

Neo4j

Base de Datos orientada a grafos (BDOG)

Un grafo es una representación pictórica de un conjunto de objetos donde algunos pares de objetos están conectados por enlaces.

Se compone de dos elementos: nodos (vértices) y relaciones (aristas).

Una base de datos de grafos es una base de datos utilizada para modelar los datos en forma de grafo. Aquí, los nodos de un grafo representan las entidades, mientras que las relaciones representan la asociación de estos nodos.

Base de Datos Orientada a Grafos

Una base de datos orientada a grafos es un sistema de administración de bases de datos en línea con operaciones de Crear, Leer, Actualizar y Eliminar (CRUD) que trabajan en un modelo de datos de grafos.

A diferencia de otras bases de datos, las relaciones tienen prioridad en las bases de datos de orientadas a grafos.

- El modelo de datos para una BDOG también es significativamente más simple y más expresivo que los de bases de datos relacionales u otras bases de datos NoSQL.

Las bases de datos de grafos se crean para su uso con sistemas transaccionales (OLTP) y se diseñan teniendo en cuenta la integridad transaccional y la disponibilidad operativa.

¿Por qué bases de datos de grafos?

- Hoy en día, la mayoría de los datos existen en forma de relación entre diferentes objetos y, más a menudo, la relación entre los datos es más valiosa que los datos en sí.
- Las bases de datos relacionales almacenan datos altamente estructurados que tienen varios registros que almacenan el mismo tipo de datos para que puedan usarse para almacenar datos estructurados y no almacenan las relaciones entre los datos.
- A diferencia de otras bases de datos, las bases de datos de grafos almacenan relaciones y conexiones como entidades de primera clase.
- El modelo de datos para bases de datos de grafos es más simple en comparación con otras bases de datos y se pueden usar con sistemas OLTP. Proporcionan características como la integridad transaccional y la disponibilidad operativa.

-
- Hay dos propiedades de las BDOG a considerar:
 - El almacenamiento subyacente: Algunas BDOG utilizan el almacenamiento de grafos nativo que está optimizado y diseñado para almacenar y administrar grafos.
 - Sin embargo, no todas las tecnologías de base de datos de grafos utilizan el almacenamiento nativo.
 - Algunos serializan los datos del grafo en una base de datos relacional, una base de datos orientada a objetos o algún otro data warehouse de propósito general.
 - El motor de procesamiento: Algunas definiciones requieren que una BDOG use adyacencia sin índice, lo que significa que los nodos conectados se "apuntan" entre sí en la base de datos.
 - Cualquier base de datos, que desde la perspectiva del usuario se comporte como una BDOG (es decir, expone un modelo de datos de grafos a través de operaciones CRUD), califica como una base de datos orientada a grafos. Sin embargo, reconocemos las importantes ventajas de rendimiento de la adyacencia sin índice y, por lo tanto, utilizamos el término procesamiento de grafos nativo para describir las BDOG que aprovechan la adyacencia sin índice.

RDBMS	Base de datos de grafos
Tablas	Grafos
Filas	Nodos
Columnas y datos	Propiedades y sus valores
Restricciones	Relaciones
Joins	Navegar

- A continuación se muestra la tabla que compara las bases de datos relacionales y las bases de datos de grafos.

RDBMS vs base de datos de grafos

Es importante tener en cuenta que el almacenamiento de grafos nativo y el procesamiento de grafos nativo no son ni buenos ni malos, sino que simplemente presentan ciertas ventajas o desventajas dependiendo de la situación.

- La ventaja del almacenamiento nativo de grafos es que la infraestructura de distribución de los datos ha sido construida y diseñada especialmente para un buen rendimiento y una alta escalabilidad en el tratamiento de modelos de grafos.
- El beneficio del almacenamiento de grafos no nativo, en contraste, es que generalmente depende de un backend no basado en grafos, con muchos años de experiencia (como MySQL), cuyas características de producción son bien conocidas por los equipos de administración.

El procesamiento de grafos nativo a través de la adyacencia libre de índices beneficia el rendimiento de consulta de los grafos, ya que permite el recorrido de los nodos a través de sus relaciones (consulta por recorrido), pero a expensas de realizar algunas consultas con cierta complejidad o de alto consumo de memoria.

Neo4j

- Neo4j fue desarrollado por Neo Technology, una startup sueca con base en Malmö y San Francisco Bay Area en Estados Unidos.
- Es una base de datos orientada a grafos nativa de código abierto, NoSQL, que proporciona un backend transaccional compatible con ACID para sus aplicaciones.
 - El desarrollo inicial comenzó en 2003, pero ha estado disponible públicamente desde 2007. Su primera versión fue lanzada en febrero de 2010.
 - El código fuente, escrito en Java y Scala, está disponible de forma gratuita en GitHub o como una descarga de aplicaciones de escritorio fácil de usar.
- Neo4j tiene una edición comunitaria y una edición empresarial de la base de datos. Enterprise Edition incluye todo lo que Community Edition tiene para ofrecer, además de requisitos empresariales adicionales, como copias de seguridad, clústeres y capacidades de conmutación por error.
- Empresas como eBay, Walmart, Telenor, UBS, Cisco, Hewlett-Packard o Lufthansa han confiado en las cualidades de Neo4j para mejorar sus servicios.

- Neo4j se centra más en las relaciones entre valores que en los puntos en común entre conjuntos de valores (como colecciones de documentos o tablas de filas).
 - De esta manera, puede almacenar datos altamente variables de una manera natural y directa.
 - En un lado del espectro de escala, Neo4j es lo suficientemente pequeño como para integrarse en casi cualquier aplicación; en el otro lado del espectro, Neo4j puede ejecutarse en grandes clústeres de servidores utilizando replicación maestro-esclavo y almacenar decenas de miles de millones de nodos y otras tantas relaciones.
- En Neo4j, los nodos dentro de los grafos actúan como documentos porque almacenan propiedades, pero lo que hace que Neo4j sea especial es que la relación entre esos nodos ocupa un lugar central.

Ventajas de Neo4j

- **Modelo de datos flexible:** Neo4j proporciona un modelo de datos flexible, simple y potente, que se puede cambiar fácilmente de acuerdo con las aplicaciones y las industrias.
- **Información en tiempo real:** Neo4j proporciona resultados basados en datos en tiempo real.
- **Alta disponibilidad:** Neo4j es altamente disponible para aplicaciones en tiempo real de grandes empresas con garantías transaccionales.
- **Datos conectados y semiestructurados:** con Neo4j, se pueden representar fácilmente datos conectados y semiestructurados.
- **Fácil recuperación:** con Neo4j, no solo puede representar, sino también recuperar fácilmente (atravesar / navegar) datos conectados más rápido en comparación con otras bases de datos.
- **Lenguaje de consulta Cypher** – Neo4j proporciona un lenguaje de consulta declarativo para representar el grafo visualmente. Los comandos de este idioma están en formato legible por humanos y muy fáciles de aprender.
- **Sin uniones:** con Neo4j, no se requieren uniones complejas para recuperar datos conectados / relacionados, ya que es muy fácil recuperar sus detalles de nodos o relaciones adyacentes sin uniones o índices.

Características de Neo4j

- Modelo de datos (esquema flexible): Neo4j sigue un modelo de datos denominado modelo de grafo de propiedades nativas. Aquí, el grafo contiene nodos (entidades) y estos nodos están conectados entre sí (representados por relaciones). Los nodos y las relaciones almacenan datos en pares clave-valor conocidos como propiedades.
- En Neo4j, no hay necesidad de seguir un esquema fijo. Puede agregar o quitar propiedades según el requisito. También proporciona restricciones de esquema.
- Propiedades ACID – Neo4j soporta reglas ACID completas (Atomicidad, Consistencia, Aislamiento y Durabilidad).
- Escalabilidad y confiabilidad: puede escalar la base de datos aumentando el número de lecturas/escrituras y el volumen sin afectar la velocidad de procesamiento de consultas y la integridad de los datos. Neo4j también proporciona soporte para replicación para la seguridad y confiabilidad de los datos.
- Aplicación web incorporada: Neo4j proporciona una aplicación web Neo4j Browser incorporada. Con esto, se pueden crear y consultar los datos del grafo.
- Controladores – Neo4j puede trabajar con
 - API REST para trabajar con lenguajes de programación como Java, Spring, Scala, etc.
 - Java Script para trabajar con marcos de UI MVC como Node JS.
 - Es compatible con dos tipos de API de Java: API de Cypher y API de Java nativa para desarrollar aplicaciones Java. Además de estos, también puede trabajar con otras bases de datos como MongoDB, Cassandra, etc.
- Indexación: Neo4j admite índices mediante Apache Lucence.

