ACCESO A DATOS

UNIDAD 6. BASES DE DATOS NOSQL

CONTENIDO

1. INTRODUCCIÓN A NOSQL 3

1.1. CARACTERÍSTICAS PRINCIPALES 3

1.2. ORIGEN Y EVOLUCIÓN 5

1.3. COMPARACIÓN CON BASES DE DATOS SQL 7

2. TIPOS DE BASES DE DATOS NOSQL 11

2.1. BASES DE DATOS DOCUMENTALES 11

2.2. BASES DE DATOS DE COLUMNAS 16

2.3. BASES DE DATOS KEY-VALUE 22

2.4. BASES DE DATOS DE GRAFOS 27

3. OPERACIONES CRUD EN BASES DE DATOS NOSQL 33

3.1. OPERACIONES CRUD 33

3.2. IMPLEMENTACIÓN DE OPERACIONES CRUD EN DISTINTOS TIPOS DE NOSQL 43

RESUMEN 47

INTRODUCCIÓN

Las bases de datos NoSQL ofrecen soluciones a las limitaciones que presentan las bases de datos relacionales tradicionales. Estas últimas están construidas sobre un modelo rígido de tablas y relaciones, lo que, en ciertas circunstancias, resulta inadecuado para aplicaciones que requieren flexibilidad y una rápida adaptación a cambios en la estructura de los datos. Las bases de datos NoSQL se diseñan para superar estos desafíos, siendo más adecuadas para manejar grandes volúmenes de datos no estructurados o semi-estructurados, así como para satisfacer los requisitos de alta disponibilidad y escalabilidad de las aplicaciones modernas.

El NoSQL no se limita a un único tipo de base de datos, sino que abarca una variedad de tecnologías y modelos de almacenamiento. **La tipología de bases de datos NoSQL incluye las documentales, de columnas, de clave-valor y de grafos**, cada una con características específicas que las hacen idóneas para diferentes usos. **Las bases de datos documentales almacenan datos en documentos que suelen estar en formato JSON**, lo que proporciona una estructura flexible que permite a los desarrolladores almacenar información compleja con facilidad. Esta flexibilidad es particularmente valiosa en aplicaciones donde los campos y formatos de datos pueden variar de una entrada a otra.

Las bases de datos de columnas, por su parte, organizan los datos en columnas en lugar de filas, lo que permite una mejor compresión de datos y un acceso optimizado en escenarios de lectura intensiva. Este modelo es beneficioso en aplicaciones analíticas donde se necesita realizar consultas sobre grandes volúmenes de información agrupada. Por otro lado, **las bases de datos de clave-valor funcionan mediante un sistema simple que asocia cada clave a un valor específico**, favoreciendo la velocidad de acceso y la simplicidad en la manipulación de datos. Este enfoque resulta idóneo para aplicaciones que requieren respuestas rápidas, como las configuraciones del usuario o el almacenamiento de sesiones.

Las bases de datos de grafos se centran en la representación de relaciones y conexiones entre datos, utilizando nodos y aristas para modelar información que es inherentemente relacional. Este tipo de base de datos es especialmente útil en áreas como las redes sociales, donde la comprensión de las interacciones entre diferentes usuarios puede ofrecer insights valiosos. Las bases de datos de grafos permiten consultas complejas que se centran en las relaciones, lo que las hace únicas en su capacidad para abordar problemas relacionados con redes y conexiones.

Las operaciones CRUD son utilizadas para interactuar con cualquier tipo de base de datos, incluida la variedad NoSQL. Crear, Leer, Actualizar y Eliminar son las acciones básicas que los desarrolladores realizan al trabajar con datos. Para leer o consultar los datos, se pueden utilizar diversas técnicas de búsqueda para recuperar documentos específicos, a menudo basándose en la estructura y contenido de los mismos. La actualización de un registro puede implicar la modificación de un documento existente, mientras que la eliminación puede llevarse a cabo borrando un documento entero del almacenamiento.

1. INTRODUCCIÓN A NOSQL

NoSQL abarca un conjunto de tecnologías de gestión de bases de datos que se alejan del modelo relacional tradicional que caracteriza a las bases de datos SQL. A medida que las aplicaciones han evolucionado, ha surgido la necesidad de mayores niveles de flexibilidad y escalabilidad, algo que los sistemas de bases de datos SQL, con su estructura rígida y esquemas predefinidos, no pueden ofrecer de forma eficiente. **NoSQL se creó para satisfacer esta demanda, permitiendo el almacenamiento y la recuperación de grandes volúmenes de datos no estructurados o semiestructurados**.

Existen diversos tipos de bases de datos NoSQL, entre los que se encuentran las basadas en documentos, clave-valor, columnares y orientadas a grafos, cada uno adaptándose a diferentes necesidades y tipos de datos. Esta variedad es uno de los rasgos que distingue a NoSQL, ya que brinda a los desarrolladores la oportunidad de seleccionar la tecnología que mejor se ajuste a los requerimientos específicos de sus aplicaciones. **La capacidad de escalar horizontalmente, en lugar de hacerlo de forma vertical, es otro rasgo significativo; esto implica que se pueden añadir más servidores para hacer frente al aumento de cargas, lo que resulta en una mayor eficiencia y velocidad ante grandes volúmenes de tráfico**.

Los sistemas NoSQL también se caracterizan por la flexibilidad en la estructura de los datos, lo que permite almacenar información en formatos diversos y dinámicos. Esto resulta útil en aplicaciones que requieren una rápida adaptación a cambios en los requerimientos de datos, algo habitual en entornos de desarrollo ágiles. **Además, la mayoría de estas tecnologías están diseñadas para gestionar datos distribuidos, lo que facilita su implementación en la nube**, mejorando la disponibilidad y la resistencia a fallos.

En los últimos años, el incremento del uso de Big Data y la necesidad de integrar distintas fuentes de información han promovido la adopción de bases de datos NoSQL. A medida que las organizaciones buscan extraer valor de sus datos de maneras más flexibles y escalables, NoSQL se ha convertido en una opción preferida para muchas aplicaciones modernas que requieren un enfoque diferente en el almacenamiento y gestión de datos en contraste con los enfoques relacionales convencionales.

1.1. CARACTERÍSTICAS PRINCIPALES

Las bases de datos NoSQL gestionan grandes volúmenes de datos no estructurados y semi-estructurados. Este enfoque presenta ventajas en comparación con los sistemas de gestión de bases de datos relacionales, especialmente en términos de escalabilidad y flexibilidad.

1.1.1. Escalabilidad horizontal

Las bases de datos NoSQL permiten realizar escalado horizontal, lo que consiste en añadir más servidores o nodos para aumentar la capacidad de procesamiento de datos. Este modelo es utilizado por empresas que experimentan un crecimiento significativo de usuarios y datos. Por ejemplo, en

aplicaciones de transmisión de vídeo en línea como Netflix, es necesario que la infraestructura soporte la distribución de información entre múltiples servidores para manejar la carga de tráfico y ofrecer un rendimiento adecuado.

1.1.2. Flexibilidad del modelo de datos

La flexibilidad en el modelo de datos de las bases de datos NoSQL permite a los desarrolladores adaptar las estructuras a las necesidades cambiantes del negocio sin interrupciones. *En MongoDB, que utiliza formato de documentos JSON*, se pueden agregar nuevos campos a los registros sin alterar las entradas existentes. Un caso claro se observa en aplicaciones de gestión de contenido, donde un sitio web puede requerir actualizaciones frecuentes en su estructura de datos para incluir características como comentarios y valoraciones. Esta capacidad para ajustarse sin necesidad de hacer refactorización extensa del esquema resulta beneficiosa en entornos ágiles.

1.1.3. Acceso optimizado a los datos

Cada tipo de base de datos NoSQL tiene un enfoque particular para el acceso a información. L**as bases de datos de clave-valor, por ejemplo, son rápidas en el almacenamiento y recuperación de datos**. *Un uso común sería Redis* en aplicaciones que necesitan mantener sesiones de usuario, donde los datos de la sesión se almacenan con claves únicas y son accesibles casi instantáneamente, un aspecto importante para aplicaciones web de alto tráfico.

Las bases de datos en columnas, como Cassandra, permiten almacenar datos optimizados para consultas que requieren agregación. Cuando las empresas recopilan grandes volúmenes de datos, como registros de sesión de aplicaciones móviles, **pueden utilizar estas bases para realizar análisis rápidos sobre tendencias y comportamientos de usuarios**.

1.1.4. Modelado de relaciones complejas

Las bases de datos orientadas a grafos, como Neo4j, se centran en modelar relaciones entre diferentes entidades. Este enfoque resulta útil en aplicaciones donde las conexiones son relevantes para el análisis, como en plataformas de redes sociales. *En una aplicación como Facebook*, donde los usuarios se conectan entre sí, **un sistema orientado a grafos permite realizar consultas sobre relaciones de manera eficaz**, identificando amigos en común, grupos o perfiles que podrían ser de interés.

1.1.5. Modelo de consistencia eventual

El modelo de consistencia eventual es una característica común en muchas bases de datos NoSQL.

> Este modelo permite que los datos sean actualizados en diferentes nodos de forma asíncrona, dando prioridad a la disponibilidad y la partición de datos frente a una sincronización inmediata.

Esta propiedad es importante en plataformas de servicios en línea donde el rendimiento en tiempo real tiene prioridad. En aplicaciones de comercio electrónico, *por ejemplo, la disponibilidad y rapidez en el acceso a los datos de productos y precios durante eventos de ventas es más relevante que asegurar que cada nodo tenga la información más actualizada en todo momento*.

Las aplicaciones en la industria de los juegos online también utilizan consistencia eventual. Durante los eventos de un juego, la experiencia del usuario es prioritaria. El sistema puede realizar actualizaciones en tiempo real, permitiendo a los jugadores ver cambios inmediatos en sus puntuaciones o en el estado del juego, incluso si la información no está completamente sincronizada entre todos los nodos.

1.1.6. Casos de uso

Un área en la que las bases de datos NoSQL han influido considerablemente es el análisis de Big Data. Empresas como Uber utilizan estas bases para manejar grandes volúmenes de datos generados por usuarios y conductores. Sistemas de bases de datos como *Apache Hadoop* y *Apache Cassandra* permiten a Uber realizar análisis en tiempo real sobre patrones de viaje y optimización de rutas.

Otro ejemplo se observa en plataformas de comercio electrónico. Empresas como Amazon emplean diversas tecnologías NoSQL para gestionar recomendaciones personalizadas y analizar el comportamiento del consumidor. Almacenar los historiales de compras y navegación de los usuarios en una base de datos orientada a documentos permite a Amazon personalizar las experiencias de compra, mostrando productos relevantes conforme el usuario navega por el sitio.

Las bases de datos NoSQL representan un componente importante en la infraestructura de aplicaciones modernas, posibilitando a las empresas aprovechar la variedad y el volumen de datos generados diariamente. Estas soluciones versátiles se han convertido en una norma en el desarrollo de software, facilitando la innovación y adaptación a las demandas del mercado.

1.2. ORIGEN Y EVOLUCIÓN

El origen de las bases de datos NoSQL comienza por la necesidad de gestionar grandes volúmenes de datos de forma eficiente en entornos web en rápida evolución. A finales de la década de 2000, el aumento del uso de internet, junto con el crecimiento de redes sociales y comercio electrónico, creó una presión significativa sobre las bases de datos relacionales. Aunque estas estructuras son efectivas en situaciones específicas, presentan limitaciones al enfrentarse a la cantidad de datos no estructurados y semiestructurados que se generan.

> Las bases de datos NoSQL emergieron como una respuesta a estas limitaciones, proponiendo alternativas centradas en la flexibilidad y la escalabilidad.

Esta necesidad se reflejó en el auge de aplicaciones web, donde los modelos de datos tradicionales no podían ofrecer el rendimiento y la adaptación requeridos por aplicaciones que experimentan cambios frecuentes en sus datos y estructuras. *Por ejemplo, aplicaciones de mensajería y redes sociales, donde cada interacción crea nuevos datos que deben almacenarse y recuperarse rápidamente, demandan un enfoque que las bases de datos relacionales no podían proveer de manera efectiva*.

Twitter/'X' es un caso que ilustra esta transición. La plataforma, en sus inicios, utilizaba bases de datos relacionales, pero enfrentó problemas de rendimiento a medida que la cantidad de mensajes y usuarios aumentó rápidamente. Para resolver estos problemas, Twitter adoptó un enfoque de bases de datos no relacionales que les permitió escalar horizontalmente al combinar múltiples servidores y gestionar de forma eficiente el almacenamiento de datos. Implementaron tecnologías como Manhattan, un sistema de bases de datos distribuido interno que gestiona tanto datos estructurados como no estructurados, lo que permite un acceso ágil y una alta disponibilidad.

La evolución de NoSQL ha dado lugar a diferentes tipos de bases de datos, cada una optimizada para usos específicos. **Las bases de datos de documentos, como MongoDB y CouchDB**, permiten a los desarrolladores almacenar datos en estructuras más flexibles que los modelos de tablas convencionales. En una aplicación de comercio electrónico, por ejemplo, cada producto puede contar con atributos variables que no se aplican a todos los artículos, como tallas, colores o características específicas. **Con MongoDB, no es necesario definir un esquema rígido de antemano**; cada documento representa un producto con sus propios atributos, permitiendo que la aplicación evolucione sin limitaciones.

Las bases de datos clave-valor han mostrado su utilidad en situaciones donde la velocidad de acceso es importante. *Redis, por ejemplo, es ampliamente utilizado en aplicaciones que requieren almacenamiento en caché*, como sitios de ventas durante períodos de gran tráfico, donde se necesita recuperar rápidamente información sobre productos populares y mantener un rendimiento óptimo. Gracias a su estructura, los datos se pueden acceder en milisegundos, mejorando la experiencia del usuario.

Las bases de datos de grafos, como Neo4j, son herramientas valiosas para representar relaciones complejas entre datos. *Este tipo de base de datos es ideal en situaciones donde el análisis de conexiones es relevante, como en la detección de fraudes en transacciones financieras o la gestión de redes sociales*. Con grafos, las consultas sobre las relaciones entre usuarios, transacciones y comportamientos se simplifican, permitiendo a las organizaciones identificar patrones que serían difíciles de discernir en un modelo relacional tradicional.

