Graph Databases For Beginners

Rendimiento

La cantidad de datos definitivamente aumentará en el futuro, pero lo que aumentará a un ritmo aún más rápido son las conexiones (o relaciones) entre sus puntos de datos individuales.

Flexibilidad

Con las bases de datos de gráficos, sus equipos de TI y arquitectos de datos se mueven al ritmo del negocio porque la estructura y el esquema de un modelo de datos de gráfico se flexibilizan a medida que cambian sus soluciones e industria.

Agilidad

Desarrollar con bases de datos de gráficos se alinea perfectamente con las prácticas de desarrollo ágil y basadas en pruebas de hoy en día, lo que permite que su aplicación respaldada por una base de datos de gráficos evolucione junto con sus cambiantes requisitos comerciales.

¿Qué es una base de datos de gráficos? (Una definición no técnica)

Un gráfico está compuesto por dos elementos: un nodo y una relación. Cada nodo representa una entidad (una persona, lugar, cosa, categoría u otro dato), y cada relación representa cómo se relacionan dos nodos.

¿Por qué importan las relaciones de datos?

La ironía de las bases de datos relacionales

Las bases de datos relacionales (RDBMS) fueron diseñadas originalmente para codificar formularios de papel y estructuras tabulares, y aún lo hacen excepcionalmente bien. Irónicamente, sin embargo, las bases de datos relacionales no son efectivas para manejar relaciones de datos, especialmente cuando esas relaciones se agregan o ajustan de manera ad hoc.

¿Por qué otras bases de datos NoSQL no solucionan el problema?

Otras bases de datos NoSQL (o Not only SQL) almacenan conjuntos de documentos, valores y columnas desconectados, lo que les da una ventaja de rendimiento sobre las bases de datos relacionales. Sin embargo, su construcción desconectada dificulta el uso adecuado de las relaciones de datos.

Los grafos colocan las relaciones de datos en el centro

Cuando se quiere una imagen cohesiva de los datos masivos, incluidas las conexiones entre los elementos, se necesita una base de datos de grafos. A diferencia de las bases de datos relacionales y NoSQL, las bases de datos de grafos almacenan las relaciones de datos como relaciones. Este almacenamiento explícito de datos de relación significa menos desconexiones entre su esquema en evolución y su base de datos real.

De hecho, la flexibilidad de un modelo de grafo le permite agregar nuevos nodos y relaciones sin comprometer su red existente ni migrar sus datos costosamente. Todos sus datos originales (y sus relaciones originales) permanecen intactos.

Con las relaciones de datos en su centro, los grafos son increíblemente eficientes cuando se trata de la velocidad de las consultas, incluso para consultas profundas y complejas.

Por qué necesitamos lenguajes de consulta para bases de datos de grafos

En el mundo de las bases de datos, es fundamental contar con un mecanismo concreto para crear, manipular y consultar datos. Esto significa que necesitamos un lenguaje de consulta.

Históricamente, el lenguaje de consulta utilizado por desarrolladores y arquitectos de datos (es decir, SQL) ha sido demasiado complicado y esotérico para que lo entiendan los tomadores de decisiones empresariales. Sin embargo, al igual que las bases de datos de grafos han hecho que el proceso de modelado sea más comprensible para los no iniciados, un lenguaje de consulta para bases de datos de grafos ha hecho que sea más fácil que nunca para cualquier persona comprender y crear sus propias consultas.

La relación íntima entre el modelado y la consulta garantiza que no haya discrepancias entre cómo se construyen los datos y cómo se analizan. Un lenguaje de consulta representa su modelo de cerca. Por eso, SQL se centra en las tablas y las uniones, mientras que Cypher se centra en las relaciones entre entidades. Al igual que el modelo de grafos es más natural para trabajar, Cypher es más fácil de entender ya que se basa en la representación pictórica de círculos conectados con flechas que incluso un niño puede entender.

En una base de datos relacional, el proceso de modelado de datos está tan alejado de las consultas SQL diarias que hay una gran disparidad entre el análisis y la implementación. En otras palabras, el proceso de construir un modelo de base de datos relacional no es adecuado para hacer (y responder) preguntas eficientemente desde ese mismo modelo.

