GUÍA DE CYPHER

Andrea Fernández

UNIVERSIDAD DE LEON

ÍNDICE

1. Operaciones en nodos

En NEO4J, los nodos hacen el papel de 'entrada' respecto a SQL. Son objetos que tienen atributos o propiedades y entre los cuales se pueden crear relaciones, similar en como ocurría en las tablas de SQL cuando queríamos indicar que un trabajador pertenecía a una empresa. Además, los nodos pueden ser de un "tipo", no me refiero a 'int' o 'double', sino más bien a algo personalizado como "Usuario", ahora se aclarará en los ejemplos. Aclarado esto, comencemos con las operaciones:

- Creación de nodos.

  ```
  CREATE (nombreVariable:Tipo {propiedad1: "Nombre",
  propiedad2: 2})
  ```

  Ejemplo:

  ```
  CREATE (user:Usuario {nombre: "Fer98", contraseña:
  "1234"})
  ```

  Cabe mencionar que las variables sólo se usan dentro de una misma query, no se recuerdan entre consulta y consulta.

- Eliminar nodos.

  ```
  MATCH (nombreVariable {propiedad: 'nombre' })
  DETACH DELETE nombreVariable
  ```

  Ejemplo:

  ```
  MATCH (n {name: 'Andy' })
  DETACH DELETE n
  ```

  El detach sirve para que además se elimine la relación que tiene, si es que la posee. También podemos eliminar un nodo por su id interna:

  ```
  MATCH (nombreVariable:Tipo) WHERE ID(p)=1
  DELETE nombreVariable
  ```

- Cambiar propiedades.

  ```
  MATCH (a:Tipo {propiedad1: "Nombre"})
  SET a.nombre = "Nombre"
  ```

  Ejemplo:

```
MATCH (u:Usuario {nombre: “Fer98”})
SET u.nombre = “Fer97”
```

2. Búsqueda y devolución de datos

Aunque ya lo hemos usado en el ejercicio anterior, el componente esencial para buscar nodos es “MATCH”. Suele ir acompañado de un “WHERE”, cuya función es la misma que en SQL, veámoslo:

- Búsqueda de nodos.

  Sin WHERE:

  ```
  MATCH (variable:Tipo)
  ```

  Sin WHERE y con condición de propiedad:

  ```
  MATCH (variable:Tipo {propiedad1: “Nombre”})
  ```

  Con WHERE:

  ```
  MATCH (v:Tipo)
  WHERE v.propiedad1 = "Nombre"
  RETURN v
  ```

  Ejemplo:

  ```
  MATCH (u:User)
  WHERE u.nombre = “Fer97”
  RETURN u
  ```

  Para todos los nodos:

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

- Búsqueda de relaciones

  Podemos buscar tanto un extremo como otro de la relación si especificamos uno de ellos. Por ejemplo, aquí especificamos el elemento de la izquierda y pedimos retornar el de la derecha, al que llamamos n:

  ```
  MATCH (:Tipo {propiedad: ‘Nombre’}) -[nombreRelación]- (n)
  RETURN n
  ```

  En este otro ejemplo, simplemente pedimos encontrar los nodos a los que llega el de la derecha, sea la relación que sea.

  ```
  MATCH (:Tipo {propiedad: ‘Nombre’}) -[*2]- (n)
  RETURN n
  ```

Encuentra todos los nodos alcanzables según el número

- Conteo

  La siguiente query contará los nodos relacionados con el nodo ‘Nodo1’:

```
MATCH (n {name: 'Nodo1'})-->(x)
RETURN count(x)
```

  Esta muestra además el nombre de cada relación:

```
MATCH (n { name: 'Nodo1' })-[r]->()
RETURN type(r), count(r)
```

3. Operaciones con relaciones

Al igual que en SQL establecíamos relaciones entre tablas con FOREIGN KEYS, en NEO4J se pueden crear relaciones entre nodos.

- Crear relación
  Las relaciones también pueden tener propiedades:

  ```
  CREATE (a)-[r:NOMBRE_RELACION { propiedadRelacion:
  "Nombre"}]->(b)
  ```

  Hay que tener en cuenta que a y b son variables, es decir, han tenido que ser declaradas anteriormente (con un MATCH, por ejemplo). La dirección de la flecha puede ser inversa o incluso inexistente:

  ```
  CREATE (a)<-[r:NOMBRE_RELACION { propiedadRelacion:
  "Nombre"}]-(b)
  ```

  Ejemplos:

  ```
  MATCH (u:User {nombre:"Fer97"})
  MATCH (a:Admin {nombre:"Andrea98"})
  CREATE (a)-[:ADMINS]->(u)

  CREATE (b:Birdie {especie: "Guacamayo"})
  WITH b
  MATCH (d:`Dueño`)
  WHERE d.nombre = 'Andrea'
  CREATE (d)-[:POSEE_A]->(b)
  ```

  La directiva "WITH" se usa entre CREATE y MATCH cuando se quiera usar una variable desde CREATE a MATCH

- Eliminar relación

  ```
  MATCH (variable:Tipo {propiedad: 'nombre'})-
  [nombreVariable:nombreRelacion]->()
  ```

  Ejemplo:

  ```
  MATCH (n:Person { name: 'Andy' })-[r:KNOWS]->()
  DELETE r
  ```

4. Otras operaciones

Aquí recogeré otras operaciones que considero bastante útiles pero no pude clasificar en los apartados anteriores:

- Eliminar la base de datos entera

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

- Ordenar

```
MATCH (v:Videogame)
RETURN (v)
ORDER BY v.rank
```

En esta query, estamos obteniendo todos los nodos de tipo Videogame para después ordenarlos por su Rank. Cabe mencionar que pueden usarse dos variables en un ORDER BY, primero ordenará los elementos según un criterio y luego otro

- Importar un archivo .CSV

A la hora de crear una base de datos, podemos hacerla desde cero o bien tomar un archivo .CSV para sacar nuestros datos de ahí. Aquí unos ejemplos reales usados en mi aplicación:

```
LOAD CSV WITH HEADERS FROM
'file:///metacritic_games.csv' AS line
WITH line LIMIT 300
CREATE (v:Videogame { name: line.game, year:
toInteger(right(line.release_date, 4)), platform:
line.platform, rating: line.rating, metascore:
line.metascore, user_score: line.user_score})

FOREACH (n IN (CASE WHEN line.developer IS NULL THEN
[] ELSE [1] END) |
MERGE (d:Developer {name: line.developer})
CREATE (d)-[:DEVELOPED]->(v)
)
FOREACH (n IN (CASE WHEN line.genre IS NULL THEN []
ELSE [1] END) |
MERGE (g:Genre {name: line.genre})
CREATE (v)-[:OF_GENRE]->(g)
)
```

Al principio, hay que indicar con 'WITH HEADERS' si la primera línea del .CSV corresponde a todos los headers, o sea, los futuros atributos de nuestros nodos ("nombre"," correo" …). Puede verse que establecí un límite de 300 lecturas. También tendremos que determinar el nombre con el que identificaremos a cada línea del archivo, en este caso, 'line'. De esta manera, podemos acceder fácilmente a cada atributo, como se ve en la creación de "Videogame". Usé el FOREACH para que no se tuviera en cuenta los valores null, y el MERGE para que, si muchos nodos tenían el mismo nombre, se juntaran en uno (por ejemplo, muchos juegos tenían a Nintendo como nodo relacionado, de no ser por MERGE, cada juego estaría relacionado con un nodo Nintendo en vez de todos a uno).