|  |  |  |
| --- | --- | --- |
|  |  | neo4j  Juan Casado Ballesteros  Gino Cocolo Rodríguez |

A picture containing vector graphics

Description automatically generated

# Índices

[Problema Existente 4](#_Toc34994186)

[Solución 6](#_Toc34994187)

[Características adicionales 6](#_Toc34994188)

[Desventajas 7](#_Toc34994189)

[Casa tipo de base de datos tiene ciertas ventajas sobre el resto sin ser ninguna la solución ideal para todos los problemas. 7](#_Toc34994190)

[En el caso de las bases de datos orientas a grafos uno de sus principales problemas es la falta de flexibilidad en su esquema. Debido a que la velocidad para realizar consultas se basa en tener referencias directas a los elementos sobre los que existe cada relación el tamaño de estos debe ser fijo y conocido con antelación. Esto hace que tampoco sean una buena base de datos para almacenar archivos de gran tamaño. 7](#_Toc34994191)

[Respecto de la forma en la que se realizan las consultas estas no son transaccionales. Es decir, si los datos cambian mientras la consulta se realiza no sabremos cuales se nos proporcionarán, los que existían antes comenzar la consulta o los que existían después del cambio. 7](#_Toc34994192)

[¿Qué es Neo4j? 8](#_Toc34994193)

[Modelo de datos 8](#_Toc34994194)

[Nodos 8](#_Toc34994195)

[Etiquetas 8](#_Toc34994196)

[Relaciones 9](#_Toc34994197)

[Propiedades 9](#_Toc34994198)

[Características propias de una base de datos 9](#_Toc34994199)

[Lenguaje de consultas 10](#_Toc34994200)

[CREATE-DELETE 10](#_Toc34994201)

[SET-REMOVE 10](#_Toc34994202)

[MATCH 10](#_Toc34994203)

[RETURN 11](#_Toc34994204)

[WHERE 11](#_Toc34994205)

[ORDER BY 12](#_Toc34994206)

[LIMIT 12](#_Toc34994207)

[WITH 12](#_Toc34994208)

[UNION 12](#_Toc34994209)

[FOREACH 13](#_Toc34994210)

[Replicación 13](#_Toc34994211)

[Demostración 14](#_Toc34994212)

[Bibliografía 15](#_Toc34994213)

# Problema Existente

Las bases de datos relacionales almacenan información estructurada de forma rígida en forma de tablas. Para aprovechar completamente las características de estas bases de datos el diseño de tablas debe normalizarse hasta alcanzar la forma normal deseada.

La información organizada en tablas se relaciona mediante las operaciones proporcionadas por el álgebra relacional.

Esta estructura, aunque idónea para garantizar la consistencia y la no duplicidad de información no es la más adecuada para todas las necesidades. Algunos de los problemas que plantea son los siguientes:

* Las uniones o relaciones entre tablas son operaciones del álgebra relacional que deben ejecutarse cada vez que se consulte la relación.

Para evitar hacer esto en ocasiones se suele recurrir a des-normalizar las tablas lo cual rompe con los principios de diseño de las bases de datos relacionales.

* Ejecutar una unión entre tablas es equivalente a buscar valores procedentes de una tabla en otra lo cual tampoco es eficiente.
* La creación de ciertos tipos de índices necesarios para buscar conjuntos de datos por clave puede no se óptima para buscar datos únicos. Debemos de ser conscientes en todo momento a la hora de crear nuevos índices de la forma en la que los datos serán buscados para cada clave.
* Las bases de datos relacionales se vuelven cada vez más lentas al realizar las operaciones de relación cuantos más datos hay almacenados en la base de datos.

Como conclusión podemos destacar que las bases de datos relacionales presentan problemas de rendimiento a la hora de realizar búsquedas que hacen uso de relaciones entre tablas.

Estos problemas no son solo propios de las bases de datos relacionales. Suceden también en otras bases de datos NoSQL como las documentales o las orientas a clave valor. Estas bases de datos están especializadas para almacenar otros tipos de información como documentos o archivos de gran tamaño. No obstante, todas ellas son lentas a la hora de manejar relaciones de datos.

También es importante destacar que los lenguajes de consultas que estas otras bases de datos utilizan no están adecuados para manejar relaciones con una sintaxis suficientemente expresiva.

# Solución

Las bases de datos orientadas a grafos tratan las relaciones como su principal activo. En lugar de centrarse en la forma en la que los datos son almacenados: tablas, documentos, columnas… se centran en las relaciones que existen entre ellos.