A medida que las organizaciones adoptan NoSQL para satisfacer sus requerimientos de datos, se ha vuelto común **implementar arquitecturas de microservicios**. Este enfoque divide las aplicaciones en componentes independientes, cada uno utilizando el tipo de base de datos que se ajuste a sus necesidades. *Por ejemplo, en plataformas de streaming de video, se pueden emplear bases de datos de documentos para gestionar la información del contenido, bases de datos de grafos para analizar*

relaciones entre usuarios y recomendaciones, y bases de datos clave-valor para el almacenamiento en caché de sesiones de usuarios.

La evolución de NoSQL también se ha manifestado con el advenimiento del Big Data. Las organizaciones que manejan volúmenes masivos de datos, como las empresas de análisis de datos y procesamiento de transacciones en tiempo real, han encontrado en tecnologías como Apache HBase y Couchbase soluciones efectivas para sus problemas de almacenamiento. *HBase permite almacenar y recuperar grandes cantidades de datos en un entorno distribuido, mientras que Couchbase ofrece un enfoque orientado a documentos que facilita la escalabilidad y la sincronización de datos entre diferentes sistemas*.

La gestión y orquestación de bases de datos NoSQL ha evolucionado con la llegada de herramientas como **Kubernetes**, que permiten manejar clústeres de bases de datos distribuidas. Esto simplifica la actualización, el despliegue y la administración de múltiples instancias, facilitando el uso eficiente de recursos y la mejora continua de las operaciones. Las organizaciones están utilizando estas tecnologías para optimizar la infraestructura y reducir costos operativos, aprovechando la capacidad de escalar de manera dinámica según la demanda.

Con el avance en el desarrollo de aplicaciones y la gestión de datos, el ecosistema NoSQL se expande y se adapta a estas nuevas realidades. La variedad de opciones disponibles ofrece a las organizaciones la flexibilidad para elegir la solución más adecuada en función de sus propios requerimientos, lo que impulsa la adopción de estos sistemas en el mercado global. La diversidad de arquitecturas y modelos de datos que NoSQL proporciona transforma la manera en que las empresas consideran el almacenamiento y la gestión de datos, evidenciando la continua evolución del sector y su impacto en la tecnología de análisis y gestión de datos en el entorno moderno.

1.3. COMPARACIÓN CON BASES DE DATOS SQL

Las bases de datos SQL (relacionales) y NoSQL (no relacionales) presentan diferencias notables en su estructura y funcionamiento. Las bases de datos SQL utilizan un modelo estructurado que define tablas y relaciones, lo que permite la creación de esquemas rígidos donde cada registro debe adherirse a una estructura predefinida. En contraste, las bases de datos NoSQL ofrecen un modelo más flexible y escalable que permite almacenar datos en formatos diversos como documentos, clave-valor, columnas o grafos, sin imponer un esquema fijo.

Las bases de datos SQL emplean el lenguaje de consulta estructurado (SQL) para la manipulación de datos, lo que proporciona una sintaxis estándar para realizar operaciones complejas. Por su parte, **las bases de datos NoSQL no tienen un estándar unificado de consulta, lo que significa que cada tecnología puede utilizar su propio método para acceder y manipular los datos**, aportando mayor variedad y, a su vez, una curva de aprendizaje que depende del sistema elegido.

La escalabilidad también presenta diferencias entre ambos tipos de bases de datos. **Las bases de datos SQL suelen ser escaladas verticalmente**, lo que implica aumentar la capacidad del servidor

existente, aunque esto puede resultar limitado. **Las bases de datos NoSQL, en cambio, están diseñadas para escalar horizontalmente, lo que permite la adición de más servidores para distribuir la carga** y manejar grandes volúmenes de datos de manera más eficiente.

El modelo de consistencia varía igualmente. Las bases de datos SQL garantizan la consistencia y transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que es determinante para aplicaciones que requieren integridad de los datos en todo momento. Las bases de datos NoSQL pueden optar por modelos de consistencia eventual, facilitando un rendimiento más alto en situaciones donde la consistencia inmediata no sea prioritaria.

Estas diferencias y características muestran la necesidad de evaluar cuidadosamente las particularidades de cada tipo de base de datos según las necesidades específicas de cada proyecto o aplicación.

1.3.1. Elección entre SQL y NoSQL

La elección entre SQL y NoSQL depende de factores específicos relacionados con la naturaleza del proyecto y las características de los datos. Las bases de datos SQL, que operan bajo un modelo relacional, son altamente estructuradas. Los datos se organizan en tablas con filas y columnas, utilizando el lenguaje de consulta estructurado (SQL) para gestionar, consultar y manipular esos datos. Este tipo de sistema es adecuado para aplicaciones que requieren una estructura clara y un esquema fijo, así como para aquellas donde la integridad de los datos es un aspecto relevante.

Las bases de datos NoSQL, por otro lado, abarcan varios modelos de datos como clave-valor, documentos, columnas y grafos. Este enfoque permite almacenar información de forma más flexible, adaptándose a cambios en los requisitos y soportando datos semiestructurados o no estructurados. Esta adaptabilidad es útil en aplicaciones que necesitan crecer rápidamente o manejar datos en constante evolución.

La consistencia de los datos es un aspecto determinante en la elección de la base de datos. Las bases de datos SQL operan siguiendo principios **ACID**, que garantizan que las transacciones se completen de manera fiable. Este enfoque es especialmente relevante en entornos donde cada operación afecta significativamente a los datos y la necesidad de mantener la coherencia es notable. Por ejemplo, en sistemas de gestión hospitalaria, donde se registran datos de pacientes, es crítico que no se produzcan errores que puedan comprometer su atención.

Las bases de datos NoSQL siguen el modelo **BASE**, que se caracteriza por ser básicamente disponible y por permitir una consistencia eventual. *Esto implica que, en lugar de asegurar que todos los nodos de la base de datos tengan los mismos datos en todo momento, se permite un periodo temporal en el que la información puede no estar actualizada antes de alcanzar un estado de coherenci*a. Este principio es beneficioso en aplicaciones que priorizan la alta disponibilidad y la escalabilidad.

Por ejemplo, un sistema de recomendaciones en una plataforma de *streaming* puede beneficiarse de esta naturaleza eventual de una base de datos NoSQL. La base de datos puede almacenar

preferencias de usuarios y generar sugerencias en tiempo real. Aunque la información sobre las preferencias no esté sincronizada al instante entre todos los nodos, el servicio puede seguir funcionando, ofreciendo recomendaciones basadas en los datos disponibles, priorizando la experiencia del usuario.

Los tipos de datos son relevantes al considerar entre SQL y NoSQL. Las bases de datos SQL son adecuadas para datos estructurados con relaciones bien definidas, mientras que las bases de datos NoSQL destacan en el manejo de datos semiestructurados y no estructurados.

> En un sistema de gestión de datos de investigación científica, donde se deben almacenar resultados experimentales que pueden variar en formato y estructura, una base de datos de documentos como MongoDB permite almacenar datos en forma de documentos JSON, lo que facilita adaptaciones a formatos en evolución.

En el caso de una aplicación de comercio electrónico, gestionar productos, usuarios, pedidos y reseñas requiere un sistema que pueda integrar datos de orígenes y formatos diversos. Aquí, una base de datos NoSQL como DynamoDB podría ofrecer la flexibilidad necesaria al adaptarse rápidamente a nuevas necesidades del mercado, como la adición de atributos a los productos o la integración de funcionalidades vinculadas a redes sociales.

La escalabilidad es otro aspecto que influye en la decisión entre estos dos tipos de sistemas. Las bases de datos SQL suelen escalar verticalmente, lo que implica actualizar el hardware del servidor para aumentar el rendimiento. Esto puede resultar costoso y limitante a medida que la aplicación crece. En contraste, las bases de datos NoSQL están diseñadas para escalar horizontalmente, permitiendo distribuir datos entre múltiples servidores o nodos. Este método no solo ofrece mayor capacidad de procesamiento, sino que también puede reducir costos y riesgos asociados a un único punto de fallo.

> En una aplicación de redes sociales que atiende a millones de usuarios a nivel global, elegir una base de datos NoSQL como Cassandra permite que los datos, tales como publicaciones, comentarios y "me gusta", se distribuyan entre varios centros de datos, garantizando que la aplicación permanezca siempre disponible y responda rápidamente a las solicitudes de los usuarios, incluso durante picos de tráfico.

El tiempo de desarrollo y la velocidad de implementación también son aspectos relevantes en la selección de la base de datos. **Las bases de datos NoSQL suelen permitir un desarrollo más ágil graficando su flexibilidad**. Por ejemplo, en entornos donde los requisitos de software cambian con frecuencia y donde se busca iterar rápidamente sobre los productos, un equipo de desarrollo puede implementar cambios en el modelo de datos sin las complicaciones típicas de un sistema SQL que requeriría migraciones de esquema cada vez que se necesiten ajustes en el diseño de la base de datos.

En el desarrollo de un prototipo para una nueva aplicación que recoge opiniones de usuarios, utilizar una base de datos NoSQL coordinada con un marco ágil podría facilitar a los desarrolladores experimentar con diferentes tipos de datos y formatos, optimizando el sistema mientras se implementan nuevas características.

El control sobre las transacciones también es un aspecto influyente en la elección entre SQL y NoSQL. **Las bases de datos SQL permiten implementar transacciones que involucran múltiples tablas y operaciones atómicas. Esto es adecuado para aplicaciones que requieren coherencia en la actualización de datos**. Por ejemplo, en una aplicación de reserva de vuelos donde un cliente puede cambiar su vuelo, es necesario modificar varios registros, como el de la reserva original y el de la nueva, asegurando que se cumplan las reglas de disponibilidad.

Por otro lado, en una plataforma de análisis de datos que manipula grandes volúmenes de información en tiempo real, un sistema NoSQL como Apache Kafka puede ser más efectivo. Esta tecnología permite manejar flujos de datos continuos, facilitando la ingestión de grandes volúmenes y proporcionando la capacidad de reaccionar a eventos en tiempo real, sin las limitaciones que podrían surgir al manejar transacciones complejas.

La elección entre SQL y NoSQL debe considerar diversas variables específicas del caso de uso, desde la naturaleza de los datos hasta los requisitos de coherencia y escalabilidad, así como los tiempos y recursos para el desarrollo de software. Las decisiones informadas en este ámbito pueden impactar en el rendimiento y la viabilidad a largo plazo de una aplicación. Las características de cada tipo de sistema ofrecen oportunidades y desafíos únicos, lo que subraya la importancia de un análisis exhaustivo de cada situación para obtener resultados óptimos durante el desarrollo de aplicaciones.

2. TIPOS DE BASES DE DATOS NOSQL

Las bases de datos NoSQL se dividen en varios tipos según su estructura y su utilización en la gestión y recuperación de información. Cada tipo está diseñado para abordar diversas necesidades en la manipulación de grandes volúmenes de datos, así como en escalabilidad y flexibilidad. Este enfoque permite optimizar el rendimiento en aplicaciones que requieren accesos rápidos y eficientes a extensos conjuntos de datos.

Entre los tipos más comunes se encuentran **las bases de datos documentales**, que almacenan información en documentos estructurados, generalmente en formatos como JSON o XML. Esta disposición facilita el manejo de datos semi-estructurados, haciendo más sencillo gestionar información diversa y cambiante.

- **Las bases de datos de columnas** organizan los datos en columnas y no en filas, lo cual optimiza la consulta y la compresión. Esta arquitectura resulta eficiente para manejar grandes volúmenes de información en entornos distribuidos que requieren rápidas operaciones de lectura y escritura. Son habituales en aplicaciones analíticas y sistemas que procesan datos en tiempo real.
- **Las bases de datos key-value** almacenan datos en pares de clave y valor, permitiendo accesos rápidos basados en las claves. Este modelo es simple y eficaz, ideal para aplicaciones que necesitan alta disponibilidad y escalabilidad, como las plataformas de juegos, el manejo de sesiones de usuario y el almacenamiento en caché.
- **Las bases de datos de grafos** están diseñadas para representar relaciones complejas entre los datos, utilizando nodos y aristas. Este tipo permite realizar consultas eficientes sobre las conexiones e interacciones, siendo especialmente útil en aplicaciones como redes sociales, sistemas de recomendaciones y análisis de redes.

Cada uno de estos tipos ofrece ventajas y desventajas según la naturaleza de los datos y las necesidades de la aplicación, lo que posibilita la selección de la solución más adecuada en cada situación específica.

2.1. BASES DE DATOS DOCUMENTALES

Las bases de datos documentales son un tipo de sistema que permite almacenar, recuperar y gestionar información en forma de documentos. **Estos documentos se organizan frecuentemente en formatos como JSON, BSON o XML, lo que facilita una estructura jerárquica de los datos**. Esta característica fomenta la flexibilidad en la organización de la información, ya que cada documento puede presentar diferentes atributos o esquemas, en comparación con las bases de datos relacionales que impone un esquema fijo.

Un aspecto distintivo de las bases de datos documentales es su capacidad para escalar de manera horizontal. Esta opción permite distribuir la carga de trabajo en múltiples servidores, gestionando así grandes volúmenes de datos y mejorando el rendimiento en situaciones de alta demanda. Esto

resulta beneficioso para aplicaciones que requieren adaptación continua a cambios en los requisitos de datos.

Además, estas bases de datos permiten la realización de consultas complejas y poseen soporte para la creación de índices. Esto optimiza la recuperación de información precisa y aumenta la eficiencia en la gestión de grandes cantidades de documentos. Su diseño facilita la integración con distintos lenguajes de programación y tecnologías, lo que las hace útiles en el desarrollo de aplicaciones web y móviles.

Por otro lado, **muchas de estas bases de datos ofrecen características de replicación y consistencia eventual**. Esto proporciona mayor disponibilidad de datos y resiliencia del sistema ante fallos en el servidor.

El uso de bases de datos documentales se ha generalizado gracias a su capacidad para manejar información no estructurada y semiestructurada, lo que las convierte en una opción atractiva para aplicaciones que requieren adaptabilidad en la gestión de datos.

2.1.1. Características

Las bases de datos documentales son una categoría dentro de las bases de datos NoSQL, diseñadas para almacenar y gestionar datos en formato de documentos. Utilizan representaciones en formatos como JSON (JavaScript Object Notation), BSON (Binary JSON) o XML (eXtensible Markup Language), lo que les otorga flexibilidad y usabilidad en diversas aplicaciones.