Modelo de Propiedades de Grafo en Neo4j

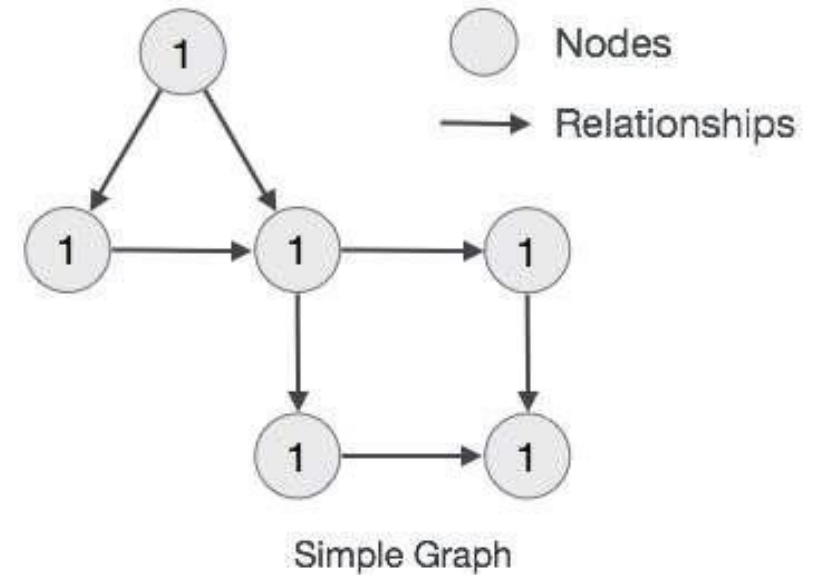
Neo4j sigue el Modelo de Propiedades de Grafo para almacenar y administrar sus datos.

Las siguientes son las características clave de Modelo de Propiedades de Grafos

- El modelo representa datos en Nodos, Relaciones y Propiedades
- Las propiedades son pares clave-valor
- Los nodos se representan mediante círculos y las relaciones se representan mediante flechas
- Las relaciones tienen direcciones: unidireccional y bidireccional
- Cada relación contiene "Nodo de Inicio" o "Nodo Desde" y "Nodo Hacia" o "Nodo Final"
- Tanto los nodos como las relaciones contienen propiedades
- Las relaciones conectan nodos

- En el modelo de datos de propiedades de grafos, las relaciones deben ser direccionales. Si intentamos crear relaciones sin dirección, se arrojará un mensaje de error de "las relaciones deben ser direccionales".
- Neo4j almacena todos sus datos en Nodos y Relaciones. No necesitamos ninguna base de datos relacional adicional para almacenar datos en Neo4j. Almacena sus datos en términos de grafos en su formato nativo.
 - Neo4j utiliza GPE (Graph Processing Engine) nativo (motor de procesamiento de grafos) para trabajar con su formato de almacenamiento de grafos nativo.

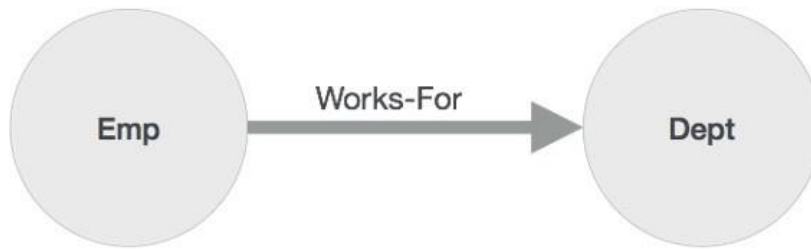
- Los principales bloques de construcción del Modelo de Grafos son:
 - Nodos
 - Relaciones
 - Propiedades
- Los Nodos se representan usando círculos.
- Las relaciones se representan mediante flechas.
 - Las relaciones son direccionales.
- Podemos representar los datos del Nodo en términos de Propiedades (pares clave-valor).
 - En este ejemplo, hemos representado la propiedad Id dentro del círculo del Nodo.



Neo4j - Bloques de construcción

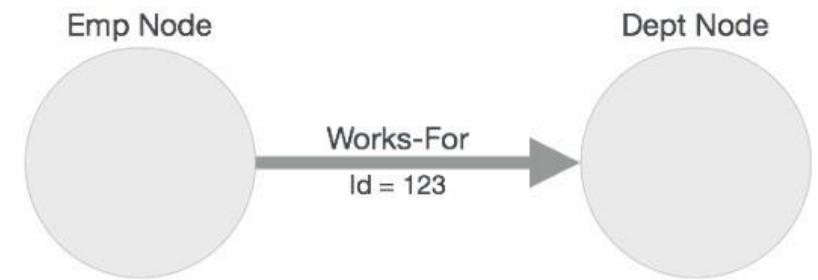
- Neo4j tiene los siguientes bloques de construcción:
 - Nodos
 - Propiedades
 - Relaciones
 - Etiquetas
 - Navegador de datos
- Nodo
 - El nodo es una unidad fundamental de un grafo. Contiene propiedades con pares clave-valor como se muestra en la siguiente imagen.
 - Aquí, Nombre de nodo = "Empleado" y contiene un conjunto de propiedades como pares clave-valor.





- Propiedades
 - La propiedad es un par clave-valor para describir los nodos y las relaciones del grafo.
- Clave = Valor
 - Donde Clave es una cadena y Valor se puede representar utilizando cualquier tipo de datos Neo4j.
- Relaciones
 - Las relaciones son otro componente importante de una base de datos de grafos. Conecta dos nodos como se muestra en la siguiente figura.
 - Aquí, Emp y Dept son dos nodos diferentes. "WORKS_FOR" es una relación entre los nodos Emp y Dept.
- Como denota, la marca de flecha de Emp a Dept, esta relación describe:
 - Emp WORKS_FOR Dept
 - Cada relación contiene un nodo de inicio y un nodo de extremo.
 - Aquí, "Emp" es un nodo de inicio, y "Dept" es un nodo final.

- Como esta marca de flecha de relación representa una relación del nodo "Emp" al nodo "Dept", la relación se conoce como "Relación Entrante" al nodo "Dept" y "Relación Saliente" del nodo "Emp".
- Al igual que los nodos, las relaciones también pueden contener propiedades como pares clave-valor.
- Aquí, la relación "WORKS_FOR" tiene una propiedad como par clave-valor, Id = 123
- Representa un Id de esta relación.



Etiquetas

Una etiqueta (Label) asocia un nombre común a un conjunto de nodos o relaciones.


- Un nodo o relación puede contener una o más etiquetas.
- Podemos crear nuevas etiquetas para nodos o relaciones existentes.
- Podemos eliminar las etiquetas existentes de los nodos o relaciones existentes.

Del diagrama anterior, podemos observar que hay dos nodos.


- El nodo del lado izquierdo tiene una etiqueta: "Emp" y el nodo del lado derecho tiene una etiqueta: "Dept".
- La relación entre esos dos nodos también tiene una etiqueta: "WORKS_FOR".


Nota – Neo4j almacena datos en Propiedades de nodos o relaciones.


Neo4j SandBox


Sandbox


Log InSign Up


 Sign in with Google

 Sign in with LinkedIn

 Sign in with Twitter

or

 yours@example.com

 your password


Don't remember your password?

LOG IN >


Select a project

☐ For Developers (10) ☒ For Data Scientists (7)

Featured Dataset



BeginnerFor Data Scientists




Graph Data Science


Leverage Neo4j Graph Data Science library to explore graph algorithms for analytics and feature engineering.

Libraries Enabled:

Graph Data Science



For Data Scientists





Bloom Visual Discovery

Neo4j Bloom is a graph exploration application for visually interacting with graph data.

Libraries Enabled:

Graph Data Science

Your own data



Blank Sandbox

A sandbox to explore connections in your own



+ New Project

Name	Status	
Blank Sandbox	Running Expires in about 3 days	Open

- Open with Browser
- Open with Bloom
- Open with NEuler

Connect to Neo4j

Database access might require an authenticated connection

Connect URL

neo4j+s://4ad0ae0ed41214e57a966260b468x

Database - leave empty for default

Authentication type

Username / Password

Username / Password

No authentication

Password

Connect

☆

?

Database Information

Use database

neo4j - default

Node Labels

There are no labels in database

Relationship Types

No relationships in database

Property Keys

There are no properties in database

Connected as

Username: neo4j
Roles: admin, PUBLIC
Admin: :server user list
 :server user add
Disconnect: :server disconnect

DBMS

Version: 4.3.7
Edition: Enterprise
Name: neo4j
Databases: :dbs
Information: :sysinfo
Query List: :queries

neo4j\$

\$:server connect

Connect to Neo4j

Database access might require an authenticated connection

You are connected as user neo4j to

bolt+s://4ad0ae0ed41214e57a966260b468d37d.neo4jsandbox..

Connection credentials are stored in your web browser.

Neo4j CQL - Introducción

- CQL son las siglas de Cypher Query Language.
- Es un lenguaje de consulta para Neo4j.
- Es un lenguaje declarativo de coincidencia de patrones.
- Sigue SQL como sintaxis.
- La sintaxis es muy simple y en formato legible por humanos.
- Tiene comandos para realizar operaciones de base de datos.
- Soporta muchas cláusulas como WHERE, ORDER BY, etc., para escribir consultas muy complejas de una manera fácil.
- Admite algunas funciones como String, Aggregation, además también es compatible con algunas funciones relacionales.

- Las instrucciones utilizadas para consultar gráficos Neo4j en Cifrado suelen tener un aspecto similar al siguiente:

```
$ MATCH [some set of nodes and/or  
relationships]  
WHERE [some set of properties holds]  
RETURN [some set of results captured  
by the MATCH and WHERE clauses]
```

Cláusulas Neo4j CQL

A continuación se presentan las cláusulas de lectura de Neo4j CQL

cláusulas	Uso
MATCH	Esta cláusula se utiliza para buscar los datos con un patrón especificado.
OPTIONAL MATCH	Esto es lo mismo que MATCH, la única diferencia es que puede usar nulos en caso de que falten partes del patrón.
WHERE	Este identificador de cláusula se utiliza para agregar contenido a las consultas CQL.
START	Esta cláusula se utiliza para encontrar los puntos de partida a través de los índices heredados.
LOAD CSV	Esta cláusula se utiliza para importar datos de archivos CSV.

