como lenguaje de ensamblaje de la base de datos: para usar solo en las situaciones más críticas.

**ENSAYO DE BASES DE DATOS MÚLTIPLES**

Estas bases de datos de múltiples modelos pueden proporcionar un único back-end que expone múltiples modelos de datos a las aplicaciones que admite. De esa manera, las bases de datos de modelos múltiples eliminan la fragmentación y proporcionan un backend consistente y bien comprendido que admite muchos productos y aplicaciones diferentes. Los beneficios para la organización son extensos, pero algunos de los beneficios más importantes incluyen:

**Consolidación**

Una base de datos de múltiples modelos admite diferentes tipos de datos para diferentes casos de uso y los consolida en una plataforma. Por lo tanto, obtiene flexibilidad en el lenguaje de consulta y el modelo de datos, pero simultáneamente se beneficia de una tecnología de motor de almacenamiento común.

**Escala De Rendimiento**

Los sistemas multimodelo que desacoplan el lenguaje de consulta y el modelo de datos del almacén de datos subyacente permiten escalar diferentes componentes dentro de la arquitectura de manera independiente a medida que cambian las necesidades.

**Complejidad Operacional**

Los entornos fragmentados causados ​​por la ejecución de diferentes bases de datos aumentan la complejidad tanto de las operaciones como del desarrollo

**Flexibilidad**

Un enfoque de modelos múltiples implica la asignación de múltiples modelos de datos en un único motor de almacenamiento subyacente que puede admitir diferentes casos de uso y aplicaciones

**Fiabilidad**

La confiabilidad de la base de datos también es un problema cuando se ejecutan múltiples bases de datos, ya que cada sistema de base de datos podría ser un punto único de falla para el sistema y la aplicación más grandes.

**Consistencia De Los Datos.**

una funcionalidad de transacción de nivel superior integrada en su aplicación, no hay soporte para transacciones en diferentes sistemas de bases de datos. En consecuencia, no hay una buena manera de mantener la coherencia entre los diferentes modelos.

**Tolerancia A Fallos**

la integración de múltiples sistemas que fueron diseñados para ejecutarse de manera independiente para que brinden tolerancia a fallas en todo el sistema impone costos de ingeniería y operativos significativos.

**Costo**

El uso de más sistemas de bases de datos distintos aumenta los costos según el hardware, el software y las necesidades operativas asociadas con cada sistema.

**Transacciones**

esun desafío proporcionar transacciones a través de múltiples máquinas, y casi todas las bases de datos NoSQL no proporcionan garantías transaccionales debido a sus diseños arquitectónicos. Debido a que un verdadero sistema de múltiples modelos requiere transacciones para garantizar que los datos se almacenen de manera consistente en la base de datos, todas sus aplicaciones heredan este contrato sólido de cómo se almacenan los datos**.**

**Mejores Aplicaciones**

En contraste, una aplicación que es compatible con una base de datos de múltiples modelos obtiene los beneficios de escalabilidad, tolerancia a fallas y, en un sistema bien diseñado, alto rendimiento integrado en el producto. Con menos lógica adicional necesaria en el nivel de la aplicación para manejar las interacciones de la base de datos y las posibles condiciones de falla, los desarrolladores pueden concentrarse en crear mejores aplicaciones. Debido a estos beneficios, los sistemas multimodelo son a los que se dirige el mercado de bases de datos: transacciones compatibles con ACID, API multimodelo y potentes motores de almacenamiento compartidos