Cada documento en este tipo de base de datos puede tener su propio esquema, **permitiendo que contenga diversos tipos de datos que pueden variar de uno a otro.** *Por ejemplo, en una aplicación de gestión de proyectos, un documento puede incluir el nombre del proyecto, un conjunto de tareas, la fecha de inicio, la fecha de finalización y su estado actual. Otro documento en la misma colección puede seguir una estructura diferente para representar otro proyecto*. Esta capacidad de adaptarse a variaciones en la estructura de los datos facilita la evolución de la información conforme cambian los requerimientos del sistema.

Una característica distintiva de las bases de datos documentales es su capacidad para operar sin un esquema rígido. Esto significa que los desarrolladores pueden realizar ajustes dinámicos en la aplicación, como agregar un nuevo campo para permitir etiquetas en los documentos de un proyecto. Esta flexibilidad resulta útil en entornos de desarrollo ágil, donde los equipos deben adaptarse a cambios frecuentes.

En términos de escalabilidad, estas bases de datos tienen la capacidad de escalar de manera horizontal. Esto implica que, en lugar de aumentar la capacidad de un único servidor, se pueden añadir múltiples servidores para distribuir la carga de trabajo. Este enfoque es especialmente útil en aplicaciones que experimentan un crecimiento rápido en usuarios y volúmenes de datos. Por ejemplo, un servicio de streaming de video en rápida expansión puede generar grandes cantidades

de datos, y las bases de datos documentales pueden manejar esta carga añadiendo más servidores sin comprometer el rendimiento.

El sistema de indexación en bases de datos documentales permite la creación de índices sobre campos específicos dentro de los documentos, lo que mejora la eficiencia en las consultas.

> En un sistema de gestión de clientes, si es necesario buscar rápidamente clientes por su dirección de correo electrónico, se puede establecer un índice sobre ese campo, lo que posibilita búsquedas más rápidas en comparación con el escaneo completo de la colección.

Aunque las bases de datos documentales permiten la creación de índices con facilidad, es importante gestionar adecuadamente estos índices. Tener demasiados índices (al igual que en las Bases de Datos de modelo estructurado) puede afectar el rendimiento en procesos de inserción y actualización, lo que podría ser un inconveniente en aplicaciones que demandan frecuentemente la escritura de datos.

Las operaciones de escritura en este tipo de bases de datos son eficientes gracias a su diseño optimizado para la inserción de documentos completos. Esto resulta útil en aplicaciones que necesitan registrar eventos o actividades de forma continua. Por ejemplo, una plataforma de redes sociales puede registrar interacciones de los usuarios –como publicaciones, comentarios y reacciones– utilizando un enfoque donde cada interacción se representa como un documento independiente en la base de datos. Esto asegura que el rendimiento no se vea perjudicado por la carga generada por nuevos datos.

También es relevante la capacidad de gestionar datos semi-estructurados y no estructurados. Aplicaciones que trabajan con una variedad de formatos de contenido, como archivos multimedia y documentos PDF, se benefician de esta habilidad. En un sistema de gestión de archivos para una biblioteca digital, cada libro se podría almacenar como un documento, incluyendo metadatos como título, autor y un enlace al archivo, con la posibilidad de añadir reseñas y calificaciones de los usuarios.

En el ámbito de nuevas empresas y compañías en crecimiento, las bases de datos documentales son elegidas por su flexibilidad y rapidez en la implementación. Estas organizaciones, que desarrollan aplicaciones móviles y web, encuentran en esta tecnología una solución que permite adaptarse de manera ágil a nuevas necesidades y requisitos, lo que puede ser determinante en un mercado competitivo.

Las características de las bases de datos documentales aportan valor en entornos donde la adaptabilidad y la eficiencia en la gestión de datos, así como la capacidad para realizar análisis en tiempo real, son necesarios para el desarrollo eficaz de aplicaciones y servicios modernos.

2.1.2. Modelo de datos

El modelo de datos en bases de datos documentales NoSQL **se basa en la representación de la información a través de documentos**, lo que permite una considerable flexibilidad en la estructura de los datos. **Este enfoque contrasta con las bases de datos relacionales, que organizan la información en tablas con un esquema rígido**. A continuación, se detallan las características y aplicaciones del modelo de datos en bases de datos documentales:

Una característica notable del modelo de datos en sistemas documentales es la capacidad para manejar datos no estructurados y semiestructurados. En este formato, cada documento puede considerarse como una representación autónoma de un objeto o entidad, permitiendo que contenga diferentes campos y tipos de datos. *Por ejemplo, en un sistema de registro de eventos, un documento que representa un evento podría incluir campos como el nombre del evento, la fecha, la ubicación, los participantes y la agenda, mientras que otro documento que represente a un participante podría incluir su nombre, rol y detalles de contacto*.

Los documentos pueden tener estructuras jerárquicas, lo que facilita la inclusión de información anidada. Un ejemplo práctico se puede observar en aplicaciones de gestión de recursos humanos. Un documento de empleado puede registrar no solo los datos personales, sino también una lista con habilidades, historial laboral y referencias, como se muestra a continuación:

```
{
    "employeeId": "E1234",
    "name": "Juan Pérez",
    "position": "Desarrollador",
    "skills": [
        {
            "skill": "Java",
            "level": "Avanzado"
        },
        {
            "skill": "JavaScript",
            "level": "Intermedio"
        }
    ],
    "workHistory": [
        {
            "company": "XYZ S.A.",
            "position": "Programador Junior",
            "years": 2
        },
        {
            "company": "ABC Ltda.",
            "position": "Desarrollador",
            "years": 3
        }
    ],
    "contacts": {
        "email": "juan.perez@example.com",
        "phone": "123-456-7890"
    }
}
```

Este tipo de modelo permite almacenar información compleja de manera organizada y accesible. **Al concentrar toda la información relacionada en un solo documento, se simplifican las consultas y se integra fácilmente la diversidad de campos necesarios para diversas operaciones**.

Las bases de datos documentales como MongoDB y Couchbase utilizan un lenguaje de consulta basado en documentos, lo que permite realizar filtrados, agregaciones y manipulaciones directamente sobre los documentos sin la necesidad de complejos joins. Esto se traduce en un acceso más eficiente y en una interacción más comprensible con las estructuras de datos. Por ejemplo, para obtener todos los empleados de un departamento específico, la consulta se puede estructurar como sigue:

db.employees.find({ "department": "Desarrollo" });

Este tipo de consulta simple ilustra el acceso directo a los documentos, en consonancia con el modelo de datos NoSQL.

Los casos de uso más comunes incluyen sistemas de gestión de contenido (CMS). Estos sistemas se benefician de la flexibilidad al permitir que diferentes tipos de contenido (artículos, videos, imágenes) tengan estructuras diversas y sean fácilmente gestionados sin requerir alteraciones en la base de datos. Por ejemplo, un artículo podría incorporar metadatos específicos como el autor, las categorías y las etiquetas que pueden no ser relevantes para otros tipos de contenido en el sistema.

Un ejemplo significativo se observa en aplicaciones de e-commerce. En este contexto, cada producto puede ser representado como un documento que contiene información como precio, descripción, imágenes y atributos adicionales, como tallas y colores. Este enfoque facilita la adición de nuevos productos sin complicaciones estructurales. El documento de un producto podría verse así:

```
{
    "productId": "P12345",
    "name": "Camiseta de algodón",
    "price": 19.99,
    "description": "Camiseta de algodón 100% con diseño moderno.",
    "availableSizes": ["S", "M", "L", "XL"],
    "colors": ["Rojo", "Azul", "Verde"],
    "reviews": [
        {
            "user": "usuario1",
            "rating": 5,
            "comment": "Excelente calidad."
        },
        {
            "user": "usuario2",
            "rating": 4,
            "comment": "Buen producto, aunque un poco caro."
        }
    ]
}
```

Este modelo permite que las empresas de e-commerce escalen rápidamente y respondan a las tendencias del mercado a través de la inclusión de nuevos atributos en sus productos sin necesidad de cambios en la estructura de la base de datos.

El modelo de datos en bases de datos documentales también proporciona eficiencia en situaciones donde la disponibilidad y la escalabilidad son prioritarias. La arquitectura de clústeres que utilizan las bases de datos documentales permite adicionar nodos de forma sencilla para gestionar un aumento en la carga de trabajo sin afectar el rendimiento. Este diseño es especialmente útil en aplicaciones que manejan grandes volúmenes de datos, como redes sociales y plataformas de comercio electrónico. **Los documentos pueden ser distribuidos entre diferentes nodos, permitiendo que la base de datos se escale horizontalmente mientras se mantiene la integridad y el acceso a los datos**.

Las cualidades del modelo de datos en bases de datos documentales resultan ventajosas también para la integración con tecnologías de análisis de datos y aprendizaje automático. Muchos modelos involucrados en el análisis pueden beneficiarse de la naturaleza flexible de los documentos. La facilidad de extracción de datos permite a los analistas realizar consultas ad hoc, explorar patrones en el uso y aplicar técnicas de aprendizaje automático para personalizar experiencias de los usuarios.

En el área del análisis de datos, se pueden identificar tendencias en la información de los usuarios mediante documentos que contengan sus interacciones. Por ejemplo, en una aplicación de streaming de música, cada usuario puede tener un documento que almacene canciones escuchadas, listas de reproducción y preferencias. Estos datos se pueden utilizar para proveer recomendaciones ajustadas al comportamiento de escucha.

El modelo de datos en bases de datos documentales NoSQL presenta ventajas para la gestión de datos en entornos que requieren adaptabilidad y evolución. Con la capacidad de ajustarse rápidamente a cambios en los requisitos comerciales y la facilidad para gestionar datos complejos, se presenta como una solución efectiva para una variedad de aplicaciones en distintos sectores que buscan eficiencia y flexibilidad en su arquitectura de datos.

2.2. BASES DE DATOS DE COLUMNAS

Las bases de datos de columnas representan un tipo de sistema de gestión de bases de datos NoSQL que organiza los datos en columnas en lugar de en filas, como se observa en las bases de datos relacionales convencionales. Este método permite una mayor eficacia en la consulta y el análisis de la información, lo que resulta especialmente relevante en situaciones que requieren un acceso ágil a grandes volúmenes de datos. **Estas bases de datos han sido concebidas para gestionar extensas cantidades de información distribuidas en múltiples servidores, favoreciendo así la escalabilidad horizontal**.

Una característica significativa de las bases de datos de columnas es su habilidad para optimizar el rendimiento en operaciones que demandan abundantes lecturas. Al almacenar los datos en columnas, es posible leer únicamente aquellas que son necesarias para realizar una consulta, evitando la carga de información que no será utilizada. Este tipo de almacenamiento también brinda la posibilidad de una compresión más efectiva de los datos, lo que reduce el espacio requerido y aumenta la velocidad de acceso.

Las bases de datos de columnas resultan especialmente ventajosas en aplicaciones de análisis de datos y en aquellos escenarios donde los patrones de acceso a la información se centran más en ciertas columnas que en otras. Un ejemplo de esto se puede observar en las aplicaciones de inteligencia empresarial, donde se requieren consultas sobre un subconjunto de columnas dentro de una extensa tabla, como registros de ventas o información de usuarios.

Algunas de las soluciones de bases de datos de columnas más reconocidas incluyen Apache Cassandra, HBase y Google Bigtable. Cada una de estas opciones presenta características particulares que las hacen aptas para diferentes aplicaciones, desde la gestión de grandes volúmenes de datos hasta el soporte para transacciones más complejas. La selección de una base de datos de columnas dependerá de la naturaleza de los datos a administrar y de los requisitos específicos de rendimiento y escalabilidad de la aplicación.

2.2.1. Características

Las bases de datos NoSQL de columnas operan bajo un modelo que agrupa los datos en columnas en lugar de utilizar el enfoque tradicional de filas. Este diseño permite una manipulación de datos más flexible y escalable, ofreciendo ventajas en escenarios donde se manejan grandes volúmenes de información.

Almacenar datos en columnas facilita la lectura selectiva. En aplicaciones donde las consultas requieren acceso a solo unas pocas columnas dentro de conjuntos de datos extensos, la estructura de las bases de datos de columnas optimiza el rendimiento. Por ejemplo, en sistemas de monitoreo de rendimiento de aplicaciones que recopilan métricas de numerosos parámetros, se pueden realizar consultas que acceden exclusivamente a las columnas relacionadas con el rendimiento, evitando la lectura de datos no pertinentes.

Una característica distintiva de las bases de datos de columnas es su capacidad para gestionar un esquema flexible. Las bases de datos relacionales requieren definiciones rígidas, lo que complica la adaptación a cambios en los requisitos de datos. Con las bases de datos de columnas, los desarrolladores tienen la posibilidad de añadir o eliminar columnas fácilmente, lo que resulta útil para proyectos donde los requisitos evolucionan con el tiempo. Por ejemplo, en el desarrollo de aplicaciones móviles, los datos de usuario pueden variar considerablemente entre diferentes versiones de la aplicación; con una base de datos de columnas, es posible implementar cambios en el modelo de datos sin realizar migraciones difíciles.

En términos de almacenamiento, las bases de datos de columnas implementan técnicas de compresión avanzadas. El almacenamiento en columnas permite guardar datos similares en grupos, lo que conlleva una compresión más eficaz en comparación con el almacenamiento en filas. Esto tiene relevancia en entornos de big data, donde las organizaciones buscan optimizar costos de almacenamiento sin sacrificar la velocidad de acceso. *Un ejemplo práctico se observa en plataformas de análisis de datos como Google **BigQuery**, que usa un formato de almacenamiento que se beneficia del enfoque de columnas para reducir el costo de las consultas sobre grandes volúmenes de información*.

La capacidad de realizar operaciones de agregación sobre grandes conjuntos de datos es una característica principal de las bases de datos de columnas. Muchas aplicaciones empresariales requieren análisis y reportes sobre datos, donde es necesario obtener estadísticas o información de grandes volúmenes. Por ejemplo, en el análisis de tendencias de ventas de una cadena de comercio, se podría utilizar una base de datos de columnas para calcular rápidamente las ventas promedio por región, producto o periodo de tiempo, mejorando la eficiencia en la toma de decisiones.

