

Proyecto Fin de Ciclo



Base de datos orientada a grafos

David García Gómez

2º CFGS Administración de Sistemas Informáticos en Red

IES Gonzalo Nazareno

2017/18

Resumen

Este Proyecto de Fin de Ciclo posee como objetivo principal el estudio y aprendizaje de la base de datos basada en grafos llamada Neo4j.

Para comprender mejor esta base de datos y su eficacia comenzaremos con una introducción donde se defina la misma, así como una explicación de su funcionamiento y las ventajas que presenta y nos puede proporcionar.

Continuando este estudio, aprenderemos cómo instalarla en nuestra máquina servidor ya sea Debian GNU/Linux o Windows.

Posteriormente, mostraremos el modo de acceso a Neo4j así como un análisis de los diferentes protocolos de acceso.

Acto seguido, mostraremos y explicaremos varios de los comandos básicos para crear, modificar, borrar, etc. datos en Neo4j.

Seguidamente, aprenderemos cómo configurar un clúster de alta disponibilidad para nuestra base de datos.

Este proyecto finalizará con una conclusión sobre los diferentes apartados que se presentan.

Palabras claves

Neo4j, Base de Datos, Grafos, Nodos, Relaciones, Clúster.

Índice

Introducción	5
¿Qué es Neo4j?	5
¿Cómo funciona Neo4j?	5
Grafos no dirigidos	5
Grafos dirigidos	5
Grafos con peso	6
Grafos con etiquetas	6
Grafos de propiedad	6
¿Cuáles son sus ventajas?	6
Rendimiento	6
Agilidad	7
Flexibilidad y escalabilidad	7
Instalación de Neo4j en Debian	7
Requisitos previos	7
Agregar el repositorio de Neo4j	8
Instalación Neo4j	9
Instalación de Neo4j en Windows	10
Configuración de los Protocolos de Neo4j	13
Configurar la conexión remota	13
Acceso a Neo4j	13
Comandos básicos de Neo4j	14
CREATE. Crear nodos y relaciones en neo4j	15
Crear nodos	16
Crear un nodo simple	16
Crear un nodo con un Label	16
Crear un nodo con datos y con una etiqueta	17
Crear relaciones	17
Crear relación simple	18
MATCH. Selección de nodos	20
RETURN. Devolver los nodos de una orden	21
WHERE. Filtrar datos en una selección	22
DELETE. Borrar nodos y relaciones	23
SET. Modificar las propiedades de los nodos y relaciones.	25
REMOVE. Eliminar las propiedades de los nodos y relaciones.	27

Configuración de un clúster de alta disponibilidad	29
Características	29
Configuración de las instancias	29
Instancia 1 en el servidor llamado neo1	29
Instancia 2 en el servidor llamado neo2	30
Instancia 3 en el servidor llamado neo3	31
Conclusiones	32
Bibliografía	33

Introducción

¿Qué es Neo4j?

Neo4j es una base de datos orientada a grafos escrita en Java, es decir la información se almacena de forma relacionada formando un grafo dirigido entre los nodos y las relaciones entre ellos. Se integra perfectamente con múltiples lenguajes como Java, PHP, Ruby, .Net, Python, Node, Scala, etc. Está especialmente indicada para modelar redes sociales y sistemas de recomendación.

¿Cómo funciona Neo4j?

Neo4j usa grafos para representar datos y las relaciones entre ellos. Un grafo se define como cualquier representación gráfica formada por vértices (se ilustran mediante círculos) y aristas (se muestran mediante líneas de intersección). Dentro de estas representaciones gráficas, tenemos varios tipos de grafos:

Grafos no dirigidos

Los nodos y las relaciones son intercambiables, su relación se puede interpretar en cualquier sentido. Las relaciones de amistad en la red social Facebook, por ejemplo, son de este tipo.

Grafos dirigidos

Los nodos y las relaciones no son bidireccionales por defecto. Las relaciones en Twitter son de este tipo. Un usuario puede seguir a determinados perfiles en esta red social sin que ellos le sigan a él.

Grafos con peso

En este tipo de gráficas las relaciones entre nodos tienen algún tipo de valoración numérica. Eso permite luego hacer operaciones.

Grafos con etiquetas

Estos grafos llevan incorporadas etiquetas que pueden definir los distintos vértices y también las relaciones entre ellos. En Facebook podríamos tener nodos definidos por términos como ‘amigo’ o ‘compañero de trabajo’ y la relaciones como ‘amigo de’ o ‘socio de’.

Grafos de propiedad

Es un grafo con peso, con etiquetas y donde podemos asignar propiedades tanto a nodos como relaciones (por ejemplo, cuestiones como nombre, edad, país de residencia, nacimiento). Es el más complejo.

¿Cuáles son sus ventajas?

Rendimiento

Las bases de datos orientadas a grafos como Neo4j tienen mejor rendimiento que las relacionales (SQL) y las no relacionales (NoSQL). La clave es que, aunque las consultas de datos aumenten exponencialmente, el rendimiento de Neo4j no desciende, frente a lo que sí sucede con las BD relacionales como MySQL.

Las BDOG responden a las consultas actualizando el nodo y la relaciones de esa búsqueda y no todo el grafo completo. Eso optimiza mucho el proceso.

Volker Pacher, desarrollador de eBay y cliente de Neo4j, explica con datos lo que supuso el cambio de MySQL a esta BDOG en el rendimiento de Shuti, la plataforma que coordina la entregas entre tiendas, mensajerías y compradores en eBay Now: “Nuestra solución Neo4j es literalmente mil veces más rápida que la solución anterior MySQL, con búsquedas que requieren entre 10 y 100 veces menos código”.

Agilidad

Neo4J tiene muchas ventajas, pero una es su agilidad en la gestión de datos. Si nosotros quisiéramos llevar al límite sus capacidades, tendríamos que superar un volumen total de 34.000 millones de nodos (datos), 34.000 millones de relaciones entre esos datos, 68.000 millones de propiedades y 32.000 tipos de relaciones.

Flexibilidad y escalabilidad

Cuando los desarrolladores de una empresa trabajan con grandes datos, buscan flexibilidad y escalabilidad. Las bases de datos orientadas a grafos aportan mucho en este sentido porque cuando aumentan las necesidades, las posibilidades de añadir más nodos y relaciones a un grafo ya existente son enormes.

Instalación de Neo4j en Debian

Para instalar Neo4j en Debian, debemos asegurarnos de que cumplimos los siguientes requisitos:

- Java 8 está instalado.
- Tenemos definido el repositorio que contiene el paquete Debian Neo4j.

Requisitos previos

Para poder instalar Neo4j en nuestro sistema Debian primero deberemos instalar. A continuación veremos como instalar Java 8.