Por otro lado, los modelos de bases de datos de grafos no solo comunican cómo están relacionados sus datos, sino que también ayudan a comunicar claramente los tipos de preguntas que desea hacer sobre su modelo de datos. Los modelos de grafos y las consultas de grafos son dos caras de la misma moneda.

Cypher es el lenguaje de consulta diseñado para ser fácilmente leído y entendido por desarrolladores, profesionales de bases de datos y tomadores de decisiones empresariales. Es fácil de usar porque coincide con la forma en que intuitivamente describimos los grafos utilizando diagramas.

La idea básica de Cypher es que le permite pedir a la base de datos que encuentre datos que coincidan con un patrón específico. Colloquially, podríamos pedir a la base de datos que "encuentre cosas como esta", y la forma en que describimos cómo se ven las "cosas como esta" es dibujándolas usando arte ASCII.

Como la mayoría de los lenguajes de consulta, Cypher está compuesto por cláusulas. Las consultas más simples consisten en una cláusula MATCH seguida de una cláusula RETURN.

Otros tipos de cláusulas de Cypher incluyen: WHERE, CREATE y CREATE UNIQUE, MERGE, DELETE / REMOVE, SET, ORDER BY, SKIP LIMIT, FOREACH, UNION, WITH. Las partes de la consulta se encadenan y los resultados se transmiten de una a la siguiente. Es similar a la canalización de comandos en Unix.

Lenguajes de consulta imperativos vs. declarativos

Lenguajes de consulta imperativos

Los lenguajes de consulta imperativos se utilizan para describir específicamente cómo se desea que se realice una tarea. Esto se logra con un control explícito de manera detallada y paso a paso; la secuencia y redacción de cada línea de código juega un papel crítico.

Algunos ejemplos de lenguajes de programación imperativos son Python, C y Java. En el mundo de las bases de datos de gráficos, no existen lenguajes de consulta puramente imperativos, pero tanto Gremlin como la API de Java (para Neo4j) incluyen características imperativas. Estas dos opciones proporcionan un mayor poder detallado sobre la ejecución de su tarea. Si se escriben correctamente, no hay sorpresas, obtendrá exactamente lo que desea.

Sin embargo, los lenguajes de consulta imperativos también pueden ser limitantes y no muy amigables para el usuario, lo que requiere un conocimiento extenso del lenguaje y una comprensión técnica profunda de los detalles de implementación físicos antes de su uso. Escribir una parte incorrectamente puede generar resultados defectuosos.

Además, los lenguajes imperativos son más propensos a errores humanos. Los usuarios también deben verificar el entorno antes y después de la consulta y estar preparados para enfrentar posibles escenarios erróneos.

Lenguajes de consulta declarativos

Los lenguajes de consulta declarativos permiten a los usuarios expresar qué datos recuperar, dejando que el motor de base de datos se encargue de recuperarlos sin problemas. Funcionan de manera más general y consisten en dar instrucciones amplias sobre qué tarea debe completarse, en lugar de los detalles específicos sobre cómo completarla. Se ocupan de los resultados en lugar del proceso, centrándose así menos en los detalles finos de cada tarea.

Algunos ejemplos de lenguajes de programación declarativos son Ruby, R y Haskell. SQL (Structured Query Language) también es un lenguaje de consulta declarativo y es el estándar de la industria para las bases de datos relacionales. En el ecosistema de bases de datos de gráficos, varios lenguajes de consulta se consideran declarativos: Cypher, SPARQL y Gremlin (que también incluye algunas características imperativas).

El uso de un lenguaje de consulta declarativo también puede resultar en un mejor código que lo que se puede crear manualmente, y generalmente es más fácil de entender el propósito del código escrito en un lenguaje declarativo. Los lenguajes de consulta declarativos también son más fáciles de usar, ya que se centran simplemente en lo que se debe recuperar y lo hacen rápidamente.

Sin embargo, los lenguajes declarativos también tienen sus propios compromisos. Los usuarios tienen poco o ningún control sobre cómo se manejan las entradas. Si hay un error en el lenguaje, el usuario deberá confiar en los proveedores del lenguaje para solucionar el problema. Del mismo modo, si el usuario quiere usar una función que el lenguaje de consulta no admite, a menudo se encuentra en una situación difícil.

Teoría de Grafos y Modelado Predictivo

Cierre Triádico

Esta es la observación de que si dos nodos están conectados a través de un camino con un tercer nodo mutuo, hay una mayor probabilidad de que los dos nodos se conecten directamente en el futuro.