Entre los ejemplos concretos de bases de datos de columnas destaca **Apache Cassandra**, conocido por su alta disponibilidad y escalabilidad. Varias empresas del sector tecnológico utilizan Cassandra para satisfacer necesidades crecientes de datos. *Un caso notable es el de Instagram, que utiliza Cassandra para almacenar y acceder a datos de fotos, comentarios y usuarios, garantizando baja latencia y alta disponibilidad*, aspectos importantes para ofrecer una experiencia de usuario fluida.

***HBase** también es un sistema notable empleado en el ecosistema de **Hadoop**, que permite a las organizaciones realizar análisis en tiempo real sobre grandes volúmenes de datos*. En el caso de las empresas de telecomunicaciones, HBase puede servir para almacenar registros de llamadas y mensajes, facilitando la búsqueda y recuperación de datos de forma instantánea para informes y análisis de tráfico.

Las bases de datos de columnas son adecuadas para aplicaciones de redes sociales. Estas plataformas generan enormes volúmenes de datos en forma de publicaciones, comentarios e interacciones, que requieren un enfoque de almacenamiento eficiente. Utilizando bases de datos de columnas, es posible realizar análisis sobre patrones de comportamiento de los usuarios, optimizando el contenido presentado y mejorando la personalización de la experiencia de navegación.

Un dominio emergente donde se están implementando las bases de datos de columnas es el sector financiero. En cuanto a la realización de análisis financieros, donde se procesa la información de transacciones y se generan informes de manera rápida, estas bases de datos permiten el acceso veloz a columnas específicas como fechas, montos y tipos de transacción. Esto permite, por ejemplo, la detección de fraudes en tiempo real, donde se aplican algoritmos de aprendizaje automático que analizan transacciones simultáneamente.

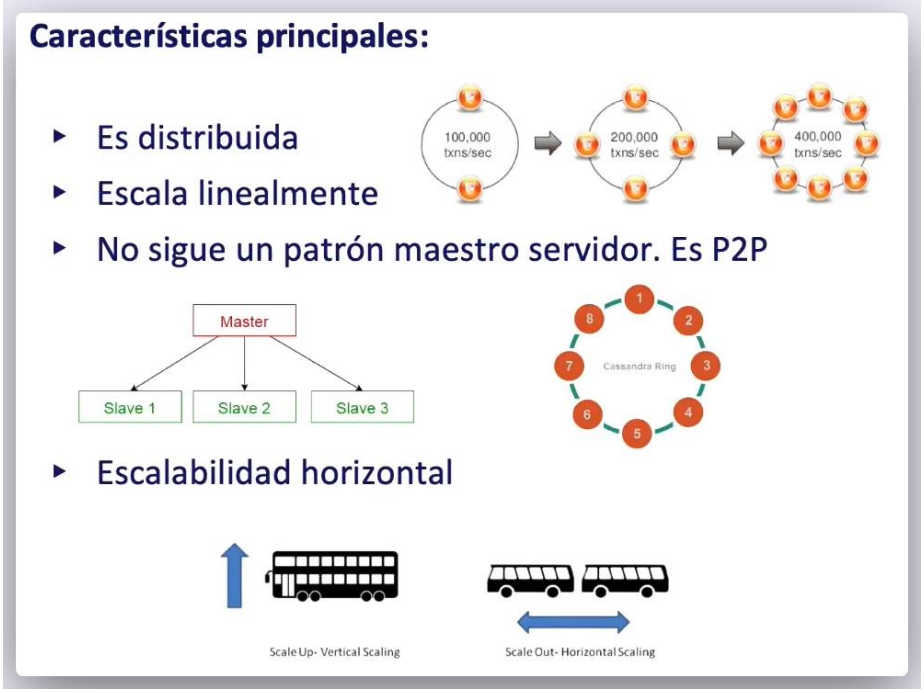
La integración de datos en tiempo real es otra capacidad significativa que proporcionan las bases de datos de columnas. Muchas aplicaciones modernas requieren que los datos sean recopilados y analizados de forma instantánea para ofrecer resultados inmediatos. En el comercio electrónico, las bases de datos de columnas ayudan a personalizar recomendaciones de productos basándose en el comportamiento de navegación y compra del usuario, actuando de manera dinámica con la información disponible.

El manejo eficiente de datos estructurados y no estructurados encuentra un buen ajuste en las bases de datos de columnas. Este modelo es apropiado para las organizaciones que gestionan datos de diferentes formatos y estructuras, facilitando el acceso rápido a la información necesaria para permitir decisiones estratégicas. En el marketing digital, se pueden almacenar datos de campañas publicitarias en columnas, lo que permite evaluar su efectividad comparando métricas clave como tasas de clics, impresiones y conversiones en diferentes períodos y segmentos de público.

Las bases de datos de columnas son rentables en entornos donde el volumen de datos es alto, ya que el costo de almacenamiento puede ser un aspecto relevante. El modelo de datos diseñado para optimizar consultas sobre columnas particulares es especialmente útil en compañías que necesitan llevar a cabo análisis regulares o generar informes. Esto se observa en industrias como la del entretenimiento, *donde plataformas de streaming tales como Spotify optimizan su almacenamiento utilizando bases de datos de columnas para gestionar catálogos masivos* de música, permitiendo así un análisis de tendencias en la escucha de usuarios de manera efectiva.

El diseño y las características de las bases de datos NoSQL de columnas proporcionan soluciones efectivas para manejar grandes volúmenes de datos en diversos sectores, optimizando tanto el almacenamiento como la recuperación de información y permitiendo una alta adaptabilidad a los cambios que surgen en el ámbito tecnológico y en las aplicaciones comerciales.

2.2.2. Modelo de datos



Modelo de Datos: Apache Cassandra

El modelo de datos en bases de datos de columnas se estructura de manera que cada fila puede contener un número variable de columnas. Esta característica permite una mayor flexibilidad en la representación de información.

> Los datos se organizan en familias de columnas, donde cada familia agrupa múltiples columnas que representan diferentes atributos de una misma entidad. Este enfoque facilita el manejo de datos semiestructurados y no estructurados de manera efectiva.

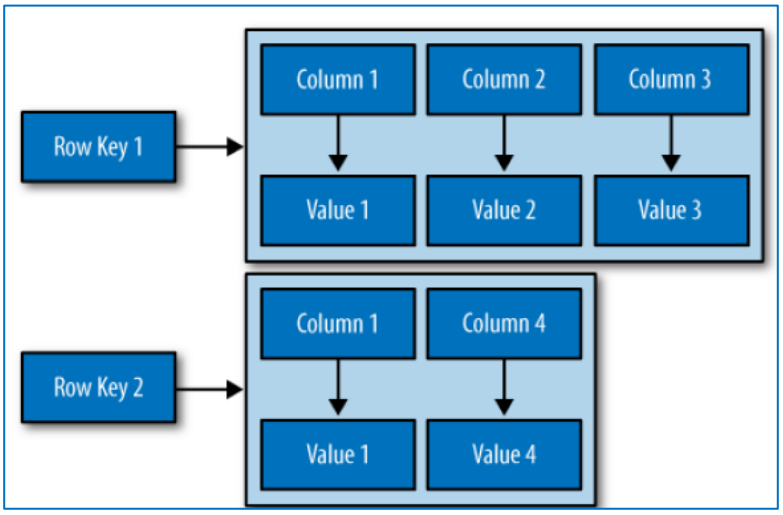
Las familias de columnas son una característica central en este tipo de bases de datos. En Apache Cassandra, por ejemplo, cada familia está diseñada para almacenar un conjunto de datos que suelen aparecer juntos, mejorando la eficiencia de las consultas. En una aplicación de seguimiento de ventas, una familia de columnas podría almacenar registros de transacciones, donde cada fila representa una venta y las columnas incluyen atributos como la identificación del producto, la cantidad vendida, el precio, la fecha y cualquier descuento aplicado. Esta capacidad de almacenar diferentes atributos en columnas específicas permite realizar análisis detallados sobre las ventas.

El modelo de datos de columnas se adapta bien a la necesidad de escalabilidad horizontal, importante en entornos donde el volumen de datos puede crecer rápidamente. Estas bases de datos distribuyen la información en múltiples nodos de un clúster, lo que permite equilibrar la carga y gestionar un acceso concurrente a los datos. Esto resulta relevante en aplicaciones como plataformas de redes sociales, donde miles de usuarios interactúan al mismo tiempo. Al almacenar

la información de cada usuario en filas de una familia de columnas, los sistemas pueden consultar eficientemente diversos aspectos de los perfiles y sus interacciones.

Otro uso se encuentra en el análisis financiero. En este ámbito, las bases de datos de columnas pueden almacenar datos históricos de transacciones, donde cada fila representa una transacción única y las columnas incluyen detalles como el monto, la fecha, el tipo de transacción y el estado. Esta estructura permite a las instituciones financieras realizar análisis de tendencias y generar informes de auditoría de modo más eficiente.

La capacidad para gestionar datos que cambian frecuentemente y que pueden no ajustarse a un esquema rígido es una característica notable de este modelo. En sistemas de gestión de contenido (CMS), por ejemplo, diferentes tipos de contenido pueden tener diversos atributos, y las bases de datos de columnas permiten que las páginas almacenen solo los elementos relevantes para cada tipo. En este caso, una entrada de blog podría tener columnas para el autor, la fecha de publicación y el texto del post, mientras que una página de producto incluiría columnas para el nombre, el precio y la descripción. Este enfoque facilita la gestión de contenido variado sin necesidad de redefinir un esquema cada vez que se añade un nuevo tipo de información.



El diseño de los esquemas también influye en la estructura del modelo de datos. **Al crear familias de columnas, es importante considerar cómo se enlistarán los datos según las consultas previstas**. Esto requiere una planificación cuidadosa para asegurar que las consultas sean eficientes. *Por ejemplo, si se requiere un informe sobre la actividad de usuarios en tiempo real, se puede diseñar un esquema que agrupe acciones recientes de los usuarios en una familia de columnas, lo que permite un acceso rápido a esta información*.

Respecto al rendimiento, las bases de datos de columnas frecuentemente implementan técnicas de compresión. La compresión de datos es especialmente efectiva debido a la naturaleza homogénea de las columnas en una familia, lo que permite tasas de compresión más altas en comparación con estructuras más heterogéneas. Por ejemplo, en sistemas que gestionan grandes

volúmenes de datos de sensores en el Internet de las Cosas (IoT), los datos de diferentes dispositivos pueden ser almacenados en familias de columnas específicas, donde se aplica compresión para reducir el espacio de almacenamiento necesario.

Un ejemplo concreto se encuentra en el ámbito del entretenimiento, como en Netflix. Esta plataforma utiliza bases de datos de columnas para gestionar grandes volúmenes de datos de usuarios, incluidos interacciones, preferencias de visualización y hábitos de consumo. Cada fila puede representar a un usuario con columnas para sus preferencias de género, historial de visualización y calificaciones de películas. Esta estructura permite la realización de análisis sobre tendencias de visualización que pueden guiar decisiones sobre la producción de nuevo contenido.

Finalmente, el modelo de datos en bases de datos de columnas permite realizar operaciones rápidas de lectura y escritura. Dado que cada columna es accesible individualmente, los sistemas pueden optimizar el acceso a datos específicos sin necesidad de cargar toda la fila. Este enfoque resulta útil en situaciones donde solo un subconjunto de datos es relevante para una consulta puntual. Por ejemplo, en un sistema de gestión de inventario, una consulta que verifica la existencia de un producto puede acceder directamente a la columna correspondiente, evitando la búsqueda en filas completas.

Estos aspectos del modelo de datos en bases de datos de columnas permiten a las organizaciones aprovechar las capacidades de almacenamiento y procesamiento de datos en situaciones donde se requieren rapidez y flexibilidad.

2.3. BASES DE DATOS KEY-VALUE

Las bases de datos Key-Value son sistemas de gestión de datos que organizan la información en pares de clave y valor. **Cada clave es única y se utiliza para acceder a su valor correspondiente, que puede ser de cualquier tipo, incluyendo cadenas de texto, números u objetos complejos**. Este modelo se caracteriza por su simplicidad y es común en aplicaciones que requieren operaciones rápidas y eficientes.

Un aspecto significativo de las bases de datos Key-Value es su alta escalabilidad, lo que permite gestionar grandes volúmenes de información a través de una distribución eficiente de los datos. Esto resulta beneficioso en entornos donde el rendimiento y la disponibilidad son importantes, adaptándose a diferentes demandas de carga de trabajo.

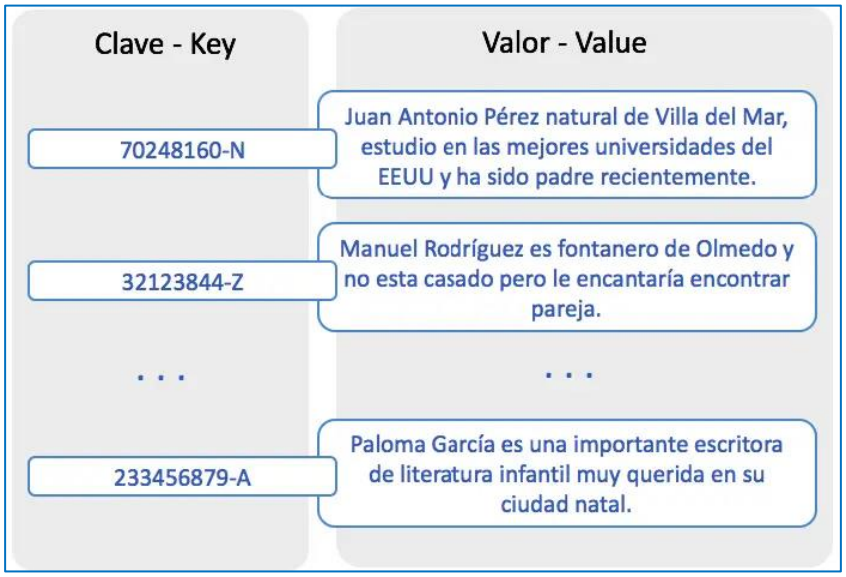
Adicionalmente, **estas bases de datos suelen ofrecer una velocidad notable en las operaciones de lectura y escritura**, lo cual se traduce en un rendimiento óptimo para aplicaciones que necesitan acceder a datos de manera rápida. Por esto, son utilizadas frecuentemente en aplicaciones web y móviles, así como en sistemas de gestión de sesiones y almacenamiento de datos temporales.