Cláusula de escritura	Uso
CREATE	Se utiliza para crear nodos, relaciones y propiedades.
MERGE	Verifica si el patrón especificado existe en el grafo. Si no, crea el patrón.
SET	Se utiliza para actualizar etiquetas en nodos, propiedades en nodos y relaciones.
DELETE	Se utiliza para eliminar nodos y relaciones o rutas, etc. del grafo.
REMOVE	Se utiliza para eliminar propiedades y elementos de nodos y relaciones.
FOREACH	Se utiliza para actualizar los datos de una lista.
CREATE UNIQUE	Usando las cláusulas CREATE y MATCH, puede obtener un patrón único haciendo coincidir el patrón existente y creando el que falta.

- Las siguientes son las cláusulas de escritura de Neo4j CQL

- A continuación se presentan las cláusulas generales de Neo4j **CQL**

Cláusulas generales	Uso
RETURN	Se utiliza para definir qué incluir en el conjunto de resultados de la consulta.
ORDER BY	Se utiliza para organizar la salida de una consulta en orden. Se utiliza junto con las cláusulas RETURN o WITH .
LIMIT	Se utiliza para limitar las filas del resultado a un valor específico.
SKIP	Se utiliza para definir desde qué fila comenzar, incluidas las filas en la salida.
WITH	Se utiliza para encadenar las partes de consulta.
UNWIND	Se utiliza para expandir una lista en una secuencia de filas.
UNION	Se utiliza para combinar el resultado de varias consultas.
CALL	Se utiliza para invocar un procedimiento implementado en la base de datos.

Funciones Neo4j CQL

Las siguientes son las funciones Neo4j CQL de uso frecuente:

Funciones CQL	Uso
String	Se utilizan para trabajar con literales string.
Aggregation	Se utilizan para realizar algunas operaciones de agregación en los resultados de la consulta CQL.
Relationship	Se utilizan para obtener detalles de relaciones como startnode, endnode, etc.

Tipos de datos Neo4j CQL

- Los tipos de datos son similares al lenguaje Java. Se utilizan para definir las propiedades de un nodo o una relación.
- Neo4j CQL admite los siguientes tipos de datos:

Tipo de datos CQL	Uso
Boolean	Representa literales booleanos: verdadero, falso.
byte	Se utiliza para representar enteros de 8 bits.
short	Se utiliza para representar enteros de 16 bits.
Int	Se utiliza para representar enteros de 32 bits.
long	Se utiliza para representar enteros de 64 bits.
float	Se utiliza para representar números de coma flotante de 32 bits.
double	Se utiliza para representar números de coma flotante de 64 bits.
char	Se utiliza para representar caracteres de 16 bits.
String	Se utiliza para representar Strings.

Operadores CQL

- A continuación se muestra la lista de operadores compatibles con Neo4j CQL

Tipo	Operadores
Matemático	+, -, *, /, %, ^
Comparación	+, <>, <, >, <=, >=
Booleano	AND, OR, XOR, NOT
String	+
Lista	+, IN, [X], [X..... Y]
Expresión regular	=-
Coincidencia de cadenas	STARTS WITH, ENDS WITH, CONSTRAINTS

Operadores booleanos en Neo4j CQL

- Neo4j admite los siguientes operadores booleanos para usar en la cláusula WHERE para admitir múltiples condiciones.

Operadores booleanos	Descripción
AND	Es como el operador SQL AND.
OR	Es como el operador SQL OR.
NOT	Es como el operador SQL NOT.
XOR	Es una palabra clave de Neo4j CQL para apoyar la operación XOR.

Operadores de comparación en Neo4j CQL

- Neo4j admite los siguientes operadores de comparación para usar en la cláusula WHERE para admitir condiciones.

Operadores booleanos	Descripción
=	"Igual A".
< >	"No Igual A".
<	"Menor Que".
>	"Mayor Que".
<=	"Menor o Igual que".
> =	"Mayor o igual que".

Neo4j CQL

- Creación de nodos

Un nodo es un dato/registro en una base de datos de grafos.

Se utiliza la cláusula CREATE para crear:

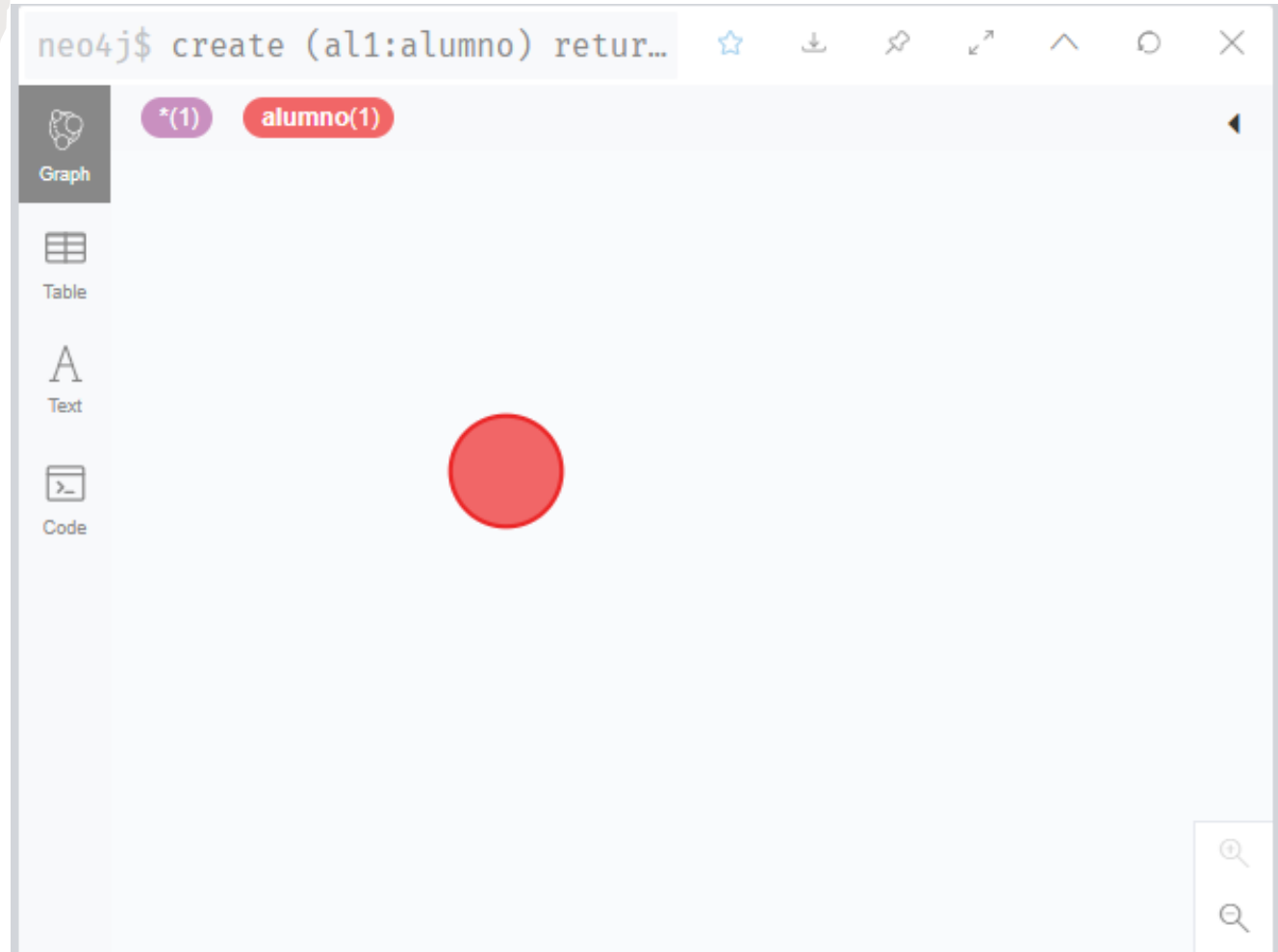
- Crear un solo nodo
- Crear varios nodos
- Crear un nodo con una etiqueta
- Crear un nodo con varias etiquetas
- Crear un nodo con propiedades
- Devolver el nodo creado

Creación de un solo nodo

- Puede crear un nodo en Neo4j especificando el nombre del nodo que se va a crear dentro de la cláusula CREATE.
- A continuación se muestra la sintaxis para crear un nodo mediante el lenguaje de consulta Cypher.

```
CREATE (node_name)
```

```
$ CREATE (al1:alumno)
```



- Las propiedades son los pares clave-valor con los que un nodo almacena datos. Puede crear un nodo con propiedades mediante la cláusula CREATE. Debe especificar estas propiedades separadas por comas dentro de llaves "{}".

```
CREATE (node:label {
key1: value, key2: value,
.  })
```

```
CREATE (a12:alumno{noboleta:
2020223, nombre:'Juan',
apellidos:'Pérez López',
genero:'M', fecha_nac:'03-
08-2000'})
```