Java 8 en Debian

Para instalar Java 8 en nuestro equipo deberemos introducir los siguientes comandos en nuestra terminal:

Incluimos los siguientes repositorios en nuestro dispositivo.

```
echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu xenial main" | tee  
/etc/apt/sources.list.d/webupd8team-java.list  
  
echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu xenial main" | tee -a  
/etc/apt/sources.list.d/webupd8team-java.list
```

Descargamos la clave pública para autenticar el paquete Java.

```
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys EEA14886
```

Actualizamos los repositorios.

```
apt-get update
```

Instalamos Java 8 en nuestra máquina.

```
apt-get install oracle-java8-installer
```

Agregar el repositorio de Neo4j

El paquete Debian está disponible en <http://debian.neo4j.org> . Para usar el repositorio, realizaremos los siguientes pasos:

```
wget -O - https://debian.neo4j.org/neotechnology.gpg.key  
  
echo "deb http://debian.neo4j.org/repo stable/" | tee -a /etc/apt/sources.list.d/neo4j.list  
  
apt-get update
```


Instalación Neo4j

Ahora instalaremos Neo4j para ello solo tendremos que utilizar el siguiente comando:

```
sudo apt-get install neo4j
```

Si queremos instalar alguna versión específica de Neo4j solo deberemos incluirlo al final del comando, como se muestra a continuación.