La flexibilidad es otro beneficio de este tipo de bases de datos. No imponen un esquema rígido, permitiendo almacenar datos de diversas estructuras sin necesidad de definiciones previas. Esto

facilita la evolución y adaptación de la base de datos a nuevas necesidades y cambios en los requisitos de la aplicación.

Las operaciones en bases de datos Key-Value son en su mayoría simples, limitándose a añadir, modificar, eliminar y buscar pares de clave y valor. Esta característica contribuye a su agilidad y efectividad, siendo idóneas para situaciones donde las relaciones complejas entre los datos no son requeridas.

2.3.1. Características



Las bases de datos Key-Value se organizan en pares de clave y valor, un modelo que permite un acceso directo y rápido a la información. **La clave sirve como un identificador único**, lo cual facilita la localización de los datos asociados en forma de valor. Esta relación entre clave y valor permite realizar operaciones de obtención, modificación y eliminación de datos de manera eficiente y con tiempos de respuesta reducidos.

Una característica notable de las bases de datos Key-Value es su estructura flexible. A diferencia de las bases de datos relacionales, que exigen un esquema preestablecido, las bases de datos Key-Value posibilitan el almacenamiento de valores con diferentes formatos dependiendo de la clave. Esta flexibilidad simplifica la gestión de datos diversos y permite la incorporación de nuevos tipos de datos con mínimo esfuerzo. *Por ejemplo, en una base de datos Key-Value se pueden guardar configuraciones de usuario en un formato JSON, imágenes en formato binario, y datos de uso como texto, todo accesible a través de claves únicas*.

El rendimiento adecuado es otro atributo de estas bases de datos. Al estar diseñadas para operaciones de lectura y escritura ágiles, son apropiadas para aplicaciones que requieren acceso en tiempo real a grandes volúmenes de datos. Este rendimiento se puede observar en entornos de

análisis de datos, donde se puede utilizar una base de datos Key-Value para almacenar resultados intermedios y acceder a ellos de forma rápida durante el procesamiento.

Un ejemplo en el ámbito del comercio electrónico es la organización de catálogos de productos en bases de datos Key-Value. Aquí, el ID del producto puede actuar como la clave, mientras que el valor es un objeto que incluye información como el nombre, la descripción, el precio y la disponibilidad. Este formato permite que las aplicaciones accedan de forma ágil a los detalles del producto y respondan eficazmente a las consultas.

Las plataformas de mensajería instantánea también utilizan bases de datos Key-Value. Una aplicación de chat puede almacenar conversaciones con una clave que represente a los interlocutores y un valor que contenga una lista de mensajes intercambiados. Esto permite un acceso eficiente a los mensajes al utilizar la clave correspondiente.

Las bases de datos Key-Value son útiles en la administración de configuraciones para diversas aplicaciones. Muchas de ellas requieren configuraciones que pueden cambiar según el entorno de ejecución. Almacenar estas configuraciones en una base de datos permite que se modifiquen de forma sencilla mediante una clave que represente la configuración y el valor que contenga los detalles. Esto proporciona un modo fácil de ajustar parámetros operativos sin necesidad de alterar el código o la estructura principal de almacenamiento.

En términos de tecnología, Redis se destaca en situaciones que demandan alta disponibilidad y rendimiento. *Por ejemplo, una aplicación que ofrece pronósticos del clima y que necesita actualizaciones frecuentes puede utilizar Redis para almacenar de manera temporal las predicciones, lo que mejora la velocidad de respuesta ante las consultas de los usuarios*.

Amazon DynamoDB, por su parte, es una opción administrada que ofrece funcionalidades de escalabilidad automática y recuperación ante fallos. Esto resulta útil en aplicaciones que enfrentan fluctuaciones significativas en la carga, como plataformas de distribución de contenido donde la demanda puede variar rápidamente. En este caso, cada usuario puede tener un identificador único, lo que permite que su historial de visualización sea almacenado como un valor. Esto facilita la personalización del contenido ofrecido de manera eficiente.

Las bases de datos Key-Value son una solución adecuada para una amplia variedad de aplicaciones en múltiples sectores gracias a su rapidez, flexibilidad y capacidad para gestionar datos que no requieren una estructura rígida. Su adaptabilidad a distintos casos de uso las convierte en una elección relevante en el presente desarrollo de software.

2.3.2. Modelo de datos

El modelo de datos en bases de datos NoSQL, específicamente en las bases de datos tipo Key-Value, opera mediante una estructura que relaciona cada clave (key) con un valor (value). Esta forma de organización permite un almacenamiento y recuperación de datos eficientes, siendo particularmente útil en situaciones donde son prioritarios la escalabilidad y la alta disponibilidad.

2.3.2.1. Estructura del modelo de datos

En este modelo, cada entrada consiste en una clave única que identifica un valor específico. La clave puede ser un identificador que se elabora a partir de un *string*, un entero o incluso un UUID (Identificador Único Universal). **El valor puede ser un tipo de dato simple, pero también puede ser un objeto complejo, como un JSON** o cualquier otra estructura que contenga múltiples atributos.

> En un sistema de gestión de bibliotecas, las claves podrían representar los identificadores de los libros, como "ISBN:978-3-16-148410-0", mientras que los valores podrían ser objetos JSON que incluyan información del libro, como el título, autor, año de publicación y disponibilidad.

2.3.2.2. Ventajas de usar bases de datos key-value

Las bases de datos Key-Value presentan varias ventajas a considerar en términos de rendimiento y escalabilidad.

- **Escalabilidad Horizontal**: A medida que una aplicación crece, se pueden añadir más nodos a la base de datos para soportar mayores cargas de trabajo. Esta característica es particularmente útil en sistemas donde la demanda puede variar considerablemente.
- **Acceso Rápido a Datos**: La estructura simple de clave y valor permite un acceso eficiente a la información. La recuperación de un valor asociado a una clave específica implica una operación sencilla de consulta que se ejecuta de forma casi instantánea.
- **Flexibilidad**: Este modelo no requiere un esquema rígido. Por lo tanto, los desarrolladores tienen la capacidad de modificar los datos y agregar atributos nuevos sin preocuparse por afectar la estructura de la base de datos existente.

Un ejemplo donde se aplica esto podría ser en una aplicación de comercio electrónico, donde se almacenan los detalles de los productos con claves como "product:101", y los valores incluyen información como el nombre, precio, imagen y descripción. Ante un cambio en el precio, solo se necesitaría actualizar ese campo dentro del objeto, sin necesidad de alteraciones en la estructura general de la base de datos.

2.3.2.3. Casos de uso en aplicaciones

Los escenarios de uso para bases de datos Key-Value son diversos:

- **Gestión de Sesiones de Usuario**: En aplicaciones web, las sesiones son importantes para mantener la continuidad en las interacciones. Estas bases de datos se pueden utilizar para almacenar el estado de la sesión de un usuario usando claves que representan el ID de la sesión. Cada valor puede contener información como el usuario autenticado, el tiempo de expiración de la sesión y las preferencias del usuario, facilitando así un acceso eficiente a los datos relacionados.

- **Sistema de Cache**: Muchas aplicaciones utilizan una base de datos Key-Value para el almacenamiento en cache de resultados de consultas frecuentes. Por ejemplo, en un sitio web de noticias, artículos que han sido leídos recientemente pueden almacenarse como "article:latest", permitiendo que los usuarios obtengan esos datos rápidamente al volver a visitar la página.
- **Almacenamiento de Información Temporal o Metadatos**: Este modelo es adecuado para almacenar datos temporales o metadatos relacionados con otras bases de datos. En un sistema de gestión de tareas, por ejemplo, podría utilizarse una base de datos Key-Value para guardar información sobre el estado de las tareas, donde la clave representa el ID de la tarea y el valor es un objeto que define el estado actual, el tiempo estimado de finalización y los recursos asignados.

2.3.2.4. Consideraciones sobre consultas complejas

A pesar de sus beneficios, las bases de datos Key-Value presentan limitaciones en la ejecución de consultas complejas. A diferencia de las bases de datos relacionales, que permiten realizar uniones y filtrados en múltiples tablas, en un sistema Key-Value, la lógica de la aplicación maneja relaciones a través de programación, lo que puede aumentar la carga computacional.

Por ejemplo, si se necesita consultar información relacionada entre múltiples entidades, como usuarios y sus compras, la aplicación debe ejecutar varias consultas hacia la base de datos y combinar manualmente los datos obtenidos. Esta aproximación difiere de las bases de datos relacionales, donde tales operaciones son más eficientes por su diseño estructurado.

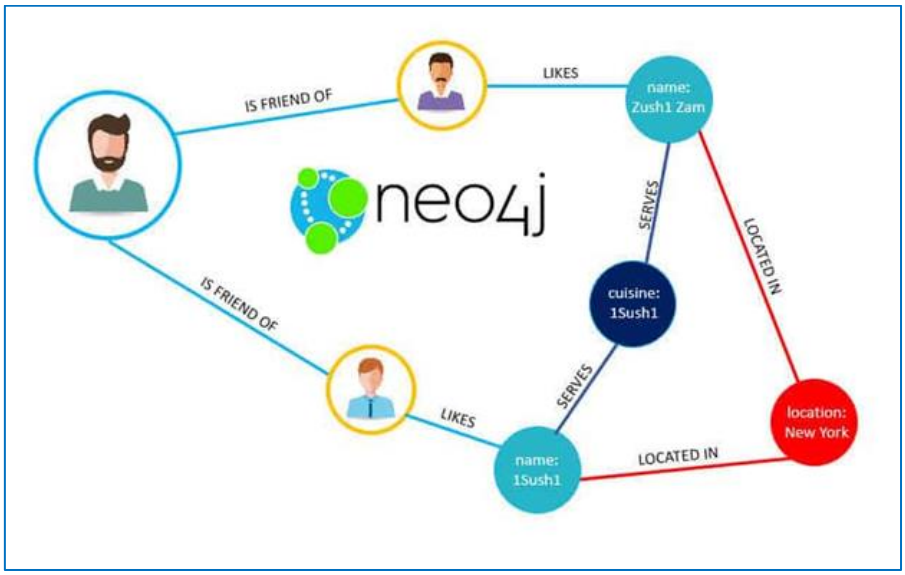
2.3.2.5. Ejemplos de bases de datos key-value

Existen diversas tecnologías de bases de datos que han adoptado el modelo Key-Value en su diseño.

- **Redis**: Esta es una de las bases de datos más populares en este ámbito. Permite el almacenamiento de pares clave-valor y soporta estructuras de datos más complejas como listas, conjuntos y hashes. Redis se utiliza frecuentemente para aplicaciones que requieren alta velocidad y baja latencia, como sistemas de mensajería o gestión de usuarios.
- **Amazon DynamoDB**: Se trata de una base de datos administrada en la nube que ofrece un rendimiento estable y escalabilidad automática. DynamoDB permite almacenar datos en forma de documentos y proporciona características de consulta que, aunque limitadas en comparación con las bases de datos relacionales, ofrecen flexibilidad para que los desarrolladores operen en entornos dinámicos.

El uso del modelo de datos Key-Value en estos sistemas permite a los desarrolladores construir aplicaciones que responden con agilidad a variaciones en la carga y en las estructuras de datos, reflejando la adaptabilidad del modelo en el ámbito del almacenamiento de información.

2.4. BASES DE DATOS DE GRAFOS



Las bases de datos de grafos son sistemas de gestión de bases de datos desarrollados para almacenar y consultar datos interconectados mediante una estructura compuesta por nodos, aristas y propiedades. Los nodos simbolizan entidades, las aristas representan las relaciones entre estas, y las propiedades son características que describen tanto a nodos como a aristas. Este enfoque permite modelar interacciones complejas de manera más eficiente que las bases de datos relacionales convencionales.

Una característica distintiva de estas bases de datos es su habilidad para manejar consultas que involucran caminos entre nodos con mayor eficiencia. Esto resulta en un rendimiento óptimo para actividades que requieren explorar rutas y conexiones, como en redes sociales, análisis de redes de transporte y sistemas de recomendación. Las consultas en este tipo de sistemas suelen ser menos costosas en recursos, especialmente al trabajar con grandes conjuntos de datos muy interrelacionados.

Existen múltiples implementaciones de bases de datos de grafos, algunas centradas en el manejo óptimo de relaciones, como **Neo4j**, que utiliza un modelo de grafos escalable. Otras, como **Amazon Neptune**, están diseñadas para integrarse con servicios en la nube, facilitando el procesamiento y análisis de datos en entornos distribuidos. La elección de una base de datos de grafos en lugar de una opción convencional depende de las necesidades específicas de cada proyecto y de la profundidad exigida en las consultas sobre relaciones.

El modelo que emplean estas bases de datos se caracteriza por su flexibilidad, permitiendo esquemas dinámicos donde se pueden agregar nuevas relaciones y atributos sin modificar la estructura existente. Esta adaptabilidad resulta particularmente útil en entornos donde los datos están en constante cambio y las consultas pueden variar frecuentemente. Las bases de datos de

grafos ganan relevancia en aplicaciones contemporáneas debido a su capacidad para resolver problemas complejos relacionados con la conectividad y las interacciones entre datos.

2.4.1. Características

Las bases de datos de grafos se centran en la representación de datos mediante estructuras que incluyen nodos, aristas y propiedades, lo que permite una gestión eficiente de conexiones complejas entre entidades. Los nodos representan las entidades, que pueden ser objetos o conceptos, mientras que las aristas representan las relaciones que existen entre esos nodos. Cada nodo y cada arista pueden tener propiedades, que son pares clave-valor que describen características adicionales. Un diseño basado en grafos resulta adecuado para aplicaciones donde es importante comprender y consultar relaciones complejas.

Las consultas en bases de datos de grafos se ejecutan de manera distinta a las bases de datos relacionales. Debido a su estructura, estas bases de datos están optimizadas para realizar búsquedas en relaciones que involucran múltiples niveles y conexiones directas. Esto permite acceder a las distintas entidades y sus interrelaciones de forma rápida y efectiva. Por ejemplo, en un sistema de gestión de conocimiento, un nodo puede representar una persona, y una arista puede representar "trabaja con". Si es necesario encontrar a todos los colaboradores de una persona, la base de datos puede seguir las aristas de manera directa sin la necesidad de uniones de tablas, lo que acelera el proceso.