Puentes Locales

Podemos ir más allá y obtener información más valiosa sobre el flujo de comunicaciones de nuestras organizaciones al analizar los puentes locales. Estos se refieren a un vínculo entre dos nodos donde los puntos finales del puente local no están conectados de otra manera, ni comparten vecinos comunes. Puedes pensar en los puentes locales como conexiones entre dos grupos distintos del grafo. En este caso, uno de los vínculos tiene que ser débil.

Por qué necesitamos bases de datos NoSQL

El mundo de las bases de datos NoSQL ha surgido como respuesta a los desafíos de los datos modernos: el volumen de datos, la velocidad de los datos y la variedad de los datos.
El volumen de datos es el primer desafío que enfrentan las bases de datos relacionales (RDBMS), ya que los conjuntos de datos grandes se vuelven muy difíciles de manejar. Además, el tiempo de ejecución de las consultas aumenta a medida que el tamaño de las tablas y el número de JOIN crecen.
La velocidad de los datos es el siguiente desafío, ya que los cambios en los datos pueden ocurrir a una velocidad variable. Las bases de datos NoSQL están diseñadas para manejar grandes cargas de escritura y picos de actividad en la base de datos, algo que las bases de datos relacionales no pueden manejar adecuadamente.
La variedad de los datos es el tercer desafío. Las bases de datos relacionales no pueden manejar datos tan variados como los que tenemos hoy en día. En cambio, las bases de datos NoSQL están diseñadas para adaptarse a una amplia variedad de datos y responder a las necesidades futuras.
En resumen, las bases de datos NoSQL están diseñadas para manejar grandes volúmenes de datos, velocidades variables y una amplia variedad de datos. Esto las hace una alternativa viable a las bases de datos relacionales tradicionales.

ACID vs BASE explicado

El modelo de consistencia ACID:

- Atómico: todas las operaciones en una transacción tienen éxito o todas las operaciones se deshacen.
  Consistente: después de completar una transacción, la base de datos está estructuralmente sólida.
- Aislado: las transacciones no compiten entre sí y el acceso conflictivo a los datos es moderado por la base de datos para que las transacciones parezcan ejecutarse secuencialmente.
- Duradero: los resultados de aplicar una transacción son permanentes, incluso en presencia de fallas.

El modelo de consistencia BASE:

- Disponibilidad básica: la base de datos parece funcionar la mayor parte del tiempo.
- Estado suave: los almacenamientos no tienen que ser consistentes en la escritura, ni diferentes réplicas tienen que ser mutuamente consistentes todo el tiempo.
- Consistencia eventual: los almacenamientos exhiben consistencia en algún momento posterior (por ejemplo, de manera perezosa en el momento de la lectura).

Un recorrido por los almacenamientos de agregados (Aggregate Stores)

- Los almacenamientos de clave-valor (Key Value Stores) son estructuras de datos hash distribuidas que almacenan y recuperan valores organizados por identificadores conocidos como claves.
- Los almacenamientos de familias de columnas (Column Family Stores), también conocidas como tiendas de columnas amplias, se basan en una tabla escasamente poblada cuyas filas pueden contener columnas arbitrarias y donde las claves proporcionan una indexación natural.
- Los almacenamientos de documentos (Document Stores) almacenan y recuperan documentos como si fuera una carpeta electrónica. Los documentos pueden incluir mapas y listas, lo que permite jerarquías naturales. El modelo de documento generalmente implica tener un documento JSON jerárquico como la estructura de datos primaria, y cualquier campo dentro de la jerarquía puede ser indexado.
  Para consultas simples, los almacenamientos de agregados utilizan la indexación, el enlace básico de documentos o un lenguaje de consulta.
  Sin embargo, para consultas más complejas, los almacenamientos de agregados no pueden generar información más profunda simplemente examinando puntos de datos individuales. Para compensar, una aplicación típicamente tiene que identificar y extraer un subconjunto de datos y ejecutarlo a través de una infraestructura de procesamiento externa como el framework MapReduce (a menudo en forma de Apache Hadoop). MapReduce es un modelo de programación paralela que divide los datos y opera en paralelo antes de reunirlos y agregarlos para proporcionar información enfocada.