- Para comprobar la creación del tipo de nodo, se ejecuta la siguiente consulta en el símbolo del sistema en dólares.

```
MATCH (n) RETURN n
```

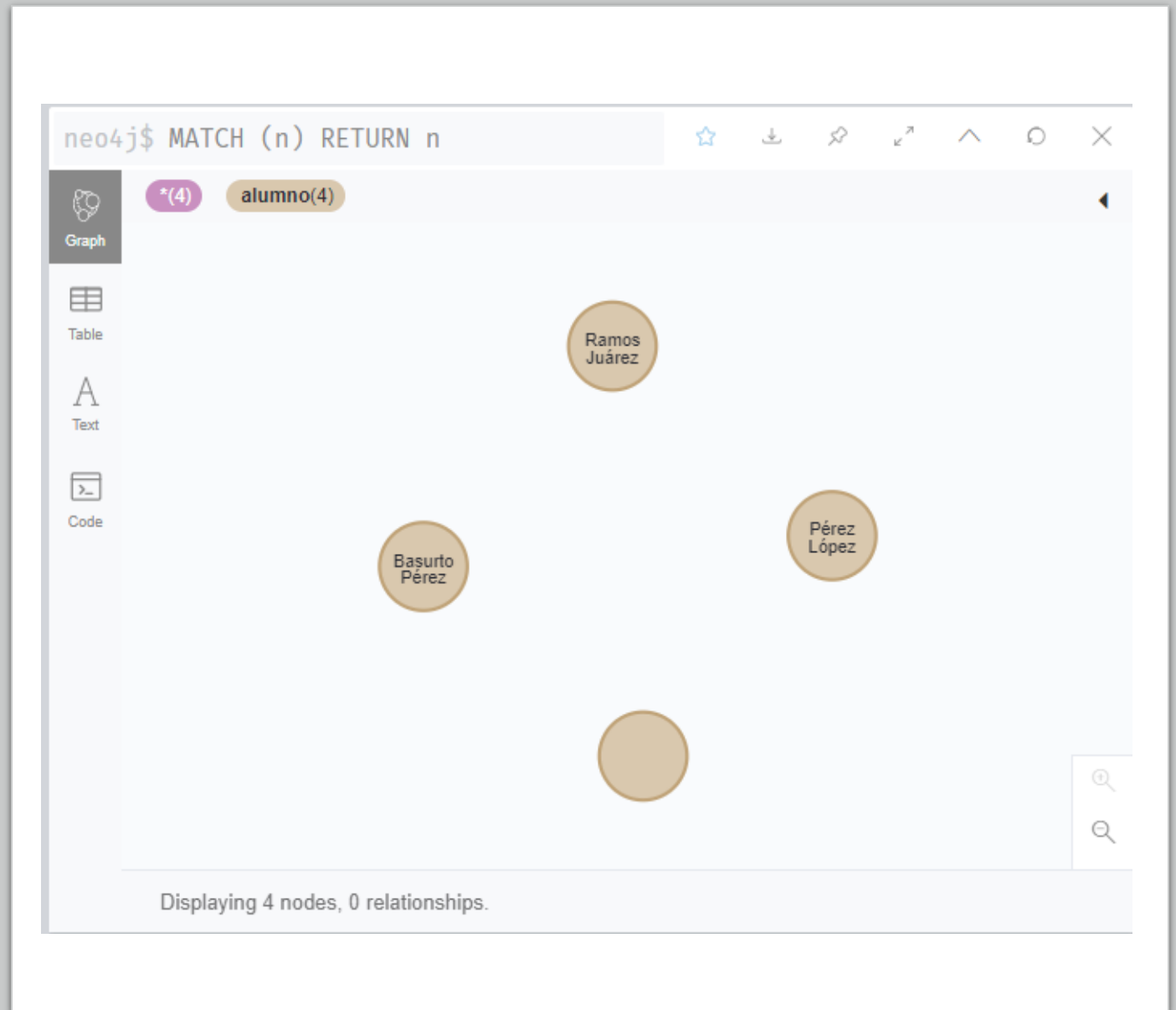
- Esta consulta devuelve todos los nodos de la base de datos.

The screenshot shows the Neo4j desktop application. At the top, a command bar contains the query `neo4j$ match (n) return n`. Below the command bar, there are two tabs: `*(2)` (active) and `alumno(2)`. The main area displays a graph view with two nodes. The top node is a brown circle labeled "Pérez López". The bottom node is a plain brown circle. On the left side, there is a sidebar with icons for "Graph", "Table", "Text", and "Code". At the bottom of the window, a status bar indicates "Displaying 2 nodes, 0 relationships."

Creación de varios nodos

- La cláusula CREATE también se utiliza para crear varios nodos al mismo tiempo. Para hacerlo, debe pasar los nombres de los nodos que se crearán, separados por una coma.

```
CREATE (nodo1), (nodo2)  
$ CREATE  
(a13:alumno{noboleta:2023  
456, nombre:'Martha',  
apellidos:'Ramos Juárez',  
genero:'F'}), (a14:alumno  
{noboleta:2021678, nombre  
:'María', apellidos:'Basu  
rto Pérez', genero:'F',  
fecha_nac:'15/03/2020'})
```



Neo4j CQL - Creando una relación

- En Neo4j, una relación es un elemento mediante el cual conectamos dos nodos de un grafo. Estas relaciones tienen patrones de dirección, tipo y forma de datos. Utilizando la cláusula CREATE podemos:
 - Crear relaciones
 - Crear una relación entre los nodos existentes
 - Crear una relación con la etiqueta y las propiedades
- Creación de relaciones con nodos no existentes
 - Especificaremos la relación dentro de los corchetes "[]" dependiendo de la dirección de la relación se coloca entre el guión "-" y la flecha "->" como se muestra en la sintaxis para crear una relación mediante la cláusula CREATE.

```
CREATE (node1)-[label:RelationshipType]->(node2)
```

```
CREATE (a13:alumno{noboleta:2020,nombre:'Isela',  
apellidos:'Martínez López', genero:'F', fecha_nac:'19/09/2020'}) -  
[amistad:es amigo] -  
>(a14:alumno{noboleta:20198765,nombre:'Martín',apellidos:'Pérez  
Cañedo', genero:'M'})
```

Creación de una relación entre nodos existentes

- También puede crear una relación entre nodos existentes mediante la cláusula MATCH.
- A continuación se encuentra la sintaxis para crear una relación mediante la cláusula MATCH.

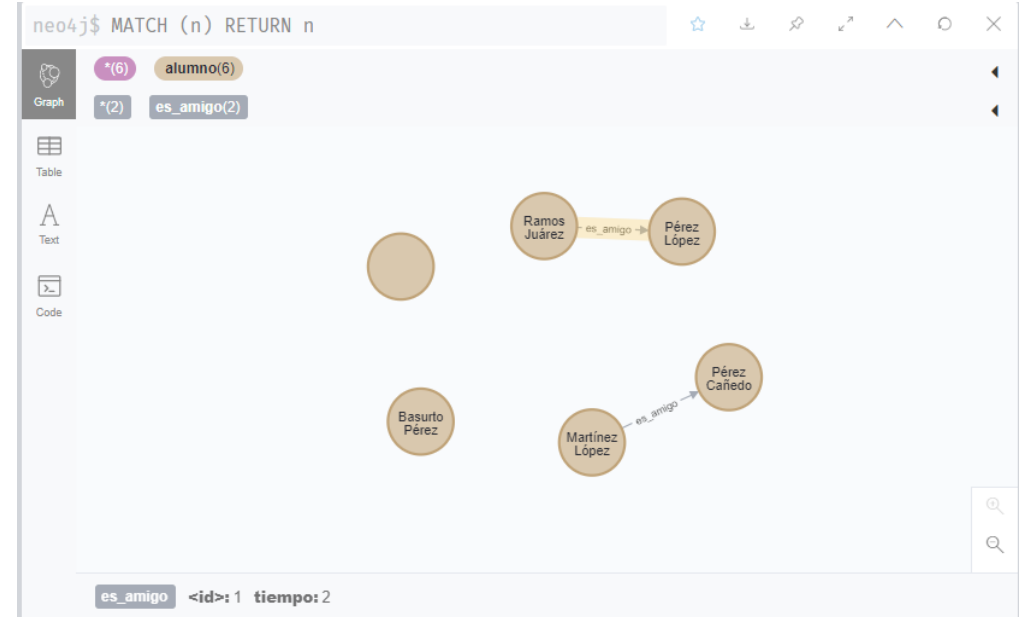
```
MATCH (a:LabeofNode1), (b:LabeofNode2)
  WHERE a.name = "nameofnode1" AND b.name =
    " nameofnode2"
```

```
CREATE (a)-[: Relation]->(b)
RETURN a,b
```

- Por ejemplo:

```
MATCH (x:alumno), (y:alumno)
where x.noboleta = 2023456 and y.noboleta
= 2020223
```

```
CREATE (x)-[amistad:es_amigo{tiempo: 2}
]->(y) RETURN x,y
```