Una característica notable de las bases de datos de grafos es su **flexibilidad** para manejar estructuras de datos cambiantes y semi-estructuradas. A medida que emergen nuevas relaciones o entidades, los modelos en bases de datos de grafos pueden adaptarse sin la necesidad de una reestructuración que es común en los sistemas relacionales. Esto resulta relevante en aplicaciones donde los requisitos evolutivos son frecuentes, como en plataformas de redes sociales donde las interacciones entre usuarios cambian continuamente.

En el ámbito de la prevención de fraudes financieros, las bases de datos de grafos permiten un análisis de redes de transacciones. Cada nodo podría ser una cuenta bancaria o un individuo, y las aristas representarían transacciones entre ellos. Utilizando algoritmos de detección de anomalías, se pueden identificar patrones de comportamiento sospechosos, como múltiples transacciones entre las mismas cuentas dentro de un breve periodo. Este enfoque facilita la identificación de redes de fraude y el análisis de patrones de transferencia que podrían no ser evidentes en una estructura de datos tradicional.

Las aplicaciones de recomendación también se benefician de las bases de datos de grafos. En el comercio electrónico, los nodos podrían ser productos, usuarios, reseñas y características específicas de cada producto. Al analizar las conexiones, se puede descubrir que un usuario que compró un artículo determinado también mostró interés en otros productos que están conectados de manera similar. *Por ejemplo, un usuario que adquirió un libro de cocina podría recibir*

recomendaciones para utensilios de cocina o ingredientes específicos, gracias a la exploración de relaciones en el grafo.

Un uso relevante se observa en la interconexión de datos en el ámbito académico y de investigación. En una base de datos que modela investigadores, publicaciones y citas, cada investigador puede ser un nodo, cada publicación otro nodo, y cada cita entre publicaciones una arista. Esto permite rastrear la evolución de ideas en un campo de estudio, facilitando el descubrimiento de colaboraciones entre investigadores y el análisis de la influencia de un trabajo en otros. Esta representación gráfica resulta útil para entender cómo ciertas investigaciones han impactado en el avance del conocimiento científico.

Dentro de los sistemas de gestión de datos geoespaciales, las bases de datos de grafos pueden utilizarse para modelar las relaciones entre distintas localizaciones. Los nodos pueden representar puntos de interés, como tiendas, restaurantes o parques, mientras que las aristas pueden representar rutas o conexiones entre ellos. Esto permite realizar un análisis eficaz de caminos, optimización de rutas y planificación de trayectos en aplicaciones de navegación, facilitando el acceso a información geográfica dinámica.

Las bases de datos de grafos también tienen aplicaciones en el análisis de la biología computacional, donde se modelan interacciones biológicas complejas. Por ejemplo, en la investigación sobre redes de proteínas, los nodos pueden representar diferentes proteínas y las aristas las interacciones entre ellas. Este enfoque permite a los investigadores identificar interacciones que influyen en procesos celulares, lo que puede resultar útil en el desarrollo de tratamientos médicos.

La capacidad de las bases de datos de grafos para realizar análisis de red, combinando consultas directas y flexibilidad para manejar datos cambiantes, las convierte en herramientas que ofrecen soluciones en diversos sectores. Su estructura intuitiva y su rendimiento efectivo en consultas con múltiples niveles de conexión continúan promoviendo su adopción en aplicaciones que requieren una comprensión detallada de las relaciones en los datos.

2.4.2. Modelo de datos

El modelo de datos en bases de datos de grafos se caracteriza por su estructura compuesta por nodos y aristas, lo que permite representar datos interconectados de manera eficiente. A continuación, se examinan los conceptos básicos de este modelo, junto con ejemplos y casos de uso que ilustran su aplicación práctica.

2.4.2.1. Nodos y aristas

Los nodos son las entidades almacenadas en la base de datos. Cada nodo puede representar diferentes tipos de información, como personas, lugares, eventos, productos u otros objetos relevantes. Por ejemplo, en un sistema de reservas de vuelos, cada nodo podría representar una ciudad, un aeropuerto o incluso un vuelo específico.

Las aristas son las conexiones que enlazan los nodos. **Estas pueden ser unidireccionales o bidireccionales, según cómo se quiera representar la relación**. En el entorno de una red social, una arista podría reflejar una relación de amistad que es bidireccional, mientras que una relación de seguimiento en Twitter suele ser unidireccional. Asimismo, las aristas pueden incluir propiedades que describen la naturaleza de la relación, como la fecha de inicio de una amistad o la frecuencia de interacción entre usuarios.

2.4.2.2. Propiedades de nodos y aristas

Tanto los nodos como las aristas pueden tener propiedades que proporcionan información adicional relevante. Las propiedades de un nodo pueden incluir atributos como nombre, edad o ubicación. *Por ejemplo, al representar una biblioteca, un nodo correspondiente a un libro podría contar con propiedades tales como título, autor y año de publicación*.

Las aristas también pueden contener propiedades que enriquecen la consulta de datos. Usando una red social como ejemplo, una arista que conecta dos usuarios podría tener propiedades que representen la fecha de inicio de la amistad y el tiempo de interacción, como la cantidad de mensajes intercambiados.

2.4.2.3. Consultas en Bases de Datos de Grafos

Las consultas en bases de datos de grafos se realizan comúnmente mediante lenguajes como Cypher. Un ejemplo de consulta podría ser la búsqueda de todos los amigos de un usuario que compartan ciertos intereses. Usando la sintaxis de Cypher, esto podría redactarse de la siguiente manera:

```
MATCH (u:Usuario {nombre: 'Juan'})-[:AMIGO]->(amigo:Usuario)-[:INTERESADO_EN]-
>(interes:Interes {nombre: 'Fútbol'})

RETURN amigo
```

Esta consulta identifica todos los nodos que representan amigos de ‘Juan’ y que, además, tienen una relación de interés definida en ‘Fútbol’.

2.4.2.4. Casos de uso en análisis de redes

En el ámbito del análisis de redes, las bases de datos de grafos resultan útiles para mapear y analizar conexiones complejas. Un caso práctico puede incluir la visualización de redes de relaciones en un entorno de inteligencia criminal, donde cada nodo representa a un sospechoso, y las aristas indican conexiones como contactos telefónicos, correos electrónicos o relaciones familiares. Esta estructura facilita la identificación de conexiones y patrones que podrían ser difíciles de detectar en otros tipos de bases de datos.

2.4.2.5. Recomendaciones de productos en comercio electrónico

El modelo de datos de grafos también es útil para realizar recomendaciones de productos en plataformas de comercio electrónico. En este escenario, los nodos representan usuarios y productos, mientras que las aristas indican interacciones como compras o valoraciones. *Por ejemplo, se podría utilizar una consulta para identificar productos en la misma categoría que otros adquiridos por un usuario, ofreciendo así recomendaciones personalizadas:*

```
MATCH (u:Usuario {id: 123})-[:COMPRO]->(p:Producto)-[:EN_CATEGORIA]->(c:Categoria)<-[:EN_CATEGORIA]-(recomendacion:Producto)

RETURN recomendacion
```

2.4.2.6. Gestión de contenido

El modelo de datos en grafos tiene aplicaciones en la gestión de contenido en plataformas digitales. Por ejemplo, podría representarse una red de artículos, autores y categorías. Los nodos representarían artículos, autores y categorías, y las aristas conectarían estas entidades mediante relaciones como “escrito por” o “pertenece a”. Este esquema permite una navegación fluida que ayuda a los usuarios a explorar contenido relacionado eficientemente.

En sistemas de gestión de contenido, los editores pueden aprovechar estas relaciones para organizar mejor el contenido. Si un artículo sobre tecnología tiene aristas que lo conectan con artículos sobre inteligencia artificial y robótica, podrá mostrarse automáticamente una sección de “artículos relacionados” que mejorará la experiencia del usuario.

2.4.2.7. Aprendizaje automático y bases de datos de grafos

La integración entre bases de datos de grafos y técnicas de aprendizaje automático abre nuevas oportunidades en el análisis de datos. Un escenario podría incluir un grafo que represente datos de usuarios y su comportamiento, donde se implementan algoritmos que se nutren de estas relaciones para predecir comportamientos futuros. *Por ejemplo, en plataformas de streaming de video, el modelo se ajusta para sugerir series y películas basado en intereses compartidos entre usuarios y tendencias actuales.*

Estos enfoques son útiles tanto en recomendaciones de contenido como en la detección de fraudes, donde es posible identificar y analizar interacciones inusuales para prevenir actividades sospechosas.

2.4.2.8. Escalabilidad y modelado flexible

Las bases de datos de grafos son escalables y permiten a las organizaciones adaptarse rápidamente a cambios en la estructura de datos. Este aspecto resulta relevante en negocios que requieren incorporar continuamente nuevas relaciones y entidades. Gracias a la capacidad de gestionar datos

no estructurados, es factible añadir nuevos nodos y aristas sin tener que redefinir por completo un esquema, lo que facilita la evolución de la base de datos.

3. OPERACIONES CRUD EN BASES DE DATOS NOSQL

Las operaciones CRUD describen las acciones básicas que se llevan a cabo en una base de datos, siendo estas Crear, Leer, Actualizar y Eliminar información. En las bases de datos NoSQL, la forma en que se implementan estas acciones puede depender del tipo específico, ya sea que se trate de sistemas documentales, en clave-valor, en columnas anchas o en grafos.

Para la operación de *Crear*, se realiza la inserción de nuevos elementos en la base de datos. En bases de datos documentales, cada elemento puede considerarse como una colección de pares clave-valor, donde un nuevo objeto se agrega utilizando un comando que organiza y almacena los datos.

La operación de *Leer* permite recuperar información almacenada. En NoSQL, esto se puede llevar a cabo mediante la búsqueda de registros completos o extrayendo atributos específicos dentro de un registro. Según el sistema, estas consultas pueden ejecutarse a través de interfaces de programación o lenguajes propios que permiten acceder a la información de manera efectiva y flexible.

Actualizar implica modificar información ya existente. En el entorno NoSQL, este proceso puede incluir la actualización de un registro completo o de ciertos campos dentro de él. La flexibilidad de la estructura de datos facilita llevar a cabo estas modificaciones sin tener que seguir un esquema rígido, permitiendo que la base de datos se adapte a los cambios en los requerimientos relacionados con la información.

La operación de *Eliminar* se refiere a quitar elementos de la base de datos. En este caso, se puede eliminar un registro completo o deshacerse de campos específicos dentro de un registro. La capacidad de realizar eliminaciones efectivas es importante para mantener la integridad de los datos y gestionar el espacio de almacenamiento de manera eficiente.

Cada una de estas acciones puede estar influenciada por consideraciones específicas en relación con las características de las bases de datos NoSQL, tales como las opciones de replicación, distribución y escalabilidad, que pueden afectar tanto el rendimiento como la consistencia de las operaciones CRUD.

3.1. OPERACIONES CRUD

Las operaciones CRUD constituyen un conjunto de funciones que permiten manejar datos dentro de un sistema de almacenamiento, ya sea una base de datos relacional o NoSQL. El significado del acrónimo CRUD proviene de las palabras en inglés: Create, Read, Update y Delete. Estas operaciones son básicas para cualquier aplicación que necesite gestionar datos y representan las acciones mínimas necesarias para interactuar con la información.

En Bases de Datos NoSQL, las operaciones CRUD se adaptan a características particulares, dado que estas bases utilizan estructuras de datos no tabulares y poseen capacidad de escalado. Por medio de estas operaciones, los desarrolladores pueden llevar a cabo inserciones de registros, acceder a

datos mediante consultas, modificar información existente y eliminar entradas de manera eficiente. Este conjunto de funciones permite el flujo dinámico de información, importante para aplicaciones que requieren cambios frecuentes y acceso rápido.

La ejecución de las operaciones CRUD puede presentar variaciones entre diferentes modelos de Bases de Datos NoSQL, como bases de datos de documentos, de clave-valor, de columnas o de grafos. Cada tipo ofrece sus propios métodos y optimizaciones, lo que permite a los desarrolladores seleccionar aquellos que se adecuen mejor a las necesidades y arquitectura de la aplicación.

3.1.1. CREATE (Crear)

La operación "Create" en bases de datos NoSQL implica la inserción de nuevos datos, siendo un aspecto necesario para el manejo eficiente de información en aplicaciones que requieren flexibilidad y rapidez. Las bases de datos NoSQL representan una alternativa a las bases de datos relacionales, ofreciendo diferentes métodos para llevar a cabo la operación "Create", ajustándose así a diversas necesidades del entorno actual.

En MongoDB, la creación de documentos se realiza mediante los comandos `insertOne()` y `insertMany()`. Utilizar `insertOne()` permite agregar un único documento a una colección. Esta operación proporciona la posibilidad de establecer los campos de manera libre, sin la necesidad de un esquema rígido. *Por ejemplo, al gestionar información de contacto de clientes, se puede efectuar la inserción de la siguiente manera*:

```
db.clientes.insertOne({
    nombre: "Laura García",
    correo: "laura.garcia@example.com",
    telefono: "987654321",
    direccion: {
        calle: "Av. Libertad",
        ciudad: "Madrid",
        codigo_postal: "28001"
    }
});
```

En este caso, se utiliza una estructura anidada para organizar los datos relacionados de forma ordenada. Esta metodología resulta útil al registrar una variedad de información, permitiendo una mejor comprensión sobre los datos de contacto.

La operación `insertMany()` permite la inserción simultánea de varios documentos. En una plataforma de comercio electrónico, este comando podría ser usado así:

```
db.productos.insertMany([
    { nombre: "Smartphone", categoria: "Electrónica", precio: 699 },
    { nombre: "Tableta", categoria: "Electrónica", precio: 299 },
    { nombre: "Cámara", categoria: "Fotografía", precio: 499 }
]);
```

Esta opción resulta práctica en situaciones donde se requiere cargar productos de un catálogo de manera rápida y eficiente.