Para instalar Neo4j Community Edition:

```
sudo apt-get install neo4j=3.3.0
```

Para instalar Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise=3.3.0
```

Al instalar Neo4j Enterprise Edition, pedirá que aceptemos el acuerdo de licencia. Una vez que se acepta el acuerdo de licencia, comienza la instalación.

Por último, deberemos iniciar el servicio de Neo4j para poder trabajar con nuestra base de datos.

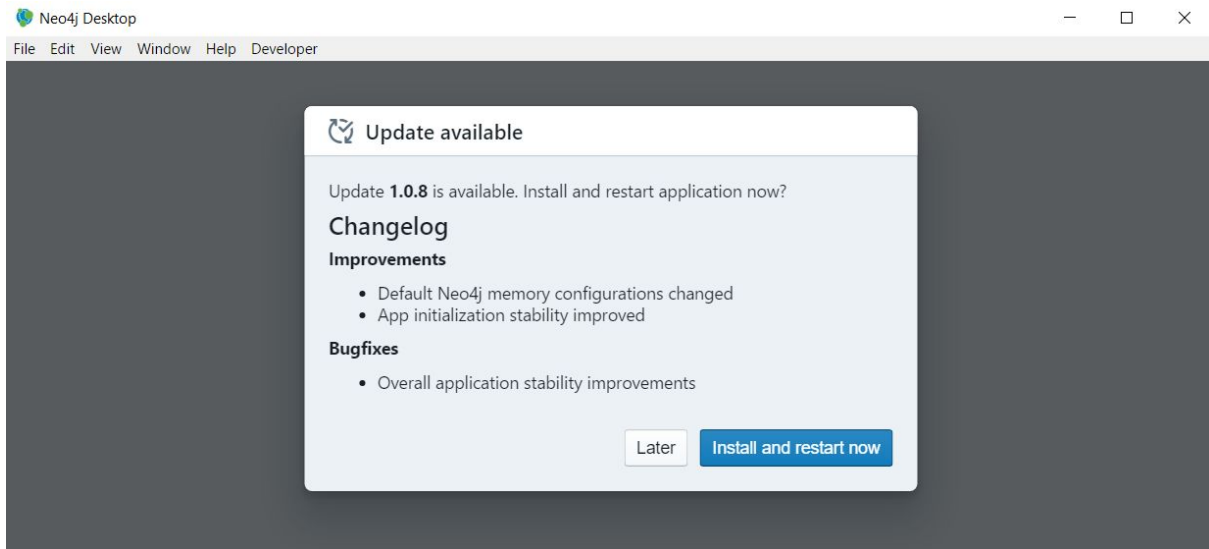
```
service neo4j start
```

Instalación de Neo4j en Windows

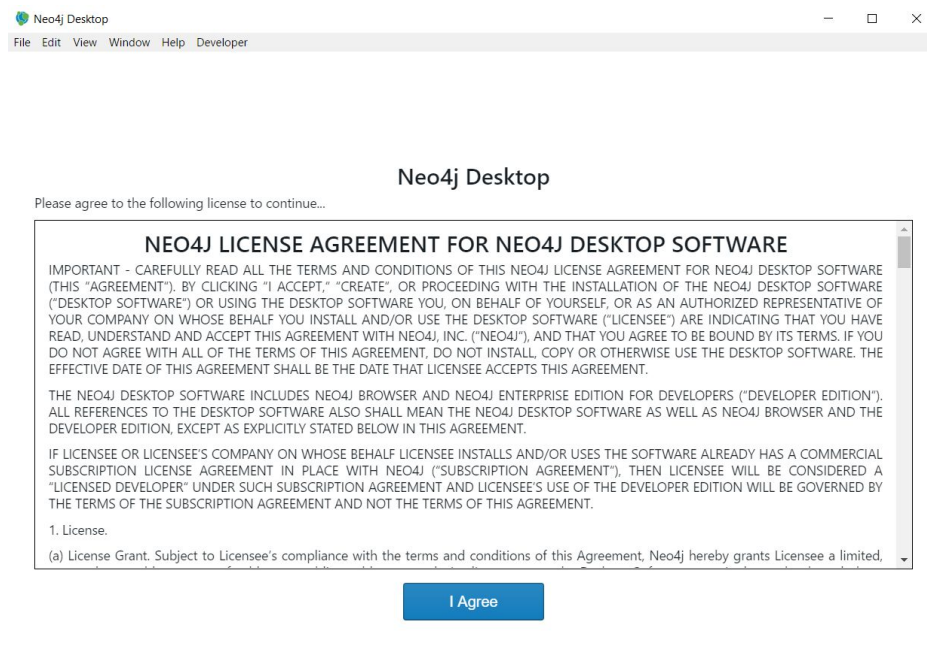
El primer paso para tener Neo4j en nuestra instancia de Windows es descargarlo desde su página oficial.

<https://neo4j.com/download/?ref=hro>

Una vez descargado el fichero de instalación lo ejecutamos.

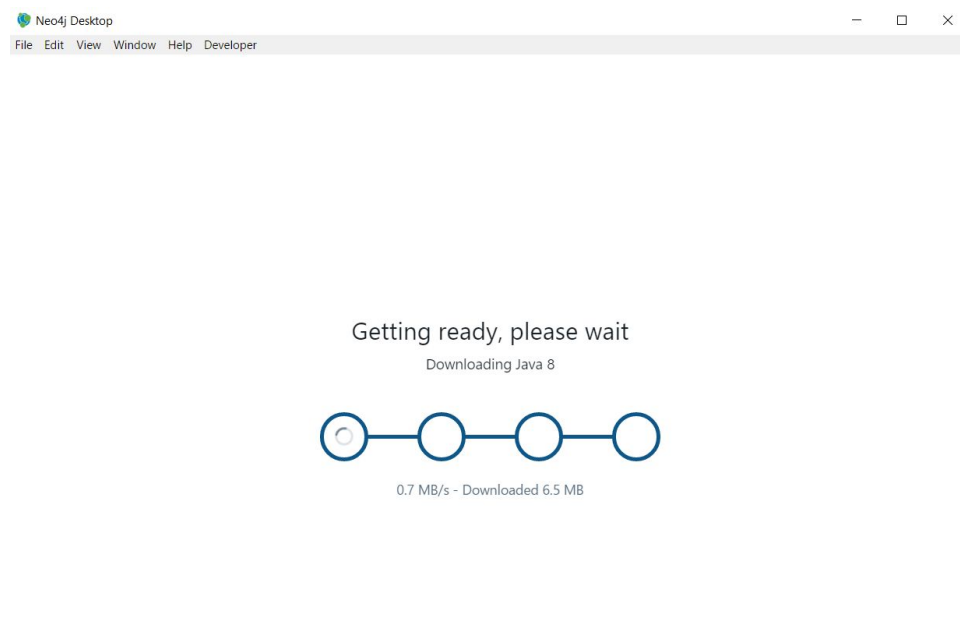


Antes de iniciar la instalación nos pedirá que aceptemos el acuerdo de licencia.

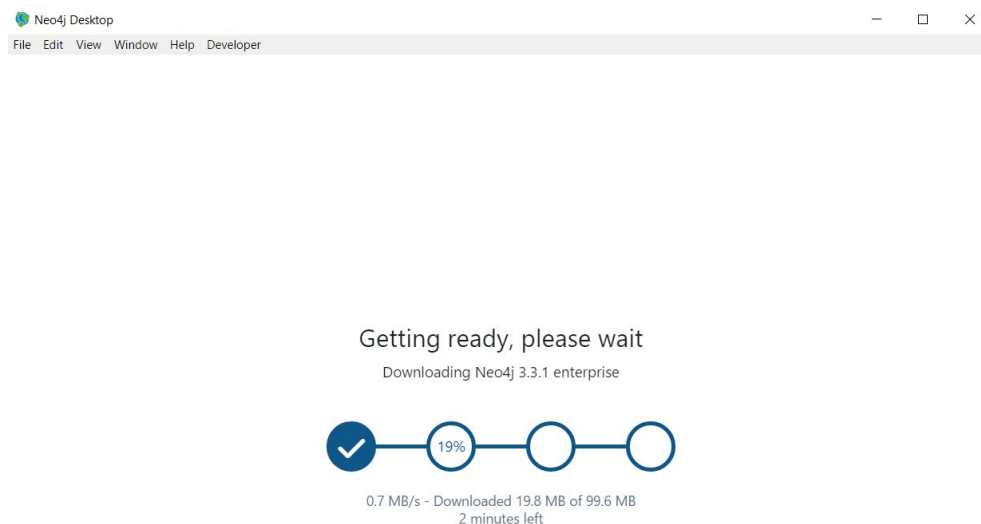


A continuación, debemos loguearnos con una cuenta de correos, de google u otras opciones de registro.

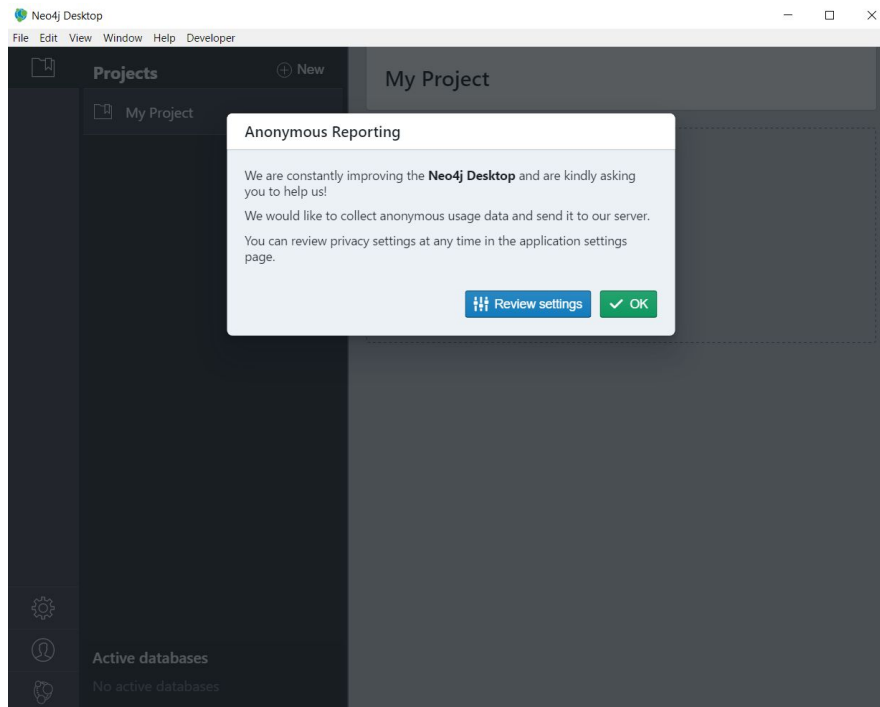
Posteriormente comenzará la instalación. Primero descarga Java 8 ya que es necesario para el uso de Neo4j.



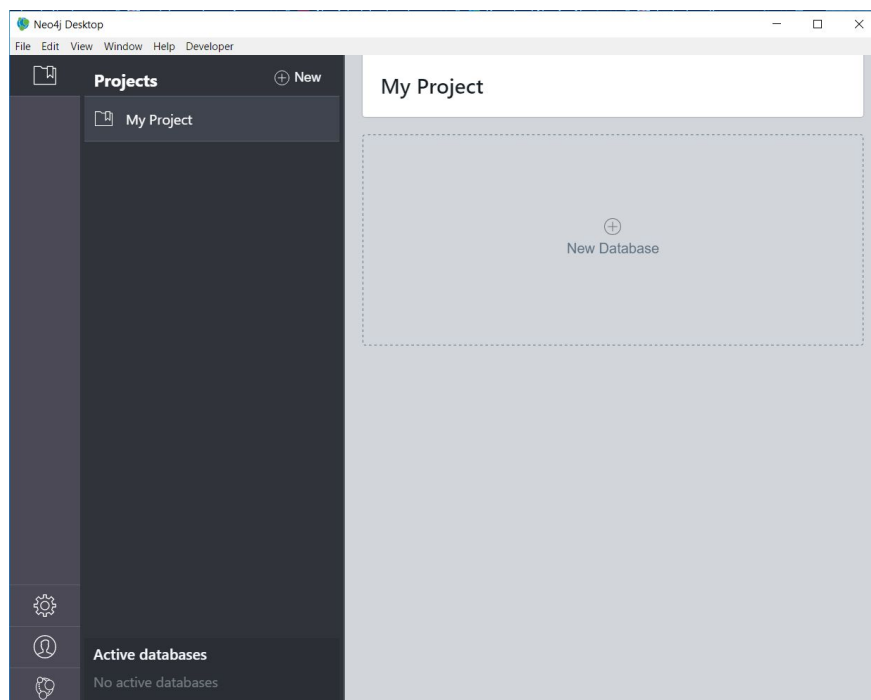
Seguidamente, instalará Neo4j en nuestro sistema.



Una vez finalizada la instalación, nos avisará de que Neo4j recauda información de nuestro uso de la aplicación con motivo de mejorar la misma.



Y listo, ya lo tenemos instalado.



Configuración de los Protocolos de Neo4j

Neo4j admite clientes que usan el protocolo binario Bolt, HTTP y HTTPS. Los tres protocolos están configurados de manera predeterminada, aunque se pueden modificar según las necesidades que se requieran.

Por defecto, los protocolos están configurados en unos puertos específicos que podemos cambiar modificando el fichero neo4j.conf.

Nombre del conector	Protocolo	Puerto
dbms.connector.bolt	Bolt	7687
dbms.connector.http	HTTP	7474
dbms.connector.https	HTTPS	7473

Configurar la conexión remota

Por defecto, la configuración de Neo4j solo acepta conexiones locales. Para aceptar conexiones externas, descomentamos la siguiente línea:

```
dbms.connectors.default_listen_address=0.0.0.0
```

Una vez que modifiquemos esta línea ya podremos acceder a Neo4j desde el buscador.

Acceso a Neo4j

El acceso a Neo4j se realiza a través del navegador. Si la instancia desde la cual accedemos es la misma máquina en la que está hospedada la base de datos, escribiremos en la barra de búsqueda la siguiente dirección:

<http://localhost:7474>

Si por el contrario accedemos a través de una máquina externa, cambiaremos localhost por el número IP de nuestro servidor.

<http://192.168.1.150:7474>

Al acceder por primera vez, Neo4j nos pedirá la contraseña para el usuario que viene por defecto (neo4j). la contraseña para este usuario es neo4j.

Una vez que accedemos nos pedirá que asignemos una nueva contraseña para este usuario.

Cuando terminemos este paso ya estaremos dentro de Neo4j.

Comandos básicos de Neo4j

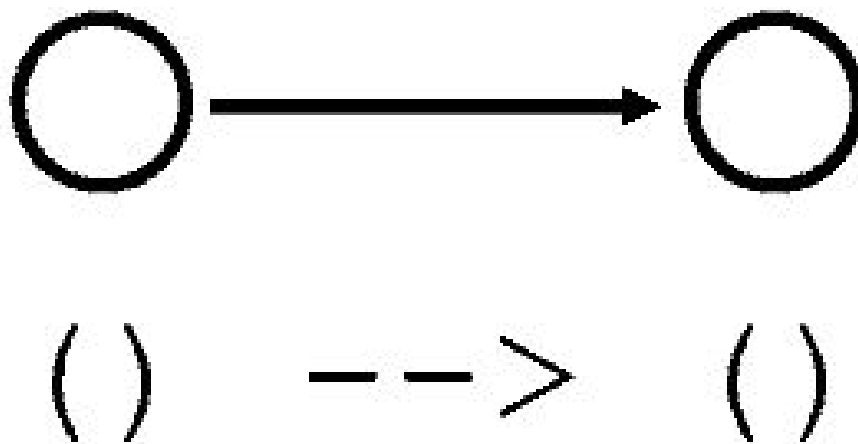
En la siguiente tabla se presentan una serie de comandos básicos para la iniciación del uso de Neo4j.

CREATE	Para crear nodos, relaciones y propiedades
MATCH	Para recuperar los datos acerca de los nodos, relaciones y propiedades
RETURN	Para volver resultados de la consulta
WHERE	Para proporcionar las condiciones para filtrar los datos de recuperación
DELETE	Para eliminar nodos y relaciones
SET	Modificar las propiedades de los nodos y las relaciones
REMOVE	Para eliminar las propiedades de los nodos y las relaciones

CREATE. Crear nodos y relaciones en neo4j

Para la creación de los nodos y relaciones en Neo4j necesitamos introducir las órdenes en un lenguaje propio de esta base de datos llamado Cypher.

Para representar las relaciones se utilizan flechas y para representar nodos se utilizan paréntesis, consiguiendo de esta forma, que las órdenes en Cypher sean muy gráficas.



En la imagen de arriba se ven dos nodos y la relación que los une, debajo se ve su representación genérica en Cypher. como se puede ver es bastante intuitivo y muy visual.

En Neo4j se pueden etiquetar los nodos y las relaciones, viene a ser como definir tipos de relaciones y tipos de nodos, es de gran utilidad su uso en las órdenes para crear consultas más específicas.

Recordad que en Neo4j el tiempo de respuesta de una orden depende de los nodos y relaciones a recorrer, no del tamaño total de la base de datos, por lo tanto cuanto más específica es una orden menos hay que recorrer y tendremos un mejor tiempo de respuesta.

Crear nodos

La cláusula CREATE se utiliza para crear nodos. Dichos nodos pueden crearse de varias maneras y con diferentes atributos.

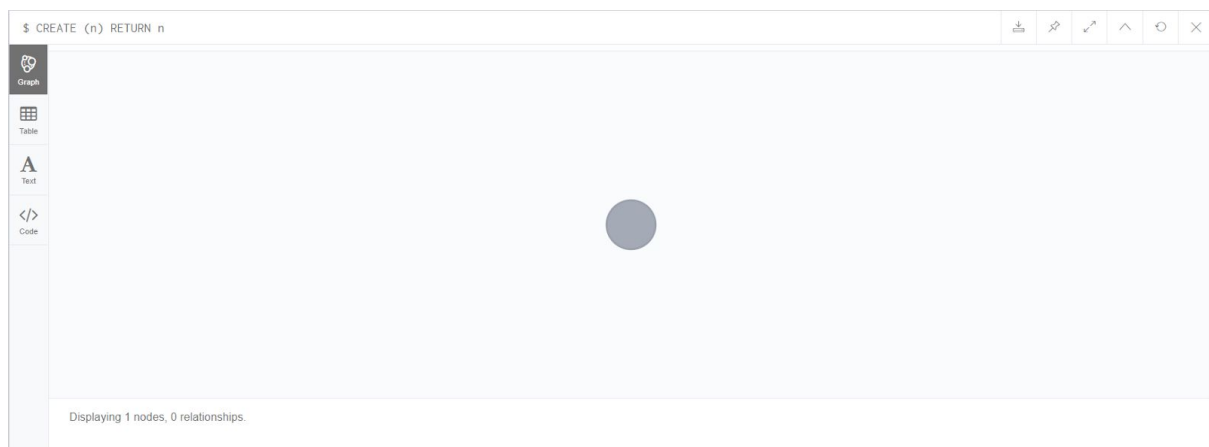
Crear un nodo simple

Si queremos crear un nodo simple solo deberemos escribir create y el nombre de la variable donde se guardará.

```
CREATE (n)
```

Crea un nodo sin datos y lo almacena en la variable n. En la orden posterior se puede utilizar esa variable y devolverla como resultado.

```
CREATE (n) RETURN n
```



Crear un nodo con un Label

Los nodos se pueden clasificar en etiquetas (Labels), es una forma de clasificar los nodos.