Creación de una relación con etiqueta y propiedades

- Puede crear una relación con etiqueta y propiedades mediante la cláusula CREATE.
- A continuación se encuentra la sintaxis para crear una relación con etiqueta y propiedades mediante la cláusula CREATE.

```
CREATE (node1)-[label:Rel_Type {key1:value1,  
key2:value2, . . . n}]-> (node2)
```

Creación de una ruta completa

- En Neo4j, se forma una ruta utilizando relaciones continuas. Se puede crear una ruta de acceso mediante la cláusula CREATE.
- A continuación se encuentra la sintaxis para crear una ruta de acceso en Neo4j utilizando la cláusula CREATE.

```
CREATE p = (Node1 {properties}) -  
[:Relationship_Type]->  
      (Node2 {properties})[:Relationship_Type]-  
>(Node3 {properties})  
RETURN p
```

- Ejemplo

```
MATCH (x:alumno),(y:alumno)
WHERE x.noboleta IS NOT
NULL and y.noboleta IS NOT
NULL and x.noboleta <> y.noboleta
```

```
CREATE (x) -[:es_compañero]->
(y) RETURN x
```

- Resultado de la ejecución de la sentencia anterior
- Para obtener todos los nodos con los cuales esté relacionado:

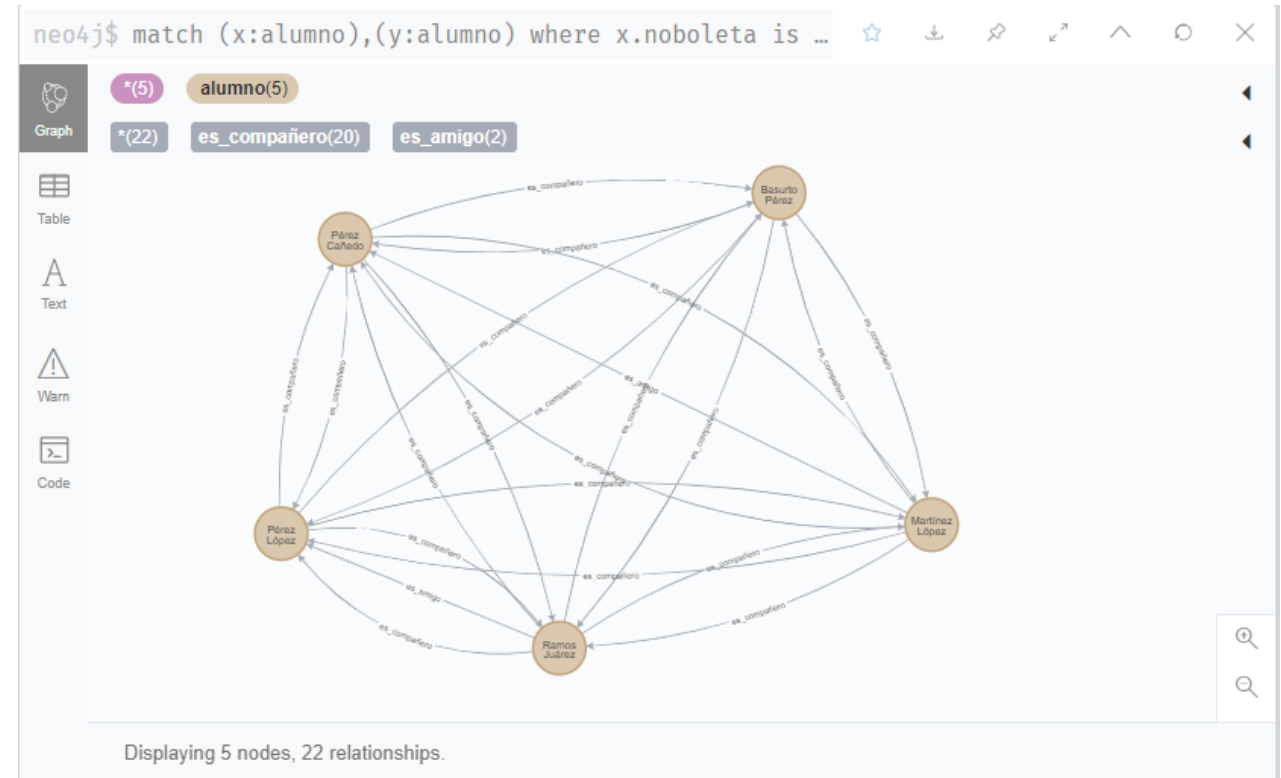
```
MATCH (x:label{propiedades}) --
> (n) RETURN n
```

- Ejemplo

```
MATCH (a:alumno{nombre:'Juan'}) -
-> (x) RETURN x
```

- Para obtener las distintas relaciones

```
MATCH () -[r]- () RETURN DISTINCT
TYPE(r)
```



Cláusula Where

- Al igual que SQL, Neo4j CQL ha proporcionado la cláusula WHERE en el comando MATCH para filtrar los resultados de una consulta.
- A continuación se encuentra la sintaxis de la cláusula WHERE.

```
MATCH (label)
WHERE label.property = "value"
RETURN label
```

Ejemplo

```
MATCH (a:alumno) WHERE a.fecha_nac is null
RETURN a
```

Cláusula WHERE con múltiples condiciones

- También puede utilizar la cláusula WHERE para verificar varias condiciones.

```
MATCH (emp:Employee)
WHERE emp.name = 'Abc' AND emp.name = 'Xyz'
RETURN emp
```

Ejemplo

```
MATCH (a:alumno) WHERE a.fecha_nac IS NULL AND
a.genero='F' RETURN a
MATCH (a:alumno) WHERE a.fecha_nac IS NULL OR
a.genero='F' RETURN a
```

Combinar un nodo con etiqueta

- Puede combinar un nodo de la base de datos en función de la etiqueta mediante la cláusula MERGE. Si intenta combinar un nodo basado en la etiqueta, Neo4j verifica si existe algún nodo con la etiqueta dada. De lo contrario, se creará el nodo actual.
- A continuación se encuentra la sintaxis para combinar un nodo basado en una etiqueta.

```
MERGE (node:label{properties . . . }) RETURN node
```

- **Ejemplo**

```
MERGE (y:materia{id:1,nombre:'Ingeniería de Software',creditos:8}) RETURN y
```

```
MERGE (y:materia{id:2,nombre:'Inteligencia Artificial',creditos:10}) RETURN y
```

Combinar un nodo con propiedades

- También se puede combinar un nodo con un conjunto de propiedades. Neo4j busca una coincidencia igual para el nodo especificado, incluidas las propiedades. Si no encuentra ninguno, crea uno.

```
MERGE (node:label {key1:value, key2:value,  
key3:value . . .})
```

- Ejemplo

```
MERGE (x:alumno{genero:'F'}) RETURN x
```

OnCreate y OnMatch

- Cada vez que ejecutamos una consulta de combinación, se busca coincidencia o se crea un nodo. Mediante on create y on match, se pueden establecer propiedades para indicar si el nodo se creó o se encontró.
- A continuación se indica la sintaxis de las cláusulas OnCreate y OnMatch.

```
MERGE (node:label {properties . . . . .  
. . . . })
```

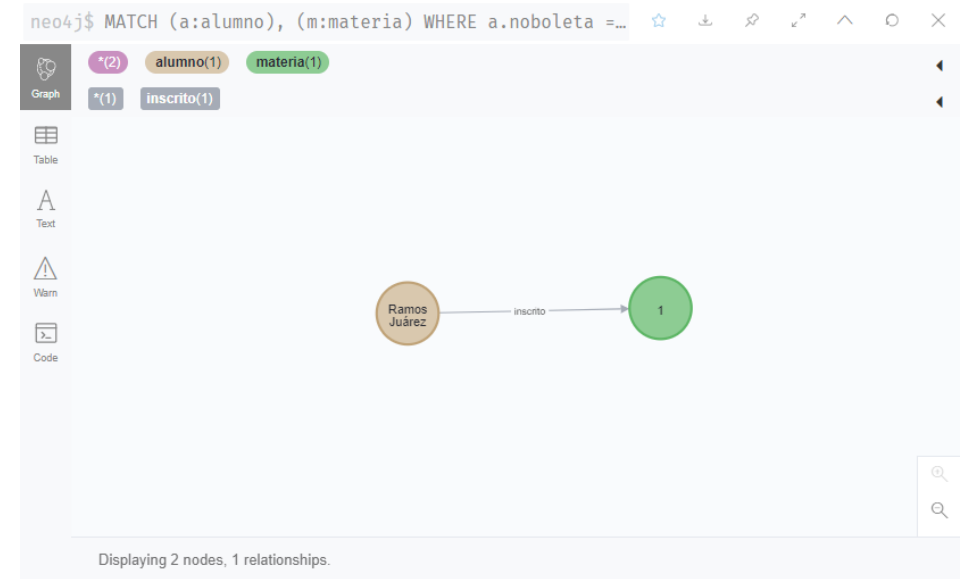
```
ON CREATE SET property.isCreated ="true"
```

```
ON MATCH SET property.isFound ="true"
```

Combinar una relación

- Al igual que los nodos, también pueden combinarse las relaciones utilizando la cláusula MERGE.

```
MATCH (a:alumno), (m:materia) WHERE a.noboleta = 2023456 AND m.id = 1
MERGE (a)-[r:inscrito]->(m) RETURN a, m.nombre
```



Cláusula Set

- Con la cláusula SET, puede agregar nuevas propiedades a un nodo o relación existente, y también agregar o actualizar valores de propiedades existentes.
- Establecer una propiedad
- Mediante la cláusula SET, puede crear una nueva propiedad en un nodo.

```
MATCH
(node:label{properties...})
SET node.property = value
RETURN node
```

- Ejemplo

```
MATCH (a:alumno) WHERE
a.fecha_nac IS NULL AND
a.genero='F'
SET a.fecha_nac='17/6/2000'
RETURN a
```

neo4j\$ match (a:alumno) where a.fecha_nac is null and a...

