NEO4J

https://bit.ly/2v5EaEv

Cypher
AS

Al igual que SQL, Neo4j también tiene una forma para cambiar la forma en la que se hace referencia a un resultado, esto es particularmente útil para cuando se usan funciones, pero se puede usar en cualquier momento:

```
neo4j$ MATCH (p:Person)
RETURN p , p.name AS name , {name: p.name, label:labels(p)}
AS Person
```

Al igual que SQL, Neo4j también tiene una forma para cambiar la forma en la que se hace referencia a un resultado, esto es particularmente útil para cuando se usan funciones, pero se puede usar en cualquier momento:

```
neo4j$ MATCH (p:Person)
RETURN p , p.name AS name , {name: p.name, label:labels(p)}
AS Person
```

Si están usando la consola web, este resultado se verá mejor en modo texto en lugar del modo grafo que tiene por defecto.

Como pueden ver del ejemplo, al RETURN se le pueden dar varios tipos de valores, prácticamente lo que sea, como un diccionario. Además con el AS podemos cambiar la forma en la que la salida hace referencia a los valores.

Cypher tiene varias funciones, como "labels()", que usamos para obtener las etiquetas del nodo en la consulta.

Cypher
DISTINCT

Cuando consultamos por propiedades de nodos (como las etiquetas en el ejemplo anterior) obtenemos un resultado por cada nodo del MATCH:

```
neo4j$ MATCH (n)
       RETURN labels(n) AS Etiquetas
```

Pero si solo queremos saber qué etiquetas existen (por ejemplo) entonces no necesitamos la respuesta de cada nodo, si no las distintas que hay, en ese caso podemos usar DISTINCT, que se usa después del RETURN:

```
neo4j$ MATCH (n)
       RETURN DISTINCT labels(n) AS Etiquetas
```

Las agregaciones también se usan en la fase del RETURN. Se tienen las agregaciones de siempre: count(), sum(), avg(), min(), max(), etc.

```
neo4j$ MATCH (n)
       RETURN count(*) AS Personas
```

Cypher

Antes de pasar a la siguiente lámina, intenten hacer una consulta que les de todas las parejas ACTOR-DIRECTOR que hayan trabajado juntos en una película.

Cypher

Probablemente llegaron a una respuesta como esta:

```
neo4j$ MATCH (actor:Person) -[:ACTED_IN]-> (m:Movie)
       MATCH (director:Person) -[:DIRECTED]-> (m)
       RETURN actor, director, m
```

Es correcto, pero Cypher ofrece también la otra dirección de relación, por lo que es posible hacer esto:

```
neo4j$ MATCH (actor:Person) -[:ACTED_IN]->
(m:Movie) <-[DIRECTED]- (director:Person)
       RETURN actor, director, m
```

Y ¿cómo podemos obtener cuántas colaboraciones a tenido cada pareja ACTOR-DIRECTOR?

Y ¿cómo podemos obtener cuántas colaboraciones a tenido cada pareja ACTOR-DIRECTOR?

```
neo4j$ MATCH (actor:Person) -[:ACTED_IN]->
(m:Movie) <-[DIRECTED]- (director:Person)
      RETURN actor, director, count(*)
```

Esto es gracias a que Cypher usa una agregación implícita, eso quiere decir que se usarán todos los elementos que se mencionen (en el RETURN) y se agruparan sobre los que no.

Cypher
ORDER BY

Si queremos saber qué tan prolífico es un actor. Podemos buscar todas las veces que actuó un actor en una película, y luego agrupar con respecto a las películas (implícitamente). Además, podemos ordenarlos con ORDER BY, ya sea de forma ascendente ASC o descendente DESC.

```
neo4j$ MATCH (a:Person) -[:ACTED_IN]-> (m:Movie)
       RETURN a, count(*) AS participaciones
       ORDER BY participaciones DESC LIMIT 10
```

Cypher
collect

Otro caso ¿qué pasa si queremos encontrar todos los actores que participan en una película? necesitamos hacer una agrupación de actores con respecto a las películas y meterlos en una lista, para eso se usa la función "collect()":

```
neo4j$ MATCH (a:Person) -[:ACTED_IN]-> (m:Movie)
RETURN m.title, collect(a.name) AS reparto, count(*) AS actores
```

Cypher
WITH

Hay ocasiones en que queremos darle un nombre a una salida parcial para poder encadenarla con otras instrucciones, para eso se usa WITH:

```
neo4j$ MATCH (person:Person) -[:ACTED_IN]-> (m:Movie)
WITH person, count(*) AS actuaciones, collect(m.title) AS
películas
WHERE actuaciones > 1
RETURN person.name,actuaciones, películas
```

OJO, solo las "columnas" que se regresan con WITH estarán disponibles para futuras operaciones, es por eso que se necesita regresar a "person".