Su rapidez a la hora de manipular las relaciones proviene de la forma en la que se busca la información. En otros tipos de bases de datos de información debe de ser buscada mediante búsquedas secuenciales, binarias si la información está ordenada o por medio de índices. En el caso de las bases de datos orientadas a grafos las relaciones son directas, cada nodo contendrá punteros a las relaciones de las que participa y estas contendrán punteros a los nodos con los que el participante está relacionado de forma directa.

Esta forma de relacionar datos mediante dos indirecciones es mucho más escalable que hacerlo mediante una búsqueda o que impedir crear relaciones directas por medio del propio lenguaje de consultas. Esto se traduce a que el aumento del coste de realizar las consultas será lineal con respecto a la cantidad de datos existente sin importar la cantidad de relaciones que haya que resolver.

## Características adicionales

Las bases de datos orientadas a grafos suponen algo más que un cambio en la forma de almacenar la información. También proporcionan mejores técnicas de visualización de la información pensadas para visualizar grafos con los que se pueda interactuar.

No obstante, su característica más relevante es la de proporcionar nuevos lenguajes de consultas orientados a manipular relaciones. En el caso de neo4j este lenguaje es Cypher, pero otras bases de datos proporcionan lenguajes de consultas similares como es el caso de GSQL para la base de datos TigerGraph.

Mediante este cambio de mentalidad que las bases de datos orientadas a grafos proporcionan y ayudados por las herramientas que ofrecen se puede de forma sencilla realizar acciones como:

* Averiguar el modelo habitual de las relaciones entre dos elementos. Lo cual nos permitiría:
  + Identificar elementos que se comportan de forma distinta para poder indagar en la razón por la que lo hacen. (Detección de fraude o usuarios maliciosos)
  + Identificar elementos similares. (permitiría por ejemplo realizar recomendaciones a un usuario en base a lo que los otros usuarios que se comportan como él estén haciendo)
* Predicción de atributos en base a los nuevos flujos de relaciones.
* Crear agrupaciones de modelos similares dentro de una misma relación.

## Desventajas

## Casa tipo de base de datos tiene ciertas ventajas sobre el resto sin ser ninguna la solución ideal para todos los problemas.

## En el caso de las bases de datos orientas a grafos uno de sus principales problemas es la falta de flexibilidad en su esquema. Debido a que la velocidad para realizar consultas se basa en tener referencias directas a los elementos sobre los que existe cada relación el tamaño de estos debe ser fijo y conocido con antelación. Esto hace que tampoco sean una buena base de datos para almacenar archivos de gran tamaño.

## Respecto de la forma en la que se realizan las consultas estas no son transaccionales. Es decir, si los datos cambian mientras la consulta se realiza no sabremos cuales se nos proporcionarán, los que existían antes comenzar la consulta o los que existían después del cambio.

Cuando estas bases de datos se escalan no son capaces de proporcionar consistencia, aunque si tolerancia a particiones y alta disponibilidad. La falta de consistencia implica que puede que dos consultas consecutivas no proporcionen la misma información de forma momentánea, aunque sí vayan a hacerlo eventualmente en el futuro.

# ¿Qué es Neo4j?

Neo4j es una base de datos orientada a grafos de código abierto e implementada en Java. Como lenguaje de consultas utiliza Cypher sobre un socket HTTP de modo que desde cualquier otro lenguaje que tenga acceso al stack de red se pueden realizar consultas.

## Modelo de datos

Los datos almacenamos en la base de datos de organizan creando un grafo. Los elementos más relevantes de este grafo son los siguientes:

### Nodos

Desde el punto de vista de la orientación a objetos los nodos serían entidades o instancias, desde el punto de vista de las bases de datos relacionales los nodos serían las filas.

Un nodo es la unidad completa mínima manejada por las bases de datos orientadas a grafos. Un nodo podrá tener un conjunto de atributos o pares clave valor de clave y valor conocidos e instanciados.

**Persona:**

Nombre: Pepe

Edad: 21

### Etiquetas

Desde el punto de vista de la orientación a objetos las etiquetas serían las clases, desde el punto de vista de las bases de datos relacionales las etiquetas serían las tablas.

Las etiquetas son los distintos tipos de nodos que existirán en la base de datos.

Las etiquetas pueden asignarse y retirarse de los nodos de forma dinámica a lo largo del ciclo de vida de los datos de modo que la instanciación de un nodo no nos compromete con una etiqueta concreta. Adicionalmente un nodo podrá tener asignada más de una etiqueta.