*(1) alumno(1)

Graph

Table

Text

Code

Ramos Juárez

alumno <id>: 0 apellidos: Ramos Juárez fecha_nac: 17/6/2000 genero: F noboleta: 2023456 nombre: Martha

Establecer varias propiedades

- De la misma manera, puede crear varias propiedades en un nodo utilizando la cláusula SET. Para ello, debe especificar los pares de valores clave con comas.

```
MATCH (node:label {properties})  
SET node.property1 = value, node.property2 =  
value  
RETURN node
```

- **Ejemplo**

```
MATCH (a:alumno) WHERE a.noboleta is null SET  
a.nombre='José Juan', a.apellidos='Rocha  
Segura', a.genero='M' RETURN a
```


Establecer una etiqueta en un nodo

- Puede establecer una etiqueta en un nodo existente mediante la cláusula SET.

```
MATCH (n {properties . . . })
```

```
SET n :label
```

```
RETURN n
```

- **Ejemplo**

```
MATCH (x:materia{id:1}) SET x  
:asignatura RETURN x
```

Configuración de varias etiquetas en un nodo

- Puede establecer varias etiquetas en un nodo existente mediante la cláusula SET. Aquí debe especificar las etiquetas separándolas con dos puntos ":"

```
MATCH (n {properties . . . })
```

```
SET n :label1:label2
```

```
RETURN n
```

Eliminación de todos los nodos y relaciones

- Para eliminar todos los nodos y las relaciones de la base de datos se usa la cláusula DELETE.

```
MATCH (n) DETACH DELETE n
```

- Esto eliminará todos los nodos y relaciones de la base de datos y la vaciará.

Eliminación de una propiedad

- Se puede quitar una propiedad existente pasándole NULL como valor a la propiedad de un nodo mediante la cláusula SET.

```
MATCH (node:label {properties})
```

```
SET node.property = NULL
```

```
RETURN node
```

- Para la eliminación de las relaciones entre nodos, hay que realizar la búsqueda que coincida con el criterio y posteriormente eliminar el conjunto de resultado
- Ejemplo

```
MATCH (x:alumno) - [y:es_compañero] -> (:alumno)
```

```
DETACH DELETE y
```

- La opción DETACH permite "desligar" los nodos relacionados para proceder a eliminar la relación

Eliminación de un nodo en particular

- Para eliminar un nodo en particular, se deben especificar los detalles del nodo para poder eliminarlo.

```
MATCH (node:label {properties . . . . })
```

```
DETACH DELETE node
```

Si el nodo tiene una relación no será posible eliminarlo hasta eliminar la relación que mantiene, por eso el uso de DETACH

Cláusula REMOVE

- La cláusula REMOVE se utiliza para eliminar propiedades y etiquetas de elementos de los grafos (Nodos o Relaciones).
 - La principal diferencia entre los comandos DELETE y REMOVE es que la operación DELETE se utiliza para eliminar nodos y relaciones asociadas, mientras que la operación REMOVE se utiliza para quitar etiquetas y propiedades.
- Eliminación de una propiedad
 - Se puede quitar una propiedad de un nodo mediante MATCH junto con la cláusula REMOVE.

```
MATCH (node:label{properties . . . })  
REMOVE node.property  
RETURN node
```

- **Ejemplo**

```
MATCH (m:materia) WHERE m.id=3 REMOVE m.creditos  
RETURN m
```

Eliminación de una etiqueta de un nodo

- Al igual que la propiedad, también puede quitar una etiqueta de un nodo existente mediante la cláusula REMOVE.

```
MATCH (node:label {properties . . . })
```

```
REMOVE node :label
```

```
RETURN node
```

- **Ejemplo**

```
MATCH (x:materia{id:1})
```

```
REMOVE x :asignatura RETURN x
```


Eliminación de varias etiquetas

- También puede quitar varias etiquetas de un nodo existente.

```
MATCH (node:label1:label2 {properties . . . })
```

```
REMOVE node :label1:label2
```

```
RETURN node
```

Obtener todos los nodos bajo una etiqueta específica

- Usando la cláusula MATCH, se pueden obtener todos los nodos bajo una etiqueta específica.

```
MATCH (node:label)
```

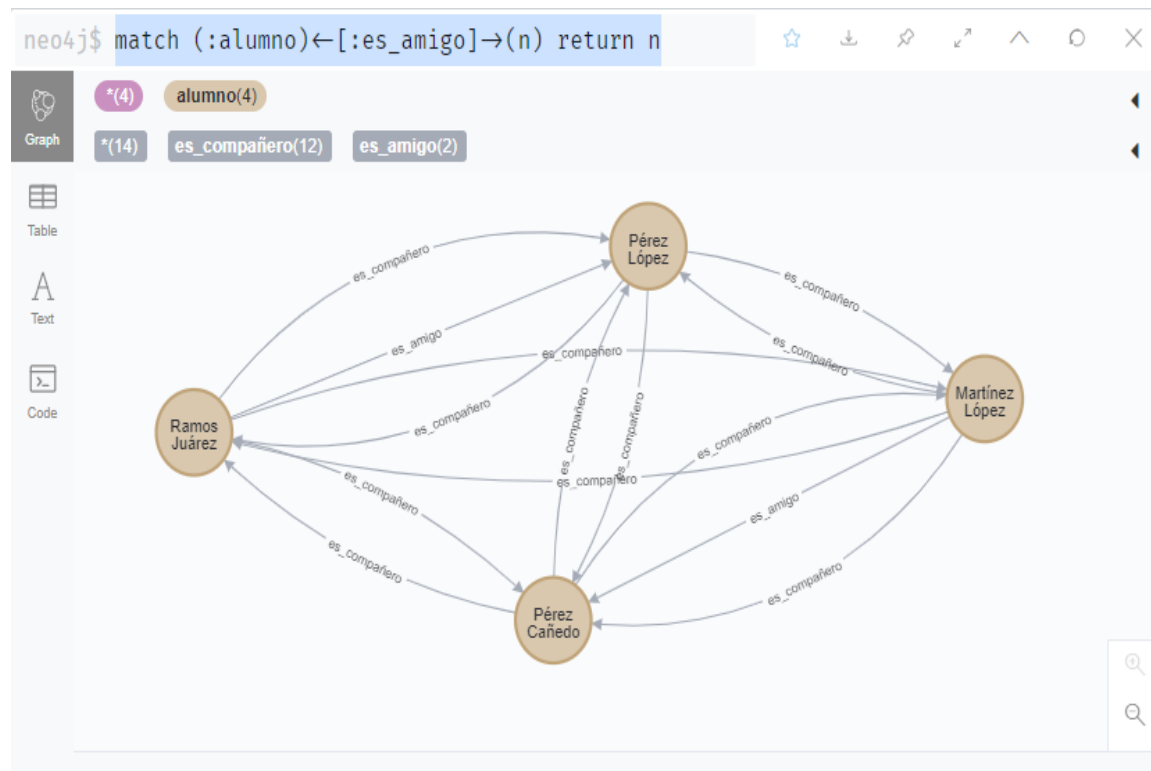
```
RETURN node
```

- **Ejemplo**

```
MATCH (a:alumno)
```

```
RETURN a
```

MATCH por relación



- Puede recuperar nodos basados en la relación mediante la cláusula MATCH.

```
MATCH (node:label)<-[: Relationship]-(n)
RETURN n
```

```
MATCH (node:label)-[: Relationship]->(n)
RETURN n
```

```
MATCH (node:label)<-[: Relationship]->(n)
RETURN n
```

- **Ejemplo**

```
match (:alumno)<-[:es_amigo]-(n) return n
match (:alumno)-[:es_amigo]->(n) return n
match (:alumno)<-[:es_amigo]->(n) return n
```

COUNT

- La función **count()** se utiliza para contar el número de nodos.

```
MATCH (label:node{properties...})  
RETURN count(label)
```

- Ejemplo

```
MATCH(a:alumno) RETURN COUNT(a)
```

- La cláusula COUNT también se utiliza para contar los grupos de tipo relación

- Ejemplo

```
MATCH ()-[r:'es_compañero']-() RETURN COUNT(r)
```

- Para contar las distintas relaciones que hay

```
MATCH ()-[r]-() RETURN DISTINCT TYPE(r), COUNT(r)
```

cláusula RETURN

- Se puede devolver un nodo mediante la cláusula RETURN.

```
Create (node:label {properties})  
RETURN node [, node...]
```

- También puede devolver relaciones

```
CREATE (node1)-[Relationship:Relationship_type]-(node2)  
RETURN Relationship
```

- También puede devolver propiedades

```
Match (node:label {properties . . . . . })  
Return node.property
```

- Puede devolver todos los elementos de la base de datos

```
Match (node:label {properties . . . . . })  
Return *
```

- Puede devolver una propiedad en particular con alias

```
Match (node:label {properties . . . . . })  
Return node.property AS Alias
```

cláusula ORDER BY

- Puede organizar los datos de resultados en orden utilizando la cláusula ORDER BY.