Cypher
WITH

Hay ocasiones en que queremos darle un nombre a una salida parcial para poder encadenarla con otras instrucciones, para eso se usa WITH:

```
neo4j$ MATCH (person:Person) -[:ACTED_IN]-> (m:Movie)
WITH person, count(*) AS actuaciones, collect(m.title) AS películas
WHERE actuaciones > 1
RETURN person.name,actuaciones, películas
```

OJO, solo las "columnas" que se regresan con WITH estarán disponibles para futuras operaciones, es por eso que se necesita regresar a "person".

Antes de correr esa instrucción, intenten entender qué es lo que le están pidiendo a la base de datos.

El principal modo en que una base de datos de grafo encuentra información es al recorrer relaciones a partir de uno o varios nodos iniciales.
Los índices son para encontrar esos nodos iniciales. Una vez que un índice se crea se usa automáticamente, no es necesario indicarlo explícitamente:

```
neo4j$ CREATE INDEX ON :Person (name)
```

```
neo4j$ MATCH (actor:Person {name: "Tom Hanks"})
       RETURN actor
```

El principal modo en que una base de datos de grafo encuentra información es al recorrer relaciones a partir de uno o varios nodos iniciales.
Los índices son para encontrar esos nodos iniciales. Una vez que un índice se crea se usa automáticamente, no es necesario indicarlo explícitamente:

```
neo4j$ CREATE INDEX ON :Person (name)

neo4j$ MATCH (actor:Person {name: "Tom Hanks"})
       RETURN actor
```

El índice que se creó es, como seguramente adivinaron, para hacer búsquedas rápidas de los nombres sobre aquellos nodos que tienen la etiqueta de "Person".
Se puede decir que implícitamente se tienen índices con respecto a las etiquetas de los nodos, pero para hacer búsquedas de propiedades particulares el sistema tiene que buscar sobre todos los nodos que comparten dicha propiedad.

Para evitar toda esa búsqueda se usan los índices.

Neo4j también soporta índices compuestos:

```
neo4j$ CREATE INDEX ON :Person (name,born)
```

Los índices solo aplican para los nodos que tengan tanto las etiquetas como las propiedades. Es decir, si hay actores que tienen nombre pero no fecha de nacimiento, no serán incluidos en este último índice.

Neo4j puede mantener restricciones sobre sus datos, al igual que lo haría una base de datos relacional. Por ejemplo, si se quisiera mantener una restricción de que los títulos de las películas fueran únicos se podría hacer lo siguiente:

```
neo4j$ CREATE CONSTRAINT ON (m:Movie)
       ASSERT m.title IS UNIQUE
```

Al igual que en las bases de datos relacionales, cuando se agrega una restricción, se agrega un índice de forma implícita.

Actividades

- Ejecutar el comando:

  neo4j$:play northwind-graph

  Y seguir las instrucciones del tutorial.

Actividades

Descargar los datos:
http://www.datos.economia.gob.mx/NormatividadMercantil/PROFECO_2015_vEspecial.csv
Crear una base de datos de tipo grafo con los datos diseñada para una aplicación que permita:

- Dado un estado y un producto, buscar lugares donde pueda encontrarlo.
- Dado un estado y una tienda, verificar si tiene algun incumplimiento con un producto.
- Dado un estado y un producto, buscar alternativas sin incumplimiento de ese producto o categoría.

Actividades

Descargar los datos:
http://www.datos.economia.gob.mx/NormatividadMercantil/PROFECO_2015_vEspecial.csv
Crear una base de datos de tipo grafo con los datos diseñada para una aplicación que permita:

- Dado un estado y un producto, buscar lugares donde pueda encontrarlo.
- Dado un estado y una tienda, verificar si tiene algun incumplimiento con un producto.
- Dado un estado y un producto, buscar alternativas sin incumplimiento de ese producto o categoría.

En la base de datos, las tiendas vienen dadas en la columna de "razón social visitada".

Actividades

Además, su aplicación deben ser capaz de:

- Llevar un registro de las compras y lugares que ha hecho cada usuario.
- Recomendar a un usuario una tienda en done pueda encontrar en un solo lugar, lo que compra en diferentes tiendas.
- Encontrar los estados con mayor y menor incumplimiento relativo al número de tiendas que tiene.
- Recomendar productos a un usuario según lo que otros usuarios han comprado. Es decir, Dado un usuario y un producto, obtener todos los productos de esa misma categoría que han comprado otros usuarios que también han comprado ese mismo producto.