```
CREATE (dosh:Ciudad)
```

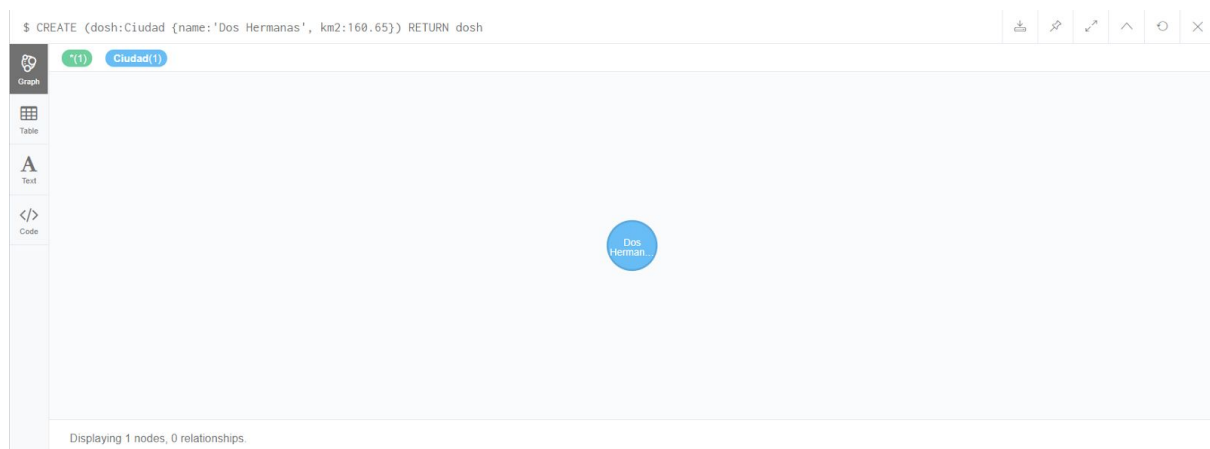

De esta forma creamos un nodo sin datos y le asignamos la etiqueta Product, si no existe la etiqueta la crea.

También podemos asignar más de una etiqueta.

```
CREATE (dosh:Ciudad:Territorio)
```

Crear un nodo con datos y con una etiqueta

```
CREATE (dosh:Ciudad {name:'Dos Hermanas', km2:160.52}) RETURN dosh
```



Los datos que asignamos al nodo son en formato Json.

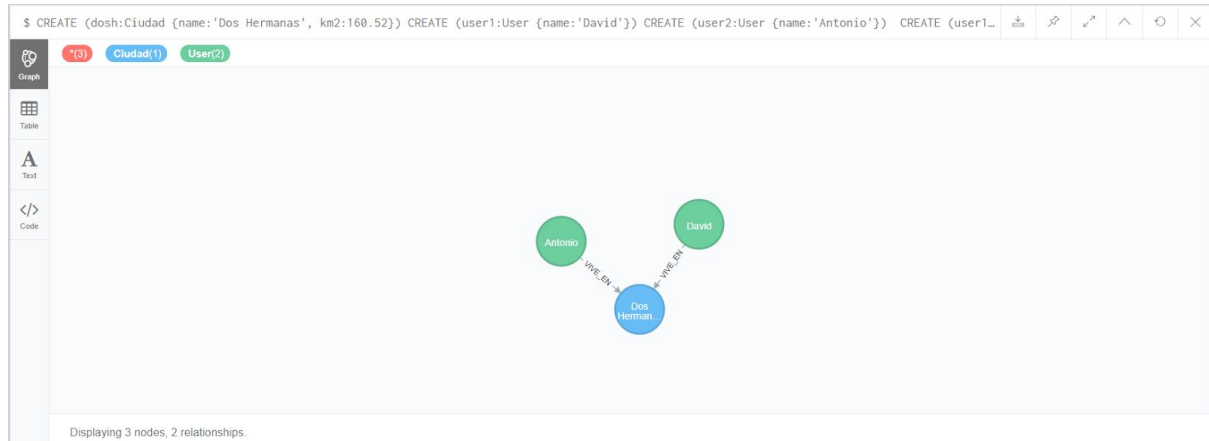
Crear relaciones

La cláusula Create también se utiliza para crear relaciones. Para crear relaciones podemos crear nodos y justo después utilizar las variables de los nodos para crear las relaciones. También podemos realizar una búsqueda de los nodos, almacenarlo en variables para después crear la relación. Como el objetivo de este post es la cláusula create voy a optar por crear los ejemplos con la primera opción.

Crear relación simple

Creamos los nodos y después usamos las variables para relacionarlos.

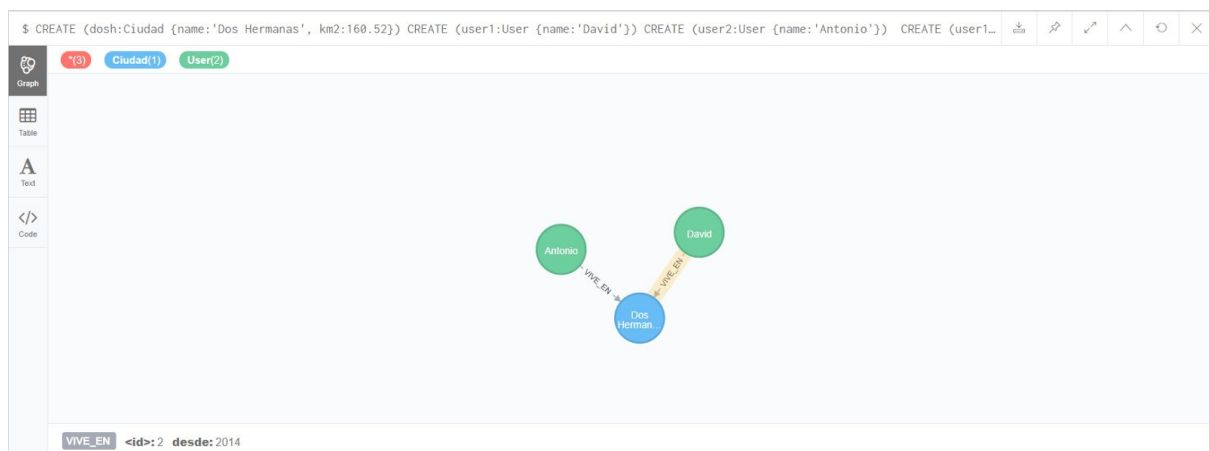
```
CREATE (dosh:Ciudad {name:'Dos Hermanas', km2:160.52})  
  
CREATE (user1:User {name:'David'})  
  
CREATE (user2:User {name:'Antonio'})  
  
  
CREATE  
  
  (user1)-[:VIVE_EN]->(dosh),  
  
  (user2)-[:VIVE_EN]->(dosh)  
  
  
RETURN dosh,user1,user2
```



Es una relación con la etiqueta VIVE_EN y tiene una dirección.

Las relaciones también pueden tener propiedades

```
CREATE (dosh:Ciudad {name:'Dos Hermanas', km2:160.52})  
  
CREATE (user1:User {name:'David'})  
  
CREATE (user2:User {name:'Antonio'})  
  
  
CREATE  
  
(user1)-[:VIVE_EN { desde:2014}]->(dosh),  
  
(user2)-[:VIVE_EN { date:2008}]->(dosh)  
  