MATCH (n)

RETURN n.property1,
n.property2

ORDER BY n.property [,
n.property...] [DESC]

- Ejemplo

```
MATCH (a:alumno) RETURN a
ORDER BY a.apellidos
```

```
neo4j$ match (a:alumno) return a order by a.apellidos
```



Graph



Table



Text



Code

"a"

{"apellidos":"Basurto Pérez","fecha_nac":"15/03/2020","noboleta":2021678,"nombre":"María","genero":"F"}

{"apellidos":"Martínez López","fecha_nac":"19/09/2020","noboleta":2020,"nombre":"Isela","genero":"F"}

{"apellidos":"Pérez Cañedo","noboleta":20198765,"nombre":"Martín","genero":"M"}

{"apellidos":"Pérez López","fecha_nac":"03-08-2000","noboleta":2020223,"nombre":"Juan","genero":"M"}

{"apellidos":"Ramos Juárez","fecha_nac":"17/6/2000","noboleta":2023456,"nombre":"Martha","genero":"F"}

{"apellidos":"Rocha Segura","nombre":"José Juan","genero":"M"}

MAX COLUMN WIDTH:

Cláusula **LIMIT**

- La cláusula **LIMIT** se utiliza para limitar el número de nodos en la salida.

MATCH (n)

RETURN n

ORDER BY n.property

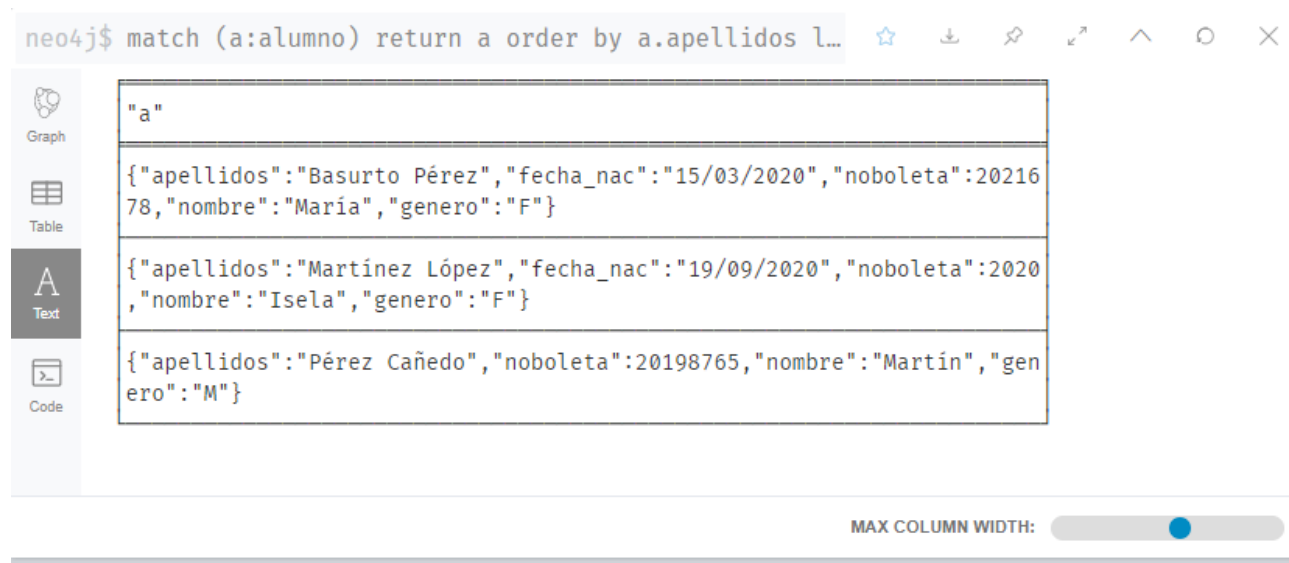
LIMIT size

- Ejemplo

MATCH (a:alumno)

RETURN a ORDER

BY a.apellidos LIMIT 3



The screenshot shows the Neo4j Cypher query editor interface. At the top, the query is entered: `neo4j$ match (a:alumno) return a order by a.apellidos l...`. Below the query, the results are displayed in a table view. The table has four rows, each representing a node 'a'. The first row is a simple string '"a"'. The subsequent three rows are JSON objects containing properties like 'apellidos', 'fecha_nac', 'noboleta', 'nombre', and 'genero'. The results are ordered by the 'apellidos' property. On the left side of the interface, there are icons for 'Graph', 'Table', 'Text', and 'Code', with 'Table' currently selected. At the bottom right, there is a slider for 'MAX COLUMN WIDTH:'.

"a"
{"apellidos":"Basurto Pérez","fecha_nac":"15/03/2020","noboleta":2021678,"nombre":"María","genero":"F"}
{"apellidos":"Martínez López","fecha_nac":"19/09/2020","noboleta":2020,"nombre":"Isela","genero":"F"}
{"apellidos":"Pérez Cañedo","noboleta":20198765,"nombre":"Martín","genero":"M"}

Cláusula SKIP

- La cláusula SKIP se utiliza para definir desde qué nodo comenzar, incluyendo los nodos en la salida.

MATCH (n)

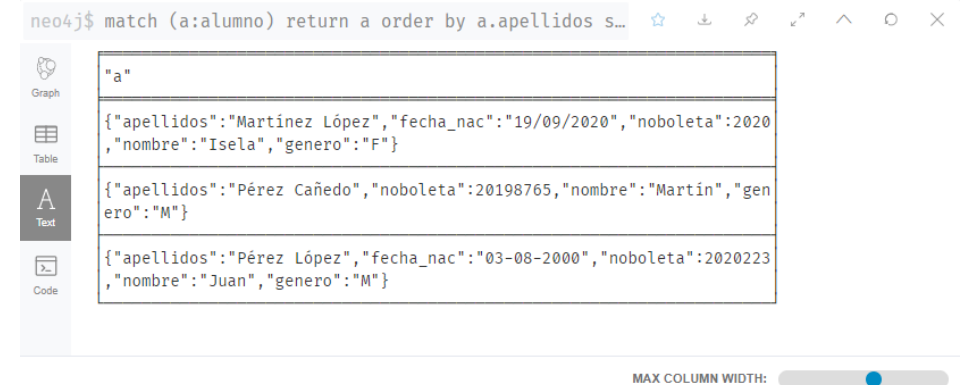
RETURN n

ORDER BY n.property

SKIP size

- Ejemplo

```
MATCH (a:alumno) RETURN a ORDER  
BY a.apellidos SKIP 1 LIMIT 3
```



Funciones de cadena de texto

- Al igual que SQL, CQL proporciona un conjunto de funciones String para usarlas en consultas para obtener los resultados requeridos.

Función	Descripción
UPPER	Se utiliza para cambiar todas las letras en mayúsculas
LOWER	Se utiliza para cambiar todas las letras en minúsculas.
SUBSTRING	Se utiliza para obtener la subcadena de una cadena dada.
REPLACE	Se utiliza para reemplazar una subcadena con una subcadena dada de una cadena.

Funciones de agregación

- CQL proporciona algunas funciones de agregación para usar en la cláusula RETURN. Es similar a la cláusula GROUP BY en SQL.
- Se usa el comando RETURN + Aggregation Functions en MATCH para trabajar en un grupo de nodos y devolver algún valor agregado.

Función	Descripción
COUNT	Devuelve el número de nodos devueltos por el comando MATCH.
MAX	Devuelve el valor máximo de un conjunto de nodos devueltos por el comando MATCH.
MIN	Devuelve el valor mínimo de un conjunto de nodos devueltos por el comando MATCH.
SUMA	Devuelve el valor de suma de todos los nodos devueltos por el comando MATCH.
AVG	Devuelve el valor medio de todos los nodos devueltos por el comando MATCH.

Índices

- Neo4j admite índices en propiedades de nodo o relación para mejorar el rendimiento de la aplicación. Podemos crear índices en propiedades para todos los nodos, que tienen el mismo nombre de etiqueta.
- Podemos usar estas columnas indexadas en el operador MATCH o WHERE o IN para mejorar la ejecución del comando CQL.

