Neo4j

Lenguaje Cypher

Versión 2.1

04/2015

Contenido

Introducción

Lenguaje

Cypher es un lenguaje declarativo que permite enfocarse en el dominio y no en el acceso a la base de datos, ya que expresa qué recuperar y no cómo.

Muchas cláusulas están inspiradas en SQL, SPARQL, Haskell y Python.

Las sentencias se componen de varias cláusulas, que se encadenan y alimentan entre sí con resultados intermedios.

Existen cláusulas para lectura y para escritura. Una parte de un query Cypher no puede a la vez seleccionar y actualizar el grafo. Cada parte puede leer y hacer match, o hacer updates en el grafo.

Cada query que actualiza el grafo se ejecuta dentro de una transacción.

Nodos

Unidad fundamental de almacenamiento de información.

Pueden contener propiedades. Pueden ser etiquetados. Las propiedades pueden ser arrays, colecciones de strings, números o booleanos.

Se los puede referenciar mediante identificadores, los cuales diferencian mayúsculas/minúsculas y pueden contener caracteres alfanuméricos y guión bajo, pero deben comenzar con una letra.

La forma más simple de un patrón es un nodo. Se describe con paréntesis y generalmente se le da un nombre:

```
(nodo)
```

La etiqueta de un nodo es el atributo más simple que se puede describir en un patrón.

```
(nodo: etiqueta)

(nodo: etiqueta { propiedades })

()
```

Ej.:

```
(nodo1: Persona { apellido:"Corrado", nombre:"Gustavo" } )
```

Relaciones

Son claves en un grafo. Permiten encontrar información relacionada.

Pueden contener propiedades.

Conectan dos nodos. Son entrantes a, o salientes de un nodo debido a su dirección.

Siempre tienen un tipo (y sólo uno).

Se referencian mediante una flecha entre nodos:

```
(nodo1)-->(nodo2)
```

```
(nodo1)-[relación]->(nodo2)
(nodo1)-[relación:TIPO]->(nodo2)
(nodo1)-[relación:TIPO { propiedades }]->(nodo2)
(nodo1)--(nodo2)
```

Ej.:

```
(Gonzalez)-[:ESTUDIO { estado: "Completo" }]->(CsEs)
```

Propiedades

Las propiedades se expresan en patrones entre llaves que encierran pares clave-valor, separados por comas.

Valores

Los tipos de valores soportados son:

- Numérico
- String
- Boolean
- Nodos
- Relaciones
- Caminos

Comentarios

Se pueden agregar comentarios a los queries mediante //.

Cláusulas de lectura

MATCH

Es la forma principal de obtener datos desde la BD.

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

```
MATCH (n)-->(m)
```

MATCH puede utilizar cualquier patrón.

```
MATCH (n:Etiqueta { propiedades } )-[:TIPO]->(m:Etiqueta)
WHERE condición
```

Los patrones de nodos pueden contener etiquetas y propiedades

```
MATCH p = (n)-->(m)
```

Se puede asignar un camino a una variable.

```
OPTIONAL MATCH (n)-[r]->(m)
```

Patrón opcional. Utiliza NULL para las partes faltantes. Podría considerarse el equivalente al outer join en SQL.

Obtener los nodos de todas las personas de la red.

```
MATCH (a:Persona)
RETURN a
```

Obtener el nodo correspondiente a la persona de apellido Domínguez.

```
MATCH (a:Persona {apellido: "Dominguez"})
RETURN a
```

Obtener el nombre y la fecha de nacimiento de la persona de apellido Domínguez.

```
MATCH (a:Persona {apellido: "Dominguez"})
RETURN a.nombre, a.fechanac
```

Obtener la lista de empresas en las que trabajó Domínguez.

```
MATCH (a:Persona {apellido:"Dominguez"}) -[r:TRABAJO]->(b:Empresa)
RETURN b
```

WHERE

```
WHERE propiedad operador valor
```

Los predicados se utilizan para filtrar. WHERE siempre es parte de otra cláusula, a la cual modifica.

```
n.property <> {value}
```

Con operadores de comparación

Obtener la lista de personas que estudiaron carreras que no son de nivel “Universitario”, y los nombres de las carreras.

```
MATCH (a:Persona) -[:ESTUDIO]->(b:Carrera)
WHERE b.nivel <> "Universitario"
RETURN a, b
```

```
n.number >= 1 AND n.number <= 10
```

Combinados con operadores booleanos.

```
n:Person
```

Verificar etiquetas de nodos.