RETURN dosh,user1,user2
```



Y las relaciones también pueden almacenarse en variables y devolverse con Return

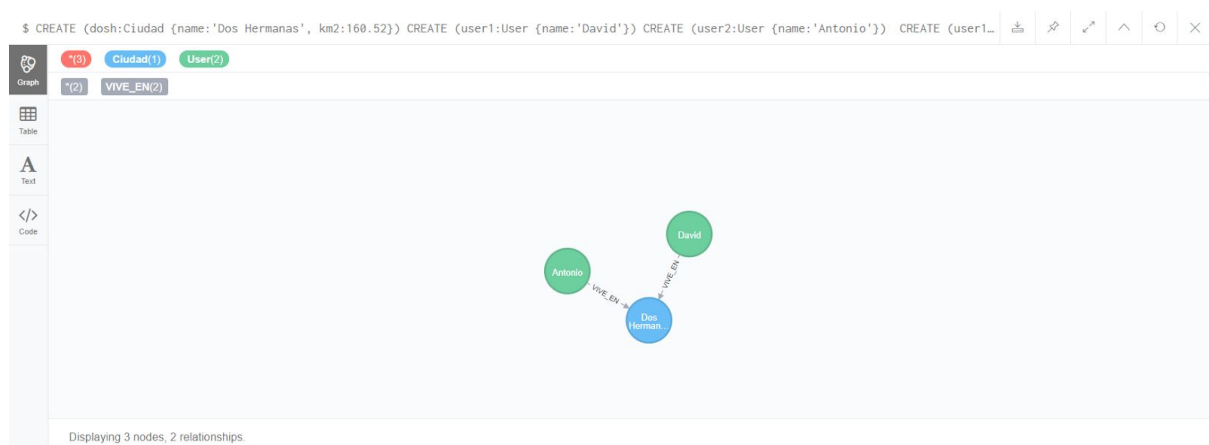
```
CREATE (dosh:Ciudad {name:'Dos Hermanas', km2:160.52})  
  
CREATE (user1:User {name:'David'})  
  
CREATE (user2:User {name:'Antonio'})
```

```
CREATE
```

```
(user1)-[r1:VIVE_EN { desde:2014}]->(dosh),
```

```
(user2)-[r2:VIVE_EN { date:2008}]->(dosh)
```

```
RETURN dosh,user1,user2,r1,r2
```



MATCH. Selección de nodos

Una vez creado el grafo vamos a realizar algunas consultas sobre él. Si queremos buscar algo en el grafo debemos buscar con MATCH. Este comando es el equivalente al comando SELECT en bases de datos SQL Para consultar todo el grafo.

Para ponerlo a prueba lo explicaremos con un ejemplo sencillo.

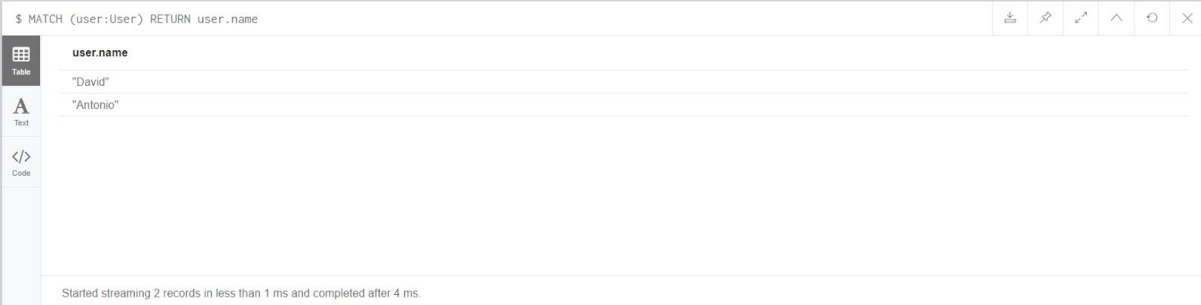
Comenzamos creando un par de nodos.

```
CREATE (user1:User {name:'David'})
```

```
CREATE (user2:User {name:'Antonio'})
```

A continuación, usaremos el comando MATCH para seleccionar a los nodos que contengan la etiqueta User. Finalmente, con la orden RETURN devolveremos los datos correspondiente a esos nodos, en este caso en nombre del usuario (user.name)

```
MATCH (user:User)
RETURN user.name
```



The screenshot shows the Neo4j Cypher query interface. The query entered is `$ MATCH (user:User) RETURN user.name`. The results are displayed in a table with the header `user.name`. The table contains two rows: `"David"` and `"Antonio"`. The interface also shows a status message at the bottom: "Started streaming 2 records in less than 1 ms and completed after 4 ms."

user.name
"David"
"Antonio"

RETURN. Devolver los nodos de una orden

En ejemplos anteriores, hemos usado el comando RETURN para que Neo4j nos retorne los datos de nodos creados o seleccionados previamente. El comando RETURN necesita ir acompañado del comando CREATE o MATCH. Si no es así, el sistema nos devolverá un mensaje de error.

```
RETURN user.name
```



The screenshot shows the Neo4j Cypher query interface with an error. The query entered is `$ RETURN user.name`. The interface displays an error message: "ERROR Neo.ClientError.Statement.SyntaxError Variable 'user' not defined (line 1, column 8 (offset: 7)) 'RETURN user.name'". The error message is highlighted in a grey box. At the bottom, a red warning icon and text state: "Neo ClientError.Statement.SyntaxError: Variable 'user' not defined (line 1, column 8 (offset: 7)) 'RETURN user.name'".

Error
ERROR Neo.ClientError.Statement.SyntaxError Variable 'user' not defined (line 1, column 8 (offset: 7)) "RETURN user.name" ^

WHERE. Filtrar datos en una selección

El comando WHERE, al igual que la orden RETURN, necesita ir acompañado de otros comandos, ya que su función es la de especificar uno o varios atributos sobre los que queramos filtrar.

Para ver su utilidad, elaboraremos un ejemplo sencillo donde veremos reflejado el uso de este comando.

Comenzaremos creando una serie de usuarios con diferentes atributos.

```
CREATE (user1:User {name:'David', role:'Jefe', sal:2000})  
  
CREATE (user2:User {name:'Antonio', role:'Portero', sal:1000})  
  
CREATE (user3:User {name:'Maria', role:'Encargada', sal:1800})  
  
CREATE (user4:User {name:'Alejandro', role:'Vendedor', sal:1500})  
  
CREATE (user5:User {name:'Ana', role:'Vendedor', sal:1500})  
  
CREATE (user6:User {name:'Jose', role:'Secretario', sal:1000})
```

Una vez tenemos estos nodos ya creados, ejecutamos lo siguiente:

```
MATCH (user:User)  
  
WHERE user.name='David'  
  
RETURN user.name, user.role, user.sal
```

Con esta orden pedimos a Neo4j que seleccione todos los usuarios (MATCH) los cuales cumplan la condición de tener en su atributo “name” el valor “David” (WHERE). Por último, el servidor nos devolverá (RETURN) los valores especificados: “name”, “role” y “sal”.