En el caso de Cassandra, el uso de comandos CQL es la forma de implementar la operación "Create". Al ejecutar un comando `INSERT`, se pueden definir todos los datos necesarios sobre un ítem. Si una aplicación gestiona reservas de alojamiento, un registro puede ser creado del siguiente modo:

INSERT INTO habitaciones (id, tipo, capacidad, precio) VALUES (UUID(), 'Doble', 2, 100);

Con este comando, se almacena información relevante acerca del tipo y capacidad de la habitación, incluyendo el precio. Esto resulta útil para la gestión de un negocio de hospedaje donde la información rápida y precisa es vital.

Cassandra permite el uso de colecciones, como listas y mapas, lo que facilita almacenar múltiples valores en una columna. *Así, un registro de habitación con varios servicios incluidos podría ser almacenado así*:

INSERT INTO habitaciones (id, tipo, capacidad, servicios) VALUES (UUID(), 'Suite', 4, {'WiFi', 'Desayuno incluido', 'Piscina'});

Esta característica resulta práctica para gestionar información adicional sobre las habitaciones de un establecimiento.

DynamoDB ofrece también una estructura flexible para las operaciones de creación. Mediante el método `put_item`, es posible insertar un ítem con múltiples atributos. U*n ejemplo de registro de un usuario en una red social podría ser*:

```
table.put_item(
    Item={
        'usuario_id': 'user123',
        'nombre': 'Pedro López',
        'email': 'pedro.lopez@example.com',
        'seguidores': 150,
        'publicaciones': [
            {'fecha': '2023-01-01', 'contenido': '¡Feliz Año Nuevo!'},
            {'fecha': '2023-01-15', 'contenido': 'Visitando la playa.'}
        ]
    }
)
```

Aquí se combinan campos simples y un atributo que incluye publicaciones, lo cual aporta riqueza a la información registrada.

Una característica distintiva de DynamoDB es el uso de índices secundarios, lo que permite crear diversas vistas sobre los mismos datos. Por ejemplo, si es necesario buscar usuarios por nombre y no solo por su identificador, se puede configurar un índice secundario, facilitando así el acceso a la información y mejorando la eficiencia de consulta.

La gestión de la operación "Create" requiere atención a la integridad y calidad de los datos ingresados. En aplicaciones orientadas a análisis, la correcta validación de la información es un aspecto a tener en cuenta para asegurar que los resultados obtenidos sean relevantes. **Las bases de datos NoSQL permiten trabajar con datos que pueden aceptar cambios a lo largo del tiempo, pero también es importante contar con un enfoque riguroso en la validación de la información mantenida**.

Algunas bases de datos NoSQL ofrecen la posibilidad de aplicar reglas de validación durante la inserción de datos. MongoDB, por ejemplo, permite establecer esquemas de validación en sus colecciones, lo que ayuda a impedir que se registre información que no cumpla con ciertos criterios definidos.

La capacidad de escalabilidad de estas bases de datos se considera en el momento de realizar la operación "Create". En entornos donde las aplicaciones se desarrollan a gran escala, **el sistema debe ser capaz de gestionar un alto volumen de inserciones simultáneas**. Estrategias como la inserción en lote o el uso de procesos asíncronos pueden contribuir a mantener un funcionamiento óptimo.

La operación "Create" en bases de datos NoSQL no solo permite la inserción de nuevos datos, sino que también facilita prácticas como la actualización dinámica de registros existentes y la migración de datos a medida que las aplicaciones evolucionan.

3.1.2. READ (Leer)

La operación "Read" o lectura en bases de datos NoSQL se basa en la capacidad de recuperar información almacenada en el sistema. Este aspecto es importante para diversas aplicaciones, ya que permite a los usuarios acceder a datos relevantes. En el entorno de bases de datos NoSQL, existen varios tipos que presentan configuraciones y métodos específicos para la lectura de datos.

En bases de datos orientadas a documentos, como MongoDB, los datos se organizan en documentos en formato JSON. Estos documentos pueden contener diferentes tipos de información y estructuras, lo que proporciona flexibilidad. La operación de lectura permite realizar consultas que filtran, ordenan y proyectan datos según los requisitos del usuario.

Por ejemplo, si una aplicación almacena información sobre productos, cada producto se guardaría como un documento. Una consulta para obtener productos de una categoría específica podría ser:

```
db.productos.find({ "categoria": "Electrónica" })
```

Esto devolvería todos los documentos que corresponden a la categoría "Electrónica".

Además, las consultas pueden ser más complejas. *Por ejemplo, para buscar productos que tengan un precio menor a un límite específico y que contengan una palabra en la descripción, se podría realizar la siguiente consulta*:

```
db.productos.find({     "precio": { "$lt": 300 },     "descripcion": { "$regex": "smart" }  })
```

Esta consulta retornaría productos cuyo precio es inferior a 300 unidades monetarias y que contienen la palabra "smart" en su descripción.

Otro tipo de bases de datos NoSQL son las que usan un modelo clave-valor, como Redis. En este modelo, los datos se almacenan como pares de clave y valor, facilitando la extracción de información. *Por ejemplo, cuando se necesita recuperar información del carrito de compras de un usuario, se podría usar una clave representativa. La operación de lectura se realizaría de la siguiente forma:*

```
GET carrito:usuario:1001
```

Este comando retornaría el contenido del carrito de compras del usuario con ID 1001. Si la información se guarda como un objeto JSON, podría incluir detalles de cada producto añadido al carrito.

Las bases de datos columnares, como Apache Cassandra, están diseñadas para almacenar datos de forma distribuida. A diferencia de las bases de datos relacionales que utilizan un formato de filas y columnas rígido, en Cassandra las columnas pueden variar de una fila a otra. Esto permite flexibilidad en las operaciones de lectura. *Por ejemplo, si se tiene una tabla que almacena información de usuarios con diferentes atributos, se puede realizar una consulta como:*

```
SELECT nombre, apellido FROM usuarios WHERE id = '2002';
```

El resultado devolvería el nombre y apellido de un usuario específico. Al tratarse de una base de datos columnar, se puede configurar el sistema para que sea eficiente en lecturas, especialmente si se realizan muchas consultas.

Las bases de datos de grafos, como Neo4j, utilizan un modelo basado en relaciones o nodos. Cada nodo representa un objeto, y los bordes representan la relación entre ellos. Realizar operaciones de lectura en este tipo de sistema permite explorar automáticamente las conexiones entre los nodos. *Por ejemplo, para encontrar amigos de un usuario en una red social, se usaría la siguiente consulta en Cypher:*

```
MATCH (usuario:Persona {nombre: "Laura"})-[:ES_AMIGO_DE]->(amigos) RETURN amigos
```

Esto no solo recupera a los amigos de Laura, sino que también permite explorar las relaciones, como encontrar amigos de sus amigos, facilitando la identificación de conexiones adicionales.

La eficiencia en las operaciones de lectura es una característica importante en bases de datos NoSQL. Con un enfoque en el manejo de grandes volúmenes de datos, el rendimiento en lecturas se puede optimizar mediante el uso de índices. *Por ejemplo, en MongoDB, se puede crear un índice en un campo que se utiliza con frecuencia en las consultas, mejorando los tiempos de respuesta. Si se anticipa que las búsquedas por ciudad serán comunes, se puede crear un índice en ese campo:*

```
db.usuarios.createIndex({ "ciudad": 1 })
```

Las consultas que utilizan ese índice resultarán en un mejor rendimiento, reduciendo el uso de recursos del sistema.

En escenarios de implementación práctica, una aplicación de streaming de música que utiliza una base de datos orientada a documentos podría permitir a los usuarios buscar canciones, álbumes o artistas. Cuando un usuario introduce un término de búsqueda, la aplicación realiza una consulta que podría incluir cualquier tipo de metadato relacionado con las canciones, como el título o el artista. *Un ejemplo de consulta en MongoDB podría ser:*

```
db.canciones.find({
    "$or": [
        { "titulo": { "$regex": "rock" } },
        { "artista": { "$regex": "Queen" } }
    ]
});
```

Esta consulta recuperaría todas las canciones que contengan "rock" en el título o que tengan "Queen" como artista.

En otro caso, una plataforma de reservas de hoteles podría utilizar una base de datos clave-valor para almacenar sesiones de usuario. Al buscar hoteles, se podrían almacenar las preferencias de búsqueda como un objeto en Redis bajo la clave correspondiente. Al leer estos valores, se podría acceder rápidamente a la información sin necesidad de realizar consultas complejas.

Las aplicaciones que utilizan bases de datos de grafos, como una plataforma de networking profesional, pueden ejecutar operaciones de lectura para mostrar conexiones entre profesionales. Consultas sobre conexiones de primer o segundo nivel permiten a los usuarios explorar su red de contactos, aumentando las posibilidades de encontrar oportunidades laborales.

Los patrones de referencia en la lectura de bases de datos NoSQL son importantes para dictar cómo se estructuran los datos. Dependiendo de la aplicación, la forma en que se realiza la lectura puede influir en la interfaz de usuario y en el modo en que se despliega la información. Por ello, es necesario entender cómo se realiza la operación y anticiparse a los usos de los datos para ofrecer una experiencia fluida y eficiente.

3.1.3. UPDATE (Actualizar)

La operación de actualización o “Update” en bases de datos NoSQL se refiere a la modificación de datos existentes en una colección. Los diferentes sistemas NoSQL permiten realizar esta función de diversas maneras, ofreciendo flexibilidad en la estructura de los datos y en la ejecución de actualizaciones.

En MongoDB, uno de los métodos más utilizados para llevar a cabo una actualización parcial es mediante el operador `$set`. Este operador facilita la modificación de campos específicos del documento, manteniendo el resto de la información intacta. *Por ejemplo, si se tiene un registro de producto en un comercio en línea, se puede actualizar solo el stock de un artículo sin afectar los demás campos:*

db.productos.update({ "id": "12345" }, { "$set": { "stock": 90 } })

Al ejecutar este comando, se actualizará el registro del producto con ID "12345", fijando un nuevo valor para el campo `stock`. La información que rodea a este campo continuará sin cambios:

{ "id": "12345", "nombre": "Camiseta", "precio": 17.99, "stock": 90, "categoria": "ropa" }

El uso del operador `$set` es eficaz en aplicaciones donde los datos requieren cambios frecuentes, evitando el tiempo de espera que suele asociarse con la reescritura total de registros.

También es posible realizar una actualización completa del documento utilizando la operación `replaceOne` en MongoDB. Este enfoque resulta apropiado cuando se quiere reestructurar completamente un registro. Por ejemplo, si se desea cambiar la representación de un producto, se puede ejecutar:

```
db.productos.replaceOne(
    { "id": "12345" },
    {
        "id": "12345",
        "nombre": "Camiseta de Algodón",
        "precio": 19.99,
        "stock": 80,
        "categoria": "ropa",
        "descripcion": "Camiseta hecha de 100% algodón."
    }
);
```

Con esto, el contenido original se sustituye por la nueva información, eliminando cualquier campo no presente en esta última.

Un aspecto a considerar en la operación de actualización es el uso del operador `$inc`, que permite incrementar o decrementar los valores de campos numéricos directamente. Este enfoque es

conveniente para gestionar contadores o stocks. *Por ejemplo, en un sistema de blog, para contar las visualizaciones de un artículo:*

```
db.articulos.update(    { "id": "abc123" },    { "$inc": { "visualizaciones": 1 } } )
```

Esta operación incrementa el número total de visualizaciones por uno cada vez que un usuario accede a dicho artículo, sin requerir la obtención del valor actual y el cálculo del siguiente.

Las actualizaciones también pueden incluir condiciones más específicas. *Por ejemplo, podría ser necesario ajustar el precio de un producto solo si su stock está por encima de un determinado número. Esto permite decisiones más precisas dentro de la lógica de negocios:*

```
db.productos.update(    { "id": "12345", "stock": { "$gt": 50 } },    { "$set": { "precio": 15.99 } } )
```

Aquí, el precio se actualizará únicamente si el stock es mayor a 50, lo que es útil en situaciones donde se aplican promociones o ajustes de precios en función de la disponibilidad de productos.

Las bases de datos NoSQL son frecuentemente utilizadas en aplicaciones en tiempo real. Por ejemplo, en una aplicación de mensajería, puede ser necesario actualizar el estado de entrega de un mensaje. Se puede modificar el campo que representa el estado de un mensaje con la siguiente operación:

```
db.mensajes.update(    { "mensajeId": "123" },    { "$set": { "estado": "leído" } } )
```

Este tipo de actualizaciones se gestionan eficazmente en sistemas que emplean tecnologías como WebSockets, permitiendo reflejar cambios de manera inmediata en la interfaz de usuario.

Al realizar estos cambios, es importante considerar la concurrencia, ya que múltiples usuarios pueden intentar alterar los mismos datos simultáneamente. Dependiendo del sistema NoSQL, pueden utilizarse diversas estrategias para manejar la concurrencia. Algunos sistemas ofrecen mecanismos de control optimista, que permiten a las operaciones de actualización proceder con la validación de que no ha habido cambios en el estado del registro desde la última lectura.

Para escenarios que involucran actualizaciones en procesos delicados, como plataformas financieras, el uso de transacciones puede ser recomendable. Esto garantiza que un conjunto de operaciones se ejecute como una única acción; si algo falla, los cambios pueden ser revertidos, manteniendo la integridad de los datos en la base de datos.

Por otro lado, diferentes sistemas NoSQL, como Couchbase o Cassandra, tienen sus propios métodos para realizar actualizaciones. *En Couchbase, el comando `UPSERT` se utiliza para insertar o actualizar documentos:*

```
UPSERT doc AS {    "tipo": "producto",    "nombre": "Zapatos deportivos",    "precio": 49.99 }
```

Con este comando, si el documento `doc` ya existe, se actualizará; si no, se creará uno nuevo. Esta capacidad de combinar inserción y actualización simplifica la lógica en ciertos usos.

La flexibilidad y adaptabilidad de las operaciones de actualización en las bases de datos NoSQL permiten a los desarrolladores crear aplicaciones dinámicas y eficientes que se adecuan a las cambiantes demandas del mercado y optimizan la experiencia del usuario mediante datos constantemente actualizados. Además, la posibilidad de realizar modificaciones específicas y de implementar condiciones complejas en las actualizaciones convierte a estas bases de datos en herramientas efectivas para la gestión de información en aplicaciones modernas.