Obtener los nodos etiquetados como Conocimiento.

```
MATCH (a:Conocimiento)
RETURN a
```

```
identificador IS NULL
```

Verificar si algo es nulo.

```
NOT has(n.propiedad) OR n.propiedad = {valor}
```

No existe la propiedad o se cumple la condición.

```
n.property = {value}
```

Una propiedad inexistente devuelve Nulo (lo cual no es igual a nada)

```
n.property =~ "Tob.*"
```

Expresión regular.

Obtener los nodos de todas las personas con nombre terminado en a.

```
MATCH (a:Persona)
WHERE a.nombre =~ ".*a"
RETURN a
```

```
(n)-[:RELACION]->(m)
```

Patrón con por lo menos un match.

```
n.property IN [{value1}, {value2}]
```

Verificar si existe un elemento en una colección.

Cláusulas de escritura

CREATE (nodo)

```
CREATE (n {name: {value}})
```

Crea un nodo con las propiedades dadas.

Crear un nodo para la persona Analía Martinelli.

```
CREATE (n:Persona {nombre: "Analía", apellido: "Martinelli"})
RETURN n
```

```
[MATCH WHERE]
(CREATE [UNIQUE] | MERGE)
[RETURN [ORDER BY] [SKIP] [LIMIT]]
```

El CREATE UNIQUE combinado con el MATCH, crea un nodo sólo si el patrón que describe el query no existe en ningún nodo.

Si Analía Martinelli no lo posee, crear el conocimiento “Cálculo” asociado a ella.

```
MATCH (n:Persona {nombre: "Analía", apellido: "Martinelli"})
CREATE UNIQUE (n)-[p:POSEE]->(c:Conocimiento {nombre: "Cálculo"})
RETURN n, p, c
```

Volver a ejecutar la sentencia anterior.

Verificar en la lista de Conocimientos si se creó duplicado.

```
MATCH (n:Conocimiento) RETURN n.nombre ORDER BY n.nombre
```

Lo que se crea en la cláusula CREATE no existe en la cláusula MATCH.

CREATE (relación)

```
CREATE (n)-[r:TIPO]->(m)
```

Crea una relación con el tipo y dirección dados, y le enlaza un identificador.

```
CREATE (n)-[:TIPO {propiedad: {valor}}]->(m)
```

Crea una relación con el tipo, dirección y propiedades dados.

Para crear una relación entre dos nodos, primero obtenemos los dos nodos. Una vez que están cargados, creamos una relación entre ellos.

Crear para Analía Martinelli una relación Estudio con la carrera "Lic en Sist de Inf", con estado “En curso”.

```
MATCH (n:Persona { nombre: "Analía", apellido: "Martinelli" }),
(c:Carrera { nombre: "Lic en Sist de Inf"})
CREATE (n)-[r:ESTUDIO { estado: "En curso"}]->(c)
RETURN n, r, c
```

CREATE UNIQUE es algo intermedio entre MATCH y CREATE. Obtendrá la coincidencia que pueda, y creará lo que falta, haciendo el menor cambio posible al grafo.

Analía Martinelli conoce a Verónica Mendez. Crear la relación asegurando que se cree una sola vez.

```
MATCH (a:Persona {apellido: "Martinelli"}),
(b:Persona {apellido: "Mendez"})
CREATE UNIQUE (a)-[r:CONOCE_A]->(b)
RETURN r
```

Reejecutar.

Reejecutar sin UNIQUE.

MERGE (nodo)

```
MERGE ...
ON CREATE SET ...
ON MATCH SET ...
```

Selecciona según el patrón o lo crea si no existe. Usa las cláusulas ON CREATE y ON MATCH para updates condicionales.

Crear un nodo para Analía Martinelli sólo si no existe, con fecha de nacimiento 30/06/1968. Si existe, setear esta fecha.

```
MERGE (n:Persona {nombre: "Analía", apellido: "Martinelli"})
ON CREATE SET n.fechanac="30/06/1968"
ON MATCH SET
n.fechanac="30/06/1968"
RETURN n
```

MERGE (relación)

```
MATCH nodos
MERGE relación
```

Encuentra o crea una relación entre nodos.

Crear si no existe la relación de trabajo entre Analía Martinelli y Accenture.

```
MATCH (a:Persona {apellido: "Martinelli"}), (b:Empresa
{nombre:"Accenture"})
RETURN a, b
```

```
MATCH (a:Persona {apellido: "Martinelli"}), (b:Empresa
{nombre:"Accenture"})
MERGE (a)-[r:TRABAJO]->(b)
RETURN a, r, b
```

Reejecutar la sentencia.