**Persona:**

Nombre: Pepe

Edad: 21

**Trabajador:**

Trabajo: Panadero

Lugar: Pontevedra

**Persona: Trabajador:**

Nombre: Pepe

Edad: 21

Trabajo: Panadero

Lugar: Pontevedra

Una etiqueta puede requerir de la existencia de cero a varios pares de clave valor en el nodo, es decir, puede ocurrir que si una etiqueta no requiere de la existencia de pares de clave valor para los nodos asignados con ella esta pueda ser asignada a cualquier nodo en cualquier momento.

### Relaciones

Las relaciones conectan dos etiquetas entre sí. Las relaciones dotan al grafo de una estructura. Las etiquetas relacionadas pueden ser la misma o ser distintas y como máximo pueden ser dos. Todos los nodos dotados con una etiqueta comenzarán a forma parte de todas las relaciones en las que la etiqueta intervenga.

Las relaciones son direccionales, es decir, van desde una etiqueta a otra y además pueden tener propiedades. Las propiedades de las etiquetas se asignan a parejas de nodos.

**Producto:**

nombre: Naranja

precio: 5.23

COMPRAR

**Persona:**

nombre: Pepe

edad: 21

### Propiedades

Son las parejas de clave valor que pueden encontrarse tanto en nodos como en relaciones. El valor también puede ser una lista de valores asignada a la clave.

## Características propias de una base de datos

Del mismo modo que sucede en otros tipos de bases de datos en node4j podremos crear índices con la intención de aumentar el rendimiento de las consultas.

También podremos garantizar la unicidad de una propiedad de los nodos asignados a una etiqueta. En el ejemplo utilizado podría ser garantizar que dos personas no puedan tener el mismo nombre.

El problema de las restricciones de unicidad es que para poder ser implementadas debe de existir un índice sobre dicha propiedad de las etiquetas.

También podremos importar datos en formato .csv desde archivos locales o remotos como en cualquier otra base de datos.

LOAD CSV FROM <file:///personas.csv> AS personas

CREATE (:Persona {nombre: personas[1], edad: personas[2]})

## Lenguaje de consultas

Node4j utiliza el lenguaje de consultas Cypher, el cual solo funciona en su propia base de datos.

### CREATE-DELETE

Permite añadir nodos o instancias sobre una etiqueta.

CREATE (pepe :Persona {nombre: “Pepe”, edad: ‘21’})

CREATE (naranja :Producto {nombre: “Naranja”, precio: 5.23})

También permite crear relaciones entre nodos.

CREATE (pepe)-[:COMPRAR]->(naranja)

Podremos borrar los nodos y las relaciones mediante DELETE

MATCH (()-[compras:COMPRAR]-()) DELETE compras

MATCH (persona :Persona {name: ‘Pepe’}) DELETE persona

### SET-REMOVE

SET nos permite cambiar el valor de una propiedad por otro y REMOVE establecerá el valor de la propiedad como null.

MATCH (persona :Persona) WHERE persona.nombre=”Pepe” SET persona.edad = 22

MATCH (persona :Persona) WHERE persona.nombre=”Pepe” REMOVE n:edad

### MATCH

Esta sentencia nos permite buscar patrones dentro del grafo de datos. Los patrones son definidos por una etiqueta ya sea de nodo o relación y por paréntesis, corchetes y flechas indicando la dirección de las relaciones.

Como podemos ver en las consultas anteriores la sentencia MATCH es muy expresiva:

* () Cualquier nodo
* [] Cualquier relación
* (:Etiqueta) Un nodo de un tipo concreto
* [:Etiqueta] Una relación de un tipo concreto
* {clave:valor, clave:valor} Restricciones sobre las propiedades de nodos o relaciones concretas.
* (nodo :Etiqueta) o [relación :Relacion] Crean variables ficticias equivalentes a cualquier nodo o relación de la etiqueta indicada que cumpla las restricciones que se le impongan.
* ()-[]-()-[]-() Tres nodos cualquiera relacionados por dos relaciones cualquiera.

Mediante construcciones similares a estas se puede crear cualquier patrón dentro de un grafo.

### RETURN

La sentencia return nos permite indicar que partes del patrón creado mediante MATCH nos interesa obtener como resultado de haber ejecutado la consulta.

Se pueden obtener nodos, relaciones o subgrafos.