```
$ MATCH(user:User) WHERE user.name='David' RETURN user.name, user.role, user.sal
```

user.name	user.role	user.sal
"David"	"Jefe"	2000

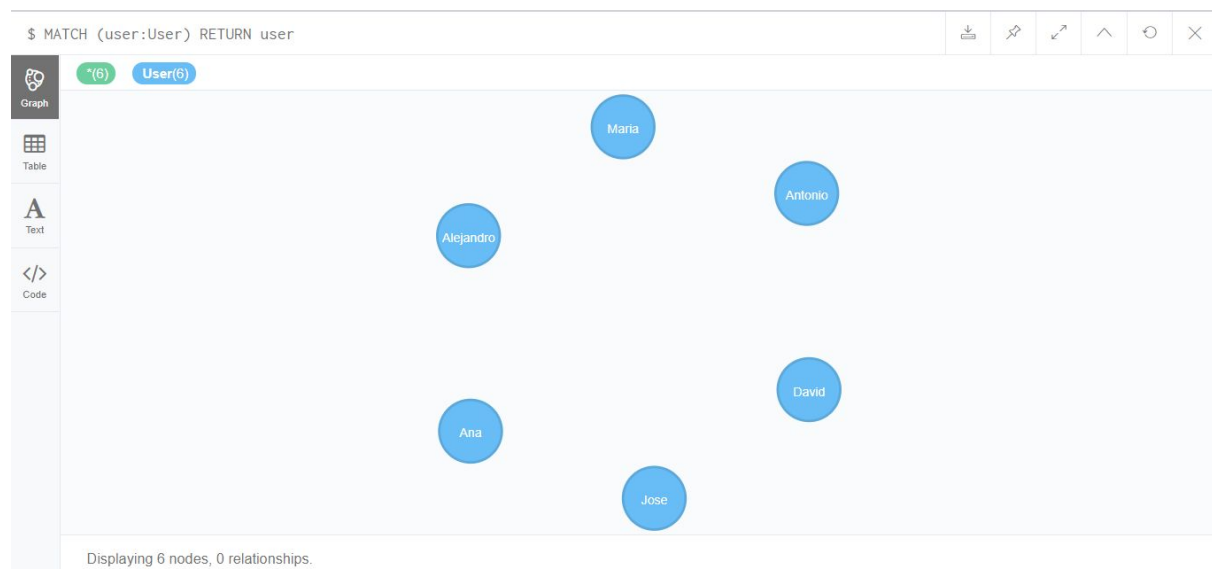
Started streaming 1 records after 4 ms and completed after 4 ms.

DELETE. Borrar nodos y relaciones

El comando DELETE se utiliza para borrar nodos y relaciones que ya no son necesarias. Su uso es muy simple. Debe ir acompañada del comando MATCH y, casi siempre, de WHERE. Para poder apreciar su uso, eliminaremos uno de los nodos anteriores.

Primero mostramos todos los nodos con la etiqueta “User”.

```
MATCH (user:User)  
RETURN user
```



Eliminamos el nodo del usuario que tenga el nombre de “Jose”.

```
MATCH (user:User)
WHERE user.name='Jose'
DELETE user
```

The screenshot shows the Neo4j Cypher console interface. At the top, the query is entered: `$ MATCH (user:User) WHERE user.name='Jose' DELETE user`. Below the query, the status bar indicates: "Deleted 1 node, completed after 20 ms." The left sidebar shows the "Table" view selected, and the "Code" view is also visible.

Y volvemos a mostrar los usuarios restantes.

```
MATCH (user:User)
RETURN user
```

The screenshot shows the Neo4j Cypher console interface. At the top, the query is entered: `$ MATCH (user:User) RETURN user`. Below the query, the status bar indicates: "Displaying 5 nodes, 0 relationships." The left sidebar shows the "Graph" view selected. The main area displays a graph with five blue circular nodes labeled "Alejandro", "Maria", "Antonio", "Ana", and "David". The "Code" view is also visible in the sidebar.

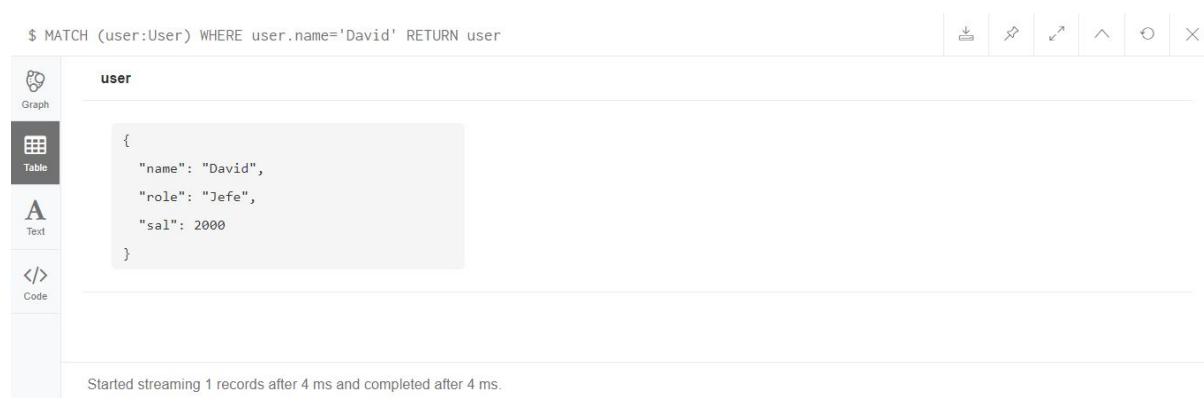
SET. Modificar las propiedades de los nodos y relaciones.

El comando SET se utiliza para modificar los atributos de los nodos y relaciones que tenemos creados. Por ejemplo, si queremos cambiar el valor del salario (sal) de uno de los nodos anteriores utilizaremos dicho comando.

En este caso, le subiremos el salario al usuario “David”.

Primero mostramos las características de este usuario.

```
MATCH (user:User)
WHERE user.name='David'
RETURN user
```



The screenshot shows the Neo4j Cypher console interface. At the top, the query is entered: `$ MATCH (user:User) WHERE user.name='David' RETURN user`. Below the query, the results are displayed in a table view. The table has a single column labeled 'user' and one row containing a JSON object: `{ "name": "David", "role": "Jefe", "sal": 2000 }`. On the left side of the interface, there is a sidebar with icons for Graph, Table (selected), Text, and Code. At the bottom of the console, a status message reads: 'Started streaming 1 records after 4 ms and completed after 4 ms.'

A continuación, modificamos su salario.

```
MATCH (user:User)
WHERE user.name='David'
SET user.sal=2500
```

```
$ MATCH (user:User) WHERE user.name='David' SET user.sal=2500
```

Set 1 property, completed after 24 ms.

Set 1 property, completed after 24 ms.

Por último, mostramos de nuevo las características de este usuario.