SET

```
SET n.property = {value},
    n.property2 = {value2}
```

Actualiza o crea una propiedad.

```
SET n={map}
```

Setea todas las propiedades. Esto eliminará cualquier propiedad existente.

```
SET n:Etiqueta
```

Agrega una etiqueta a un nodo.

Agregarle a Analía Martinelli la etiqueta “Empleado” y el país Argentina.

```
MATCH (a:Persona {apellido: "Martinelli"})
SET a:Empleado,
a.pais="Argentina"
RETURN a
```

REMOVE

```
REMOVE n:Etiqueta
```

Elimina una etiqueta de n.

```
REMOVE n.property
```

Elimina una propiedad.

Eliminar de Analía Martinelli la fecha de nacimiento y la etiqueta Persona.

```
MATCH (a:Persona {apellido: "Martinelli"})
REMOVE a:Persona,
a.fechanac
RETURN a
```

DELETE

`DELETE n, r`

Borra nodos y relaciones.

Eliminar del grafo a Analía Martinelli y todas sus relaciones.

```
MATCH (a { nombre: "Analía", apellido: "Martinelli"})-[r]-()
RETURN r, a

MATCH (a { nombre: "Analía", apellido: "Martinelli"})-[r]-()
DELETE r, a
```

Exploración de datos (para entornos no productivos)

Contar nodos

```
MATCH (n)
RETURN count(*);
```

Contar tipos de relaciones

```
MATCH (n)-[r]->()
RETURN type(r), count(*);
```

Listar todos los nodos y sus relaciones.

```
MATCH (n)-[r]->(m)
RETURN n as desde, r as `->`, m as hasta;
```

Otros elementos de sintaxis

CASE

```
CASE identificador
 WHEN condición1 THEN valor1
[ WHEN condición2 THEN valor2]
[ ELSE condición3]
END
```

Variante: CASE genérico

```
CASE
 WHEN condición1 THEN valor1
[ WHEN condición2 THEN valor2]
[ ELSE condición3]
END
```

Devuelve el valor de THEN correspondiente al valor WHEN que coincide. El valor de ELSE es opcional. Si no está, se sustituye por NULL.

Obtener los nombres y rubros de las empresas registradas, reemplazando el rubro Telefonía por IT.

```
MATCH (e: Empresa)
RETURN e.nombre, CASE e.rubro
WHEN "Telefonía" THEN "IT"
ELSE e.rubro
END
```

ETIQUETAS

```
CREATE (n:Etiqueta {propiedad:{valor}})
```

Crea un nodo con una etiqueta y propiedad.

```
MERGE (n:Etiqueta {propiedad:{valor}})
```

Recupera o crea nodo único con etiqueta y propiedad.

```
SET n:Lbl1:Lbl2:Lbl3
```

Agrega etiquetas a un nodo.

```
MATCH (n:Etiqueta)
```

Recupera nodos etiquetados con esa etiqueta.

```
MATCH (n:Etiqueta)
WHERE n.propiedad = {valor}
```

Recupera nodos cuya propiedad cumplen con ese valor, etiquetados con esa etiqueta.

```
WHERE (n:Person)
```

Chequea la existencia de la etiqueta en el nodo.

```
labels(n)
```

Etiquetas del nodo.

Se desea saber qué etiquetas tienen los nodos que son destino de la relación ESTUDIO.

```
MATCH ()-[:ESTUDIO]->(b)
RETURN labels(b)
```

```
MATCH ()-[:ESTUDIO]->(b)
RETURN DISTINCT labels(b)
```

Observar nodo de inicio de la relación en el query.

`REMOVE n:Etiqueta`

Elimina la etiqueta del nodo.

Operadores

Matematicos

`+, -, *, /, %, ^`

De comparación

`=, <>, <, >, <=, >=`

`Nota: false < true. NULL < valor.`

Booleanos

`AND, OR, XOR, NOT`

De String

`+ (concatenación)`

De Colección

`+, IN, [x], [x .. y]`

Expresiones Regulares

`=~`

NULL

NULL se utiliza para representar valores faltantes o indefinidos.

NULL no es igual a NULL. La expresión NULL = NULL devuelve NULL y no TRUE. Para chequear si una expresión es nula, usar `IS NULL` (o `IS NOT NULL`).

Las expresiones aritméticas, las comparaciones y llamados a funciones devuelven NULL cuando alguno de sus argumentos es NULL.

Los elementos faltantes, como una propiedad que no existe, devuelven NULL.

Colecciones

```
['a','b','c'] AS coll
```

Las colecciones literales se declaran entre corchetes, con los elementos separados por comas.

```
length({coll}) AS len, {coll}[0] AS value
```

Las colecciones se pueden pasar como parámetro.

```
range({first_num},{last_num},{step}) AS coll
```

Range crea una colección de números (el parámetro step es opcional).

