NEO4J

https://bit.ly/2v5EaEv

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})
```

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})
```

- En primer lugar se tiene un nodo anónimo, sin ninguna característica ni restricción.
- En segundo lugar se tiene un nodo sin restricciones, pero esta vez ya se le ha bautizado con un nombre (se puede ver como una variable). Cuando se quiera hacer referencia a ese nodo (solo en esa consulta) se puede usar ese nombre.
- En tercer lugar podemos ver cómo restringir un nodo para que cumpla con una etiqueta en particular.
- En cuarto lugar se combina la restricción con darle un nombre al nodo para poder hacer referencia a él.
- En quinto y sexto lugar se agrega a la restricción propiedades particulares para el nodo, se usan como una lista de pares llave-valor.

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

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

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

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

Podemos ver un comportamiento muy similar.

- En primer lugar, una relación anónima y sin restricciones.
- En segundo lugar, una relación sin restricción pero ya con un nombre para poder referenciarla.
- En tercer lugar, una relación con nombre y que tiene que coincidir con una etiqueta particular.
- Por último, una relación también puede tener propiedades, se expresan de la misma manera: con una lista de pares llave-valor.

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"})
```

Y para ahorrar trabajo, incluso se pueden usar variables para guardar patrones completos:

```
actor_en = (:Actor) -[:ACTUO_EN]-> (:Película)
```

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 {nombre:"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
```

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
```

Con "=∼" se indica que se va a usar una expresión regular. Si no están familiarizados con expresiones regulares, son comandos con significado especial que buscan textos que cumplen patrones específicos, en el ejemplo, se está buscando cualquier nombre que comience con la letra "K". Les recomiendo que quiten los otros dos filtros y se queden solo con ese para que vean qué ocurre.

Si solo quieren el actor y no las películas en las que actuó, pueden quitar "r" y "m" del RETURN, recuerden que lo que se está buscando es la relación completa, por eso si no quitan esta parte su respuesta tendrá también las películas en las que actuaron los actores.

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
```

Además, a diferencia de los primeros ejemplos que estuvimos manejando, en la base de datos "completa" los roles de los actores en las películas se maneja en listas. Con este ejemplo podemos ver dos cosas:

- Los pares llave-valor, pueden tener también listas completas como su "valor"
- Para buscar dentro de una lista, podemos usar "IN"

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.