```
MATCH (user:User)
WHERE user.name='David'
RETURN user
```

```
$ MATCH (user:User) WHERE user.name='David' RETURN user
```

user

```
{
  "name": "David",
  "role": "Jefe",
  "sal": 2500
}
```

Started streaming 1 records after 4 ms and completed after 4 ms.

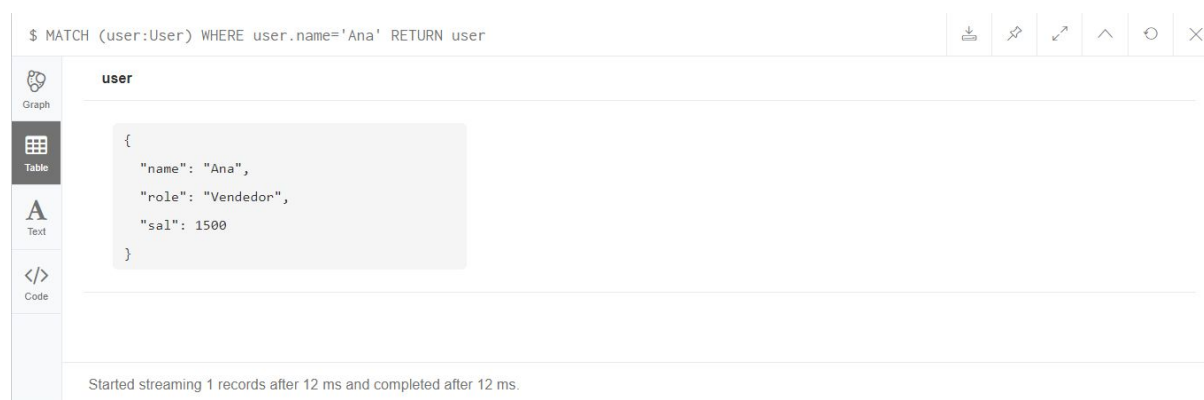
REMOVE. Eliminar las propiedades de los nodos y relaciones.

La función de este comando básico es la de eliminar una propiedad innecesaria de cualquier nodo o relación. La sintaxis de esta orden es similar a la de SET.

Comprobemos su uso con ejemplo.

Eliminaremos el atributo “role” del usuario “Ana”. Para ello, primero mostramos las características de este usuario.

```
MATCH (user:User)
WHERE user.name='Ana'
RETURN user
```



The screenshot shows the Neo4j Cypher query interface. The query entered is: `$ MATCH (user:User) WHERE user.name='Ana' RETURN user`. The interface has a sidebar with icons for Graph, Table, Text, and Code. The 'Table' view is selected, displaying a single record for the user 'Ana' with the following properties: `{ "name": "Ana", "role": "Vendedor", "sal": 1500 }`. At the bottom, a status message reads: "Started streaming 1 records after 12 ms and completed after 12 ms."

Posteriormente, eliminamos la propiedad “role” de este nodo.

```
MATCH (user:User)
WHERE user.name='Ana'
REMOVE user.role
```

\$ MATCH (user:User) WHERE user.name='Ana' REMOVE user.role

Set 1 property, completed after 12 ms.

Table

</>
Code

Set 1 property, completed after 12 ms.

Finalmente, volvemos a mostrar las características del usuario “Ana”.

```
MATCH (user:User)
WHERE user.name='Ana'
RETURN user
```

\$ MATCH (user:User) WHERE user.name='Ana' RETURN user

user

```
{
  "name": "Ana",
  "sal": 1500
}
```

Started streaming 1 records in less than 1 ms and completed in less than 1 ms.

Configuración de un clúster de alta disponibilidad

En esta sección, implementaremos localmente un cluster de alta disponibilidad a partir de tres máquinas virtuales.

Características

Las tres instancias tendrán las siguientes características:

Datos \ Nombre	Instancia 1 (neo1)	Instancia 2 (neo2)	Instancia 3 (neo3)
Sistema Operativo	Debian Jessie 8 (sin entorno gráfico)	Debian Jessie 8 (sin entorno gráfico)	Debian Jessie 8 (sin entorno gráfico)
IP	192.168.1.151	192.168.1.152	192.168.1.153

Una vez tengamos creadas la tres instancias con dichas características deberemos instalar Neo4j como ya explicamos en uno de los apartados anteriores.

A partir de ahora configuraremos cada máquina individualmente.

Configuración de las instancias

Para configurar un clúster de alta disponibilidad en Neo4j deberemos modificar el fichero neo4j.conf.

Instancia 1 en el servidor llamado neo1

Accedemos al fichero de configuración de Neo4j.

```
nano /etc/neo4j/neo4j.conf
```

Modificamos las siguientes líneas.

```
# Unique server id for this Neo4j instance  
# can not be negative id and must be unique
```

```
ha.server_id=1

# List of other known instances in this cluster
# ha.initial_hosts=neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
ha.initial_hosts=192.168.1.151:5001,192.168.1.152:5001,192.168.1.153:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
dbms.connector.http.enabled=true
dbms.connector.http.listen_address=:7474
```

Instancia 2 en el servidor llamado neo2

Accedemos al fichero de configuración de Neo4j.

```
nano /etc/neo4j/neo4j.conf
```

Modificamos las siguientes líneas.

```
# Unique server id for this Neo4j instance
# can not be negative id and must be unique
ha.server_id=2

# List of other known instances in this cluster
# ha.initial_hosts=neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001
# Alternatively, use IP addresses:
ha.initial_hosts=192.168.1.151:5001,192.168.1.152:5001,192.168.1.153:5001

# HA - High Availability
# SINGLE - Single mode, default.
dbms.mode=HA

# HTTP Connector
```

```
dbms.connector.http.enabled=true  
dbms.connector.http.listen_address=:7474
```

Instancia 3 en el servidor llamado neo3

Accedemos al fichero de configuración de Neo4j.

```
nano /etc/neo4j/neo4j.conf
```

Modificamos las siguientes líneas.

```
# Unique server id for this Neo4j instance  
# can not be negative id and must be unique  
ha.server_id=3  
  
# List of other known instances in this cluster  
# ha.initial_hosts=neo4j-01.local:5001,neo4j-02.local:5001,neo4j-03.local:5001  
# Alternatively, use IP addresses:  
ha.initial_hosts=192.168.1.151:5001,192.168.1.152:5001,192.168.1.153:5001  
  
# HA - High Availability  
# SINGLE - Single mode, default.  
dbms.mode=HA  
  
# HTTP Connector  
dbms.connector.http.enabled=true  
dbms.connector.http.listen_address=:7474
```

Conclusiones

Neo4j es una base de datos diferente a todas las que he visto antes. En mi opinión, es una base de datos muy completa.

Uno de sus mejores atributos es la facilidad que ofrece en su instalación. Podemos instalarla en varios tipos de sistemas operativos. En nuestro caso, implantar Neo4j en Debian GNU/Linux y Windows es muy sencillo ya que se realiza en pocos pasos.

En cuanto a su uso, podemos definirlo como intuitivo, ya que con su elemental sintaxis y sus similitudes con otras bases de datos es fácil de comprender para usuarios principiantes.

Además, con su interfaz gráfica podemos visualizar nuestros proyectos y hacernos una clara idea de cómo están relacionados nuestros nodos.

En definitiva, mi experiencia con Neo4j me ha dejado un buen sabor de boca y ha despertado mi interés en conocerla mejor y saber más de ella.

Bibliografía

<https://www.adictosaltrabajo.com/tutoriales/neo4j-first-steps/>

<http://xurxodeveloper.blogspot.com.es/2014/04/cypher-como-crear-datos.html>

http://www.w3ii.com/es/neo4j/neo4j_cql_introduction.html

<https://bbvaopen4u.com/es/actualidad/neo4j-que-es-y-para-que-sirve-una-base-de-datos-orientada-grafos>

<https://neo4j.com/docs/operations-manual/current/installation/linux/debian/>

<https://neo4j.com/blog/neo4j-video-tutorials/>

<https://es.slideshare.net/zeley/tutorial-neo4j-en-espaol>

<https://www.tutorialspoint.com/neo4j/>

<https://www.quackit.com/neo4j/tutorial/>

<https://www.airpair.com/neo4j/posts/getting-started-with-neo4j-and-cypher>

<https://www.lynda.com/Neo4j-tutorials/Up-Running-Neo4j/155604-2.html>