3.1.4. DELETE (Borrar)

La operación de "Delete" o "Borrar" en bases de datos NoSQL se refiere a la capacidad de eliminar documentos, registros o claves de colecciones o estructuras de datos. Estas bases de datos se caracterizan por su versatilidad y la capacidad para gestionar grandes volúmenes de datos no estructurados. Esto da lugar a una variedad de métodos y prácticas para realizar operaciones de eliminación, que dependen del tipo de base de datos NoSQL utilizada.

En bases de datos documentales como MongoDB, el borrado implica seleccionar documentos de acuerdo con un criterio específico. **MongoDB ofrece diversas funciones para la eliminación. El método `deleteOne()` elimina el primer documento que coincide con el filtro determinado**. *Por ejemplo, si se quiere remover un usuario específico identificado por su correo electrónico, el comando sería:*

```
db.usuarios.deleteOne({ email: "ejemplo@correo.com" });
```

Este comando asegurará que solo se elimine el primer documento que corresponde al correo proporcionado.

El método `deleteMany()` se utiliza para eliminar varios documentos de una vez. *Por ejemplo, si se necesita borrar todos los documentos de un departamento cerrado, el comando podría ser:*

```
db.empleados.deleteMany({ departamento: "ventas" });
```

Este comando eliminará todos los registros en la colección "empleados" que se asocian con el departamento "ventas", facilitando la gestión eficiente de la información.

En situaciones donde se busca asegurar que los datos sean eliminados de forma cuidadosa, MongoDB permite combinar el borrado con transacciones. Por ejemplo, al eliminar usuarios de un sistema que contiene información sensible. Asegurar un seguimiento de las operaciones ayuda a implementar un manejo de errores que pueda revertir la acción en caso de que ocurra un fallo.

Un caso práctico sería en una aplicación de reservas en línea. Cuando un usuario cancela su reserva, el sistema debería eliminar la información correspondiente de manera que no afecte la consistencia de la base de datos, mientras conserva una copia de registro para auditorías futuras.

En bases de datos de tipo clave-valor como Redis, el proceso de borrado es bastante sencillo. Redis es conocido por su rapidez en la manipulación de claves y valores. Mediante el comando `DEL`, se puede eliminar una clave y el valor asociado. *Por ejemplo, al borrar la sesión de un usuario que ha cerrado su sesión, se utilizaría:*

```
DEL session:usuario123
```

Este comando retira de la base de datos la clave que contiene la información de la sesión del usuario identificado como `usuario123`. **Además, Redis permite eliminar múltiples claves usando el comando `UNLINK`, que lleva a cabo la eliminación de manera asíncrona, lo que optimiza la gestión de datos en tiempo real**.

Un ejemplo relevante sería en una aplicación de chat donde los mensajes se almacenan en memoria. Tras la finalización de una conversación, se podrían eliminar todas las claves asociadas a ese chat utilizando:

```
UNLINK chat:1001:messages
```

Este comando ayuda a gestionar de forma más rápida y eficiente el espacio en memoria.

En bases de datos orientadas a columnas, como Cassandra, se emplea un enfoque diferente para la eliminación. Debido a su arquitectura distribuida, Cassandra utiliza un sistema de marcas que indican que un registro ha sido eliminado. *Al ejecutar un comando de borrado, como*:

```
DELETE FROM usuarios WHERE id = '123';
```

Este comando no eliminará inmediatamente el registro con `id = '123'`, sino que se creará una marca que indica su eliminación. Esta característica es parte de la estructura de Cassandra, que busca asegurar la coherencia eventual. **Los registros marcados permanecen en el sistema hasta que se realiza la compactación, un proceso en el que se eliminan físicamente los datos marcados**.

Un caso de uso podría surgir en una aplicación de monitoreo de usuarios, donde se registra el comportamiento de cada usuario a lo largo del tiempo. Si un usuario solicita la eliminación de su cuenta, la aplicación podría ejecutar un comando de borrado, marcando los datos para su eliminación, aunque podrían permanecer hasta que la base de datos realice la compactación, lo que promueve un manejo seguro de la información del usuario.

Es importante tener en cuenta los aspectos de rendimiento y estrategias de acceso a los datos al aplicar operaciones de eliminación. **Si es necesario eliminar grandes volúmenes de datos, se recomienda dividir las operaciones en lotes.** Esto ayuda a prevenir efectos adversos en la disponibilidad de la base de datos y asegura que el rendimiento total no se vea comprometido. Por ejemplo, en lugar de eliminar mil registros en un solo comando, se pueden gestionar cien registros de diez en diez en intervalos regulares.

La implementación de políticas de auditoría también resulta eficaz en la gestión de operaciones de borrado. Establecer un sistema de registro que documente las eliminaciones puede ser útil en caso de necesidad de recuperación de información. En un sistema financiero donde se realizan eliminaciones frecuentes de documentos, contar con un historial de borrados puede ser valioso para auditorías y cumplimiento de normativas.

Para restaurar datos eliminados por error, algunas bases de datos NoSQL cuentan con funcionalidades de respaldo y recuperación. La utilización de herramientas que permitan crear copias de seguridad de la información antes de llevar a cabo operaciones masivas de eliminación es una práctica recomendable en ambientes de producción de alta importancia.

Las operaciones de borrado en bases de datos NoSQL abarcan una serie de funcionalidades y métodos específicos para cada tipo de base de datos, cada uno de los cuales debe ser empleado considerando el entorno operativo, el volumen de datos y los requisitos de seguridad y rendimiento. Cada método de eliminación presenta ventajas y desventajas que deben ser analizadas en su aplicación práctica.

3.2. IMPLEMENTACIÓN DE OPERACIONES CRUD EN DISTINTOS TIPOS DE NOSQL

La implementación de operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en bases de datos NoSQL se basa en distintos paradigmas de almacenamiento, donde cada tipo ofrece características particulares. Se presentará un análisis de las operaciones CRUD en diferentes tipos de bases de datos NoSQL, junto con ejemplos y casos de uso relevantes.

3.2.1. Bases de datos de documentos

Las bases de datos de documentos, como MongoDB y CouchDB, permiten almacenar información en documentos que pueden ser manipulados en formatos como JSON o BSON. Este enfoque facilita la flexibilidad y evolución del modelo de datos, permitiendo que los documentos en una colección tengan diferentes estructuras.

- **Crear**: Para insertar documentos, se utiliza el método `insertOne()` para un único documento o `insertMany()` para múltiples documentos. Por ejemplo, para registrar nuevos usuarios en una aplicación de gestión de clientes:

  db.customers.insertMany([{ name: "Juan", age: 30, email: "juan@example.com" }, { name: "Maria", age: 28, email: "maria@example.com" }]);

  En este caso, se está creando un conjunto de documentos en la colección `customers`.

- **Leer**: Para obtener datos, `find()` permite recuperar múltiples documentos, mientras que `findOne()` retorna un solo documento. Por ejemplo, para buscar todos los clientes de una ciudad específica:

```
db.customers.find({ city: "Madrid" });
```

Aquí se están buscando todos los documentos de la colección `customers` donde la ciudad es Madrid. Esto resulta útil para realizar análisis específicos de la base de datos.

- **Actualizar**: Al actualizar un documento, `updateOne()` se usa para modificar un único documento, mientras que `updateMany()` modifica múltiples documentos que cumplen con un criterio específico. Por ejemplo:

  ```
  db.customers.updateOne(  { name: "Juan" },  { $set: { age: 31 } }  );
  ```

  Esto actualiza la edad del cliente Juan. En un escenario en el que una empresa realiza un ajuste de edad en su base de datos, sería un uso práctico de esta operación.

- **Eliminar**: La eliminación de documentos se efectúa con `deleteOne()` y `deleteMany()`. Por ejemplo, para eliminar un usuario específico:

  ```
  db.customers.deleteOne({ name: "Maria" });
  ```

  Este comando elimina el documento correspondiente a Maria, lo que puede ser necesario en situaciones donde se necesita limpiar datos o gestionar bajas de usuarios.

Un caso aplicable para este tipo de bases de datos se encuentra en sistemas de gestión de contenido (CMS), donde cada entrada puede tener atributos diversos y dinámicos.

3.2.2. Bases de datos clave-valor

Las bases de datos clave-valor, como Redis y Amazon DynamoDB, ofrecen una manera sencilla de almacenar datos utilizando pares de clave y valor. Este modelo es particularmente útil para aplicaciones con alta demanda de velocidad y eficiencia.

- **Crear**: Para almacenar un nuevo par clave-valor, se utiliza el comando `SET`. Por ejemplo, en un sistema de gestión de sesiones para una aplicación web:

  ```
  SET session:12345 "user_id:1001"
  ```

  Aquí, se establece una sesión identificada con la clave `session:12345` que almacena el ID del usuario.

- **Leer**: La recuperación de datos se realiza mediante el comando `GET`. Para acceder a la información de la sesión:

  ```
  GET session:12345
  ```

  Esto devuelve el valor asociado con la clave `session:12345`, lo que permite identificar rápidamente el usuario correspondiente.

- **Actualizar**: En este modelo, actualizar un valor se lleva a cabo con el mismo comando `SET`, ya que la redefinición de la clave con un nuevo valor reemplaza el existente:

  SET session:12345 "user_id:1002"

  Este comando permite cambiar la sesión para vincularla con un nuevo usuario.

- **Eliminar**: La eliminación de un par clave-valor se realiza utilizando el comando `DEL`:

  DEL session:12345

  En un aspecto de seguridad, este comando puede ser útil para cerrar sesiones de manera controlada.

Las bases de datos clave-valor son comunes en aplicaciones que requieren almacenamiento en memoria, como sistemas de puntuaciones en juegos o datos de configuración de aplicaciones.

3.2.3. Bases de datos de grafos

Las bases de datos de grafos, como Neo4j, están diseñadas para modelar y gestionar datos que tienen relaciones complejas, representadas por nodos y aristas. Este tipo de bases de datos resulta útil en aplicaciones donde las relaciones son de suma importancia además de los propios datos.

- **Crear**: Para crear nodos y relaciones se utiliza la instrucción `CREATE`. Por ejemplo, al modelar una red social:

  CREATE (a:User {name: "Juan", age: 30}), (b:User {name: "Maria", age: 28}), (a)-[:FRIENDS_WITH]->(b);

  Se están creando dos nodos (`User`) y una relación que indica que Juan es amigo de Maria.

- **Leer**: La consulta se realiza por medio de `MATCH`, permitiendo buscar patrones en el grafo. Para encontrar todos los amigos de Juan:

  MATCH (a:User {name: "Juan"})-[:FRIENDS_WITH]->(friend) RETURN friend;

  Esta consulta permite obtener todos los nodos relacionados con Juan a través de la relación `FRIENDS_WITH`.

- **Actualizar**: Para modificar propiedades de un nodo se utiliza `SET`:

  MATCH (a:User {name: "Juan"}) SET a.age = 31;

  Esto cambia la edad de Juan, un proceso que puede ser importante en aplicaciones que requieren mantener datos actualizados.

- **Eliminar**: Para eliminar nodos y relaciones, se usa `DELETE`. Al eliminar a Juan y sus conexiones:

MATCH (a:User {name: "Juan"}) DETACH DELETE a;

Este comando elimina el nodo de Juan junto con todas las relaciones que tenía, lo que resulta adecuado en procesos de eliminación de datos.

Las bases de datos de grafos son frecuentemente utilizadas en motores de recomendaciones, donde las relaciones entre los elementos son de gran relevancia.

3.2.4. Bases de datos en columna

Las bases de datos en columna, como Cassandra y HBase, están diseñadas para manejar grandes volúmenes de datos distribuidos entre múltiples nodos. Este enfoque permite una alta escalabilidad y rendimiento para aplicaciones que requieren lecturas y escrituras rápidas.

- **Crear**: Para insertar datos se usa la instrucción `INSERT INTO`. Por ejemplo, al almacenar los registros de ventas de un comercio:

  INSERT INTO sales (transaction_id, user_id, product_id, amount) VALUES (1, 1001, 2001, 29.99);

  Cada transacción se registra como una fila en la tabla `sales`, facilitando la auditoría y el análisis de datos.

- **Leer**: Para consultar los datos, se utiliza `SELECT`:

  SELECT * FROM sales WHERE user_id = 1001;

  Este comando recupera todas las transacciones realizadas por un usuario específico, lo cual permite establecer patrones de comportamiento en la actividad de ventas.

- **Actualizar**: La actualización se realiza con `UPDATE`, permitiendo modificar datos en una fila existente:

  UPDATE sales SET amount = 24.99 WHERE transaction_id = 1;

  En este caso, se modifica el monto de la transacción específica, necesario, por ejemplo, en escenarios que involucren descuentos o devoluciones.

- **Eliminar**: La eliminación de filas se lleva a cabo utilizando `DELETE`:

  DELETE FROM sales WHERE transaction_id = 1;

  Esta operación permite llevar a cabo la gestión de datos obsoletos o erróneos.

Las bases de datos en columna son comunes en análisis de datos y aplicaciones que manejan grandes cantidades de información, como plataformas de logística o análisis financiero.

RESUMEN

- Características:
  - Alternativa al modelo relacional tradicional.
  - Ofrecen flexibilidad, escalabilidad y alta disponibilidad para aplicaciones modernas.
- Modelos principales:
  - **Documentales** (MongoDB, CouchDB): Usan documentos JSON/BSON para datos semi-estructurados.
  - **Columnas** (Cassandra, HBase): Organizan datos en columnas, optimizando consultas analíticas.
  - **Clave-valor** (Redis, DynamoDB): Accesos rápidos mediante pares clave-valor.
  - **Grafos** (Neo4j): Modelan relaciones complejas mediante nodos y aristas.
- Operaciones CRUD:
  - Documentales: Métodos como insertOne() y find() (MongoDB).
  - Columnas: Optimización de consultas específicas (INSERT en Cassandra).
  - Clave-valor: Simplicidad con SET, GET y DEL (Redis).
  - Grafos: Uso de lenguajes como Cypher para patrones relacionales (Neo4j).
- Usos destacados:
  - Big Data, análisis en tiempo real y redes sociales.
  - Empresas como Uber y plataformas de comercio electrónico las utilizan.
- Ventajas:
  - Consistencia eventual para alta disponibilidad.
  - Flexibilidad para datos dinámicos y escalabilidad global.