NEO4J

Neo4j es una base de datos de grafos, lo que quiere decir que toda su información es representada y expresada como nodos y relaciones.
Este tipo de base de datos son famosas por su compatibilidad de representación con visualizaciones amigables como las que se podrían presentar en un pizarrón.

Instalación

Pre-requisitos:

```
$ sudo apt-get install openjdk-11-jdk
```

Para obtener los binarios se puede correr el comando:

```
$ wget https://neo4j.com/artifact.php?name=neo4j-community-4.2.6-unix.tar.gz
```

Y para descomprimir:

```
$ tar -xzf artifact.php?name=neo4j-community-4.2.6-unix.tar.gz
```

Instalación

Desde la nueva carpeta en *bin* se puede correr cassandra, pero seguramente prefieren agregarlo al *path*

```
$ echo 'export PATH=$PATH:~/neo4j-community-4.0.1' >>
~/.bashrc
```

Una vez que Neo4j esté instalado se puede iniciar con:

```
$ neo4j console
```

Lo siguiente es entrar a la página de administración:

https://localhost:7474

Con los datos:

Usuario: neo4j
Contraseña: neo4j

En cypher toda expresión se hace a partir de patrones que buscan asemejarse a nodos y relaciones de un grafo:

```
(:Persona) -[:VIVE_EN]-> (:Ciudad)
(:Ciudad) -[:PARTE_DE]-> (:País)

(:Persona) -[:VIVE_EN]-> (:Ciudad) -[:PARTE_DE]-> (:País)
```

Comencemos a ver las formas en las que se puede representar un nodo:

```
()
(matrix)
(:Película)
(matrix:Película)
(matrix:Película {título: "The Matrix"})
(matrix:Película {título: "The Matrix",fecha:1997})
```

Por su parte, las relaciones también tienen una sintaxis propia:

```
-->
-[rol]->
-[:ACTUO_EN]->
-[rol:ACTUO_EN]->
-[rol:ACTUO_EN {papel: "Neo"}]->
```

Es posible combinar estas expresiones, con lo que podemos regresar a un patrón completo:

```
(keanu:Actor {nombre:"Keanu Reeves"}) -[rol:ACTUO_EN
{papel:"Neo"}]-> (matrix:Película {título:"The Matrix"})
```

Ahora que ya entendimos el tipo de datos que maneja neo4j, es hora de usarlos:

```
neo4j$ CREATE (:Película {título:"The Matrix", fecha:1997})
```

Eso solo nos indica que se creó el nodo, pero también se puede hacer que regrese el nodo que se acaba de crear:

```
neo4j$ CREATE (p:Persona {nombre:"Keanu Reeves",
nacimiento:1964})
    RETURN p
```

Cypher
CREATE

Dentro de la misma instrucción podemos crear nodos y relaciones, se pueden crear varias relaciones gracias a las referencias mediante sus nombres:

```
neo4j$ CREATE (a:Persona {nombre:"Tom Hanks",
nacimiento:1956})
      -[r:ACTUO_EN {rol:"Forrest"}]->
      (b:Película {título:"Forrest Gump", fecha:1994})
CREATE (d:Persona {nombre:"Robert Zemeckis" ,
nacimiento:1951})
      -[:DIRIGIO]->(b)
RETURN a,b,r,d
```

No solo crear, también se tiene que poder encontrar:

```
neo4j$ MATCH (p:Película)
       RETURN p
```

También se pueden encontrar nodos gracias a sus atributos, no solo sus etiquetas:

```
neo4j$ MATCH (p:Persona {nombre:"Keanu Reeves"})
       RETURN p
```

Con el match y gracias a los nombres también podemos hacer consultas de otras propiedades. Para consultarlas, se usa la notación con punto:

```
neo4j$ MATCH (p:Persona {nombre:"Tom Hanks"})
      -[r:ACTUO_EN]-> (m:Película)
RETURN m.título,r.rol
```

Cypher
MATCH *and* CREATE

Ahora que sabemos como encontrar nodos, podemos agregar relaciones a esos nodos:

```
neo4j$ MATCH (p:Persona {nombre:"Tom Hanks"})
CREATE (m:Película {título:"Cloud Atlas", fecha:2012})
CREATE (p) -[r:ACTUO_EN {rol:"Zachry"}]->(m)
RETURN p,r,m
```

Cypher
MATCH *or* CREATE = MERGE

Si buscamos una coincidencia que no existe, no funcionará el MATCH. Pero si creamos un nodo que ya existe, se va a duplicar la información.
Se necesita una forma de verificar si existe un nodo, y si no, crearlo. Esa es el trabajo de MERGE:

```
neo4j$ MERGE (m:Película {título:"Cloud Atlas"})
       ON CREATE SET m.fecha = 2016
       RETURN m
```

Cypher
MATCH *or* CREATE = MERGE

¿Ya pusimos la relación de Neo en matrix? Puede que si, puede que no, si no se acuerdan al momento de hacer la actualización a la base de datos, es mejor usar MERGE, porque si no, pueden crear una relación duplicada.

```
neo4j$ MATCH (p:Persona {nombre:"Keanu Reeves"})
       MATCH (m:Película {título:"The Matrix"})
       MERGE (p) -[r:ACTUO_EN]-> (m)
       ON CREATE SET r.rol="Neo"
       RETURN p,m,r
```

Cypher

Limpiemos todo para comenzar con nuevos ejemplos mas completos.

```
neo4j$ MATCH (n)
       DETACH DELETE n
```

Podemos cargar la base de datos del TXT para que tengamos todos la misma información y se puedan hacer las pruebas.

Para filtrar información, podemos usar WHERE:

```
neo4j$ MATCH (m:Movie)
       WHERE m.title = "The Matrix"
       RETURN m
```

De hecho, este mismo resultado ya sabíamos como obtenerlo directamente con la consulta:

```
neo4j$ MATCH (m:Movie {title:"The Matrix"})
       RETURN m
```

Cypher
WHERE

Podemos utilizar una serie de herramientas para complementar los filtros WHERE. Podemos usar cosas como:

- Expresiones regulares
- Comparaciones niméricas
- Búsquedas en listas

```
neo4j$ MATCH (p:Person) -[r:ACTED_IN]-> (m:Movie)
WHERE p.name =~ "K.+"
OR m.released > 2000
OR "Neo" IN r.roles
RETURN p,r,m
```

WHERE es capaz de seleccionar elementos particulares dentro de un MATCH general, pero también es capaz de quitar aquellos elementos que no nos interesan:

```
neo4j$ MATCH (p:Person) -[r:ACTED_IN]-> (m:Movie)
WHERE NOT (p) -[:DIRECTED]-> ()
RETURN p,m
```

Con esto logramos buscar a todos los actores que no hayan dirigido nada.

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
```

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".

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
```

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

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.