MATCH (persona :Persona {name: ‘Pepe’})-[:COMPARA]->(:Producto) RETURN persona

MATCH (:Persona {name: ‘Pepe’})-[compra :COMPARA]->(:Producto) RETURN compra

MATCH grafo = (:Persona {name: ‘Pepe’})-[:COMPARA]->(:Producto) RETURN grafo

### WHERE

Añade restricciones y filtros al patrón creado mediante MATCH. Se debe destacar que algunos de estos filtros se pueden incluir dentro del propio patrón mientras que otros no se podrían.

MATCH (persona :Persona) WHERE persona.edad < 30 RETURN persona

MATCH (persona :Persona) WHERE persona.nombre = ‘Pepe’ persona

### ORDER BY

Permite ordenar el patrón seleccionado de forma ascendente (por defecto) o descendente (DESC) según una o varias propiedades.

MATCH (persona :Persona)

RETURN persona.nombre, persona.edad

ORDER BY persona.nombre, persona.edad

### LIMIT

Permite truncar la cantidad de instancias del patrón que serán devueltas por la consulta.

MATCH (persona :Persona) RETURN persona LIMIT 3

### WITH

Permite aplicar filtros y operaciones matemáticas a nivel de propiedad y de grafo sobre el patrón indicado en MATCH antes de devolverlo. Los filtros más comunes son por agregación, por orden y por factor de ramificación.

MATCH (persona :Persona)-[:COMPRAR]->(producto :Producto)

WITH persona, producto, count(\*) AS cantidad\_producto WHERE cantidad\_producto < 5

RETURN persona.nombre

MATCH (persona :Persona)

WITH persona ORDER BY persona.nombre DESC LIMIT 3

RETURN collect (persona.nombre)

### UNION

Combina el resultado de dos consultas distintas. Por defecto se eliminarán los duplicados, pero podremos mantenerlo aumentando la velocidad de la consulta mediante la sentencia ALL.

MATCH (persona :Persona) RETURN persona.nombre AS nombre

UNION ALL

MATCH (product: Producto) RETURN product.nombre AS nombre

### FOREACH

Permite aplicar operaciones a todos los patrones seleccionados.

MATCH (persona :Persona) WITH COLLECT(persona) AS lista FOREACH (n IN lista | SET n.edad=n.edad+1)

## Replicación

Al igual que la mayoría de las bases de datos node4j proporciona herramientas de replicación.

Su modelo de replicación busca proporcionar la posibilidad de realizar particiones y alta disponibilidad de los datos. Se crearán nodos maestros y nodos réplica. Los nodos maestros serán responsables de garantizar la consistencia eventual de los datos y de realizar las escrituras de los datos. Por el contrario, los nodos réplicas son solo de lectura. Algunos de los nodos réplica tendrán la responsabilidad adicional de realizar informes y análisis sobre los datos almacenados.

Este esquema de replicación esta orquestado por los llamados servidores de aplicación. Los servidores de aplicación recibirán las peticiones en el lenguaje Cypher y dependiendo de si son lecturas o escrituras las reenviarán a nodos maestros o de réplica para ser procesadas. En caso de que sean una escritura las réplicas quedarán desactualizadas hasta que eventualmente y de forma asíncrona alguno de los nodos maestros las actualice.

# Demostración

Inciciamos una nueva base de datos con docker mediante el comando:

docker run --publish=7474:7474 --publish=7687:7687 neo4j:4.0

CALL db.createLabel("Persona")

CALL db.createLabel("Producto")

CALL db.createRelationshipType("COMPRAR")

CALL db.createProperty("nombre ")

CALL db.createProperty("edad")

CALL db.createProperty("precio")

CREATE (:Persona{nombre:"Pepe", edad:"21"})-[:COMPRAR]->(:Producto{nombre:"Naranja",precio:5.23})

CREATE CONSTRAINT ON (movie:Movie) ASSERT movie.title IS UNIQUE

Desde la aplicación de neo4j para visualizar la base de datos y realizar las consultas disponible en … nos conectamos con

:server connect

El usuario y la contraseña serán neo4j

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

# Bibliografía

Información general sobre neoj4, instalación, configuración y uso: <https://neo4j.com/docs/operations-manual/current/>

Información sobre Cypher, el lenguaje de consultas para neo4j:

<https://neo4j.com/docs/cypher-manual/current/>

Presentaciones con introducción y ejemplos a neo4j:

<https://es.slideshare.net/maxdemarzi/>

Ejemplos de uso de neo4j con distintos lenguajes y distintos fines:

<https://github.com/maxdemarzi>

Documentación de otra base de datos orientada a grafos:

<https://docs.tigergraph.com>

<https://neo4j.com/blog/data-modeling-basics/>

<https://neo4j.com/sandbox/>