- Creación de un índice

- CQL proporciona el comando "CREATE INDEX" para crear índices en las propiedades Node o Relationship.

```
CREATE INDEX ON :label (node)
```

- Proporciona el comando "DROP INDEX" para eliminar índices en las propiedades Node o Relationship.

```
DROP INDEX ON :label (node)
```

Restricción UNIQUE

- El comando CREATE siempre crea un nuevo nodo o relación, lo que significa que aunque use los mismos valores, inserta una nueva fila. Para evitar la duplicidad, se deben utilizar algunas restricciones de base de datos para crear una regla sobre una o más propiedades de un nodo o relación.
- Al igual que SQL, Neo4j también admite la restricción UNIQUE en las propiedades de nodo o relación. La restricción UNIQUE se utiliza para evitar registros duplicados y para hacer cumplir la regla de integridad de datos.
- Crear restricción UNIQUE
 - CQL proporciona el comando "CREATE CONSTRAINT" para crear restricciones únicas en las propiedades de nodo o relación.

```
CREATE CONSTRAINT ON (a:alumno) ASSERT a.noboleta IS UNIQUE
```

- Para eliminar la restricción:

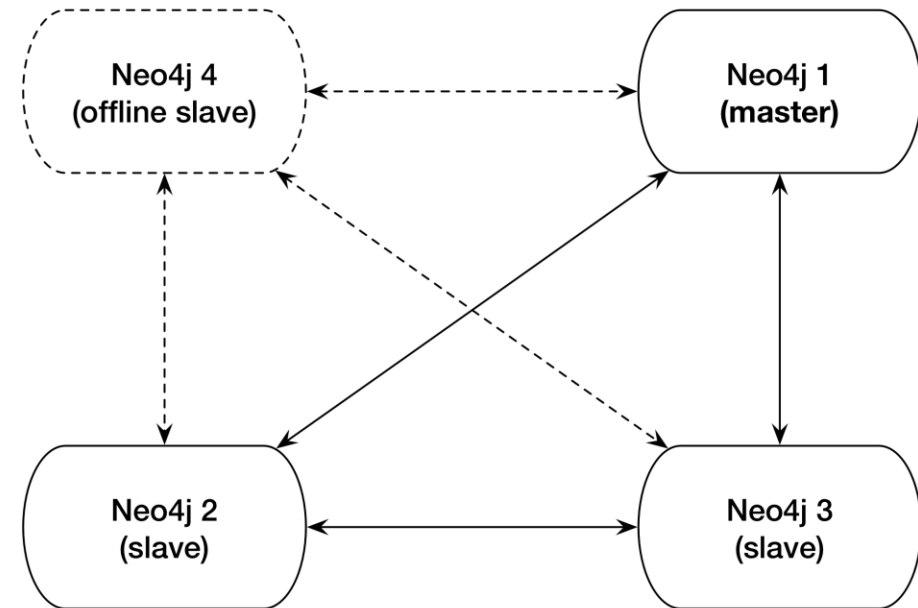
```
DROP CONSTRAINT ON (a:alumno) ASSERT a.noboleta IS UNIQUE
```

Alta disponibilidad

- El modo de alta disponibilidad es la respuesta de Neo4j a la pregunta: "¿Puede escalar una base de datos de grafos?" Sí, pero con algunas salvedades.
- Una escritura a un esclavo no está inmediatamente sincronizada con todos los demás esclavos, por lo que existe el peligro de perder la consistencia (en el sentido CAP) por un breve momento (lo que la hace eventualmente consistente).
- HA perderá transacciones puras compatibles con ACID. Es por esta razón que Neo4j HA se promociona como una solución en gran medida para aumentar la capacidad de lectura.
 - Al igual que MongoDB, los servidores del clúster elegirán un maestro que tenga la responsabilidad primaria de administrar la distribución de datos en el clúster.
 - Sin embargo, a diferencia de MongoDB, los esclavos en Neo4j aceptan escrituras.
 - Las escrituras esclavas se sincronizarán con el nodo maestro, que luego propagará esos cambios a los otros esclavos.

Cluster HA (High Availability)

- Para usar Neo4j con HA, primero se debe configurar un clúster.
- Anteriormente, Neo4j usaba clústeres en ZooKeeper como un mecanismo de coordinación externa, que funcionaba bien pero requería mucha administración adicional, ya que ZooKeeper tendría que ejecutarse por separado. Eso ha cambiado en versiones más recientes.
- Ahora, los clústeres Neo4j son autogestionados y auto coordinados.
- Los clústeres pueden elegir su propia configuración maestro/esclavo y volver a coordinarse cuando los servidores se desconectan.



Los nodos 1, 2 y 3 están actualmente en línea y se replican entre sí correctamente, mientras que el nodo 4 está fuera de línea.

Cuando el nodo 4 vuelva a estar en línea, volverá a entrar como nodo esclavo.

Si el nodo 1, el nodo maestro actual, se desconecta, los otros nodos elegiría automáticamente un líder (sin la ayuda de un servicio de coordinación externo).

- Esta es una configuración de ALTA disponibilidad bastante estándar en la industria hoy en día, y los ingenieros detrás de Neo4j han hecho a los administradores un gran servicio al habilitar una configuración de alta disponibilidad sin ZooKeeper.

En los clústeres de HA Neo4j, la elección del maestro ocurre automáticamente. Si el servidor maestro se desconecta, otros servidores notarán y elegirán un líder de entre ellos.

Iniciar el servidor maestro anterior lo agregará de nuevo al clúster, pero ahora el antiguo maestro seguirá siendo un esclavo (hasta que otro servidor se caiga).

La alta disponibilidad permite que los sistemas de lectura muy pesada se ocupe de replicar grafos en múltiples servidores y, por lo tanto, balancear la carga.

Aunque el clúster en su conjunto solo es consistente eventualmente, hay trucos que puede aplicar para reducir la posibilidad de leer datos obsoletos en sus propias aplicaciones, como asignar una sesión a un servidor.

Con las herramientas adecuadas, la planificación y una buena configuración, puede crear una base de datos de grafos lo suficientemente grande como para manejar miles de millones de nodos y vértices, y casi cualquier número de solicitudes que se puedan necesitar.

Simplemente agregue copias de seguridad periódicas y tendrá la receta para un sistema de producción sólido.

Copias de seguridad

Las copias de seguridad son un aspecto necesario de cualquier base de datos.

Aunque las copias de seguridad se integran de manera efectiva cuando se utiliza la replicación en un grupo de alta disponibilidad, las copias de seguridad periódicas (nocturnas, semanales, por hora, etc.) que se almacenan sin sitio siempre son una buena idea para la recuperación ante desastres.

- Neo4j Enterprise ofrece una herramienta llamada neo4j-admin que realiza una amplia variedad de acciones, incluidas las copias de seguridad.

El método más eficaz al ejecutar un servidor de alta disponibilidad es crear un comando de copia de seguridad completa para copiar el archivo de base de datos del clúster a un archivo con estampa de tiempo en una unidad montada.

- Apuntar la copia a todos los servidores del clúster permitirá obtener los datos más recientes disponibles.
- El directorio de copia de seguridad creado es una copia totalmente utilizable.
- Si se necesita recuperarse de un fallo, se reemplaza el directorio de datos de cada instalación con el directorio de copia de seguridad, y estará listo para comenzar.
- Se debe comenzar con una copia de seguridad completa.



Comentarios sobre Neo4j

Neo4j es una implementación de código abierto de clase superior (relativamente rara) de bases de datos de grafos.

Las bases de datos de grafos se centran en las relaciones entre los datos, en lugar de los puntos en común entre los valores.

Modelar datos basado en grafos es simple.

- Solo se crean nodos y las relaciones entre ellos y, opcionalmente, pares clave-valor para ellos.
- Consultar es tan fácil como declarar cómo recorrer el grafo desde un nodo inicial.

Fortalezas

Las bases de datos de grafos son perfectas para datos no estructurados, en muchos sentidos, incluso más que las bases de datos de documentos.

- Neo4j no solo es sin tipos ni esquemas, sino que no impone restricciones sobre cómo se relacionan los datos.
- Actualmente, Neo4j puede soportar 34.4 mil millones de nodos y 34.4 mil millones de relaciones, lo cual es más que suficiente para la mayoría de los casos de uso.
- Las distribuciones Neo4j proporcionan varias herramientas para búsquedas rápidas con Lucene, el lenguaje de consulta Cypher y la interfaz REST.

Más allá de la facilidad de uso, Neo4j es rápido.

- A diferencia de las operaciones de unión en bases de datos relacionales o las operaciones de map-reduce en otras bases de datos, los recorridos de grafos son en un tiempo constante.
- Los datos están a solo un paso entre nodos, en lugar de unir valores y filtrar los resultados deseados, que es cómo funcionan la mayoría de las bases de datos.
- No importa cuán grande sea el grafo; pasar del nodo A al nodo B es siempre a un paso si comparten una relación.

Por último, la edición Enterprise ofrece sitios de alta disponibilidad y alto tráfico de lectura a través de Neo4j HA.

Debilidades de Neo4j



Aunque la Community Edition es GPL, probablemente se necesitará comprar una licencia si desea ejecutar un entorno de producción utilizando las herramientas Enterprise (que incluyen HA y copias de seguridad).

Actualmente no se puede fragmentar subgrafos, lo que todavía pone un límite en el tamaño del grafo.

Aunque HA es excelente en la replicación, solo puede replicar un grafo completo a otros servidores.

Neo4j HA está disponible y es tolerante a particiones (AP).

- Cada esclavo regresará solo lo que tiene actualmente, lo que puede estar fuera de sincronía con el nodo maestro temporalmente.

La utilidad de Neo4j puede ser desagradable si no está acostumbrado a modelar datos de grafos.

Proporciona una poderosa API de código abierto con años de uso en producción y, sin embargo, no ha obtenido la misma tracción que otras bases de datos.

- Atribuimos esto a la falta de conocimiento porque las bases de datos de grafos se entrelazan de manera tan natural con la forma en que los humanos tienden a conceptualizar los datos.

Para ciertas clases de problemas, como las redes sociales, Neo4j es una opción obvia.