Otras funciones que devuelven colecciones son: labels, nodes, relationships, rels, filter, extract.

```
MATCH (a)-[r:KNOWS*]->()
RETURN r AS rels
```

Los identificadores de releaciones de un camino de longitud variable contienen una colección de relaciones.

```
coll[{idx}] AS value,
coll[{start_idx}..{end_idx}] AS slice
```

Se puede acceder a los elementos de una colección con subíndices entre parántesis. Los índices no válidos retornan NULL. Se pueden obtener fragmentos con intervalos desde start_idx hasta end_idx. Se ignoran los elementos fuera de rango.

unwind

Con UNWIND se puede transformar cualquier colección en filas individuales nuevamente.

```
UNWIND[1, 2, 3] AS x
RETURN x
```

Devuelve cada valor de la colección original como una fila individual.

Funciones

Funciones matemáticas

abs()

Valor absoluto.

rand()

Valor aleatorio. Devuelve un nuevo valor para cada llamada.

round()

Redondea al entero más cercano. Ceil y floor encuentran el próximo entero hacia arriba o abajo.

sqrt()

Raíz cuadrada.

sign()

0 si es cero, -1 si es negativo, 1 si es positivo.

sin()

Funciones trigonométricas, también cos, tan, cot, asin, acos, atan, atan2, haversin.

degrees({expr}), radians({expr}), pi()

Convierte radianes en grados o grados en radianes. Obtiene el valor del número pi.

log10({expr}), log({expr}), exp({expr}), e()

Logaritmo en base 10, natural, e elevado al parámetro, valor de e.

Funciones de string

str()

```
str({expression})
```

Representación de caracteres de la expresión.

replace()

```
replace({original}, {search}, {replacement})
```

Reemplaza todas las ocurrencias de search con replacement.

substring()

```
substring({original}, {begin}, {sub_length})
```

Obtiene parte de un string. El último argumento es opcional.

left(), right()

```
left({original}, {sub_length}),
  right({original}, {sub_length})
```

La primera parte de un string. La última parte de un string.

trim, ltrim, rtrim

```
trim({original}), ltrim({original}),
  rtrim({original})
```

Recorta todos los espacios en blanco, o a la izquierda o a la derecha.

upper, lower

```
upper({original}), lower({original})
```

Mayúsculas y minúsculas.

split

```
split({original}, {delimiter})
```

Parte un string en una colección de strings.

Funciones de relaciones

type

```
type(a_relationship)
```

Representación en string del tipo de relación.

startNode

```
startNode(a_relationship)
```

Nodo de inicio de una relación.

endNode

```
endNode(a_relationship)
```

Nodo de fin de una relación.

Id

```
id(a_relationship)
```

Id interno de la relación.

Funciones de grupo

Toman múltiples valores de entrada y calculan un valor agregado a partir de ellos.

Si se retornan dos expresiones, una de ellas sin función de agregación, será la clave de agrupamiento.

count

`count(*)`

El número de nodos coincidentes.

`count(identifier)`

El número de valores no nulos.

`count(DISTINCT identifier)`

Elimina los duplicados de los valores a contar.

Contar la cantidad de nodos del grafo.

```
MATCH (n)
RETURN "Hello Graph with "+count(*)+" Nodes!"
as welcome;
```

Contar la cantidad de personas del grafo.

```
MATCH (n:Persona)
RETURN "Se registraron "+count(*)+" personas en la red"
```

Contar la cantidad de personas que estudiaron una carrera (en cualquier estado).

```
MATCH (n:Persona)-[:ESTUDIO]->(c:Carrera)
RETURN count(distinct n)
```

¿Por qué utilizamos distinct?

Contar la cantidad de personas que estudiaron una carrera completa (estado: “Completo”).

```
MATCH (n:Persona)-[:ESTUDIO {estado: "Completo"}]->(c:Carrera)
RETURN count(distinct n)
```

sum

```
sum(n.property)
```

Sumar valores numéricos. Otras funciones similares son avg, min, max.

Funciones avanzadas de caminos

Procesar caminos

```
(n)-->(m)
```

Existe una relación de n a m.

```
(n:Persona)
```

Recupera nodos con la etiqueta Persona.

```
(n:Persona:Empleado)
```

Recupera nodos con las etiquetas Persona y Empleado.

```
(n:Etiqueta {propiedad: {valor}})
```

Recupera nodos con las propiedades declaradas.

```
(n:Persona)-->(m)
```

Nodo n etiquetado Persona tiene una relación hacia m.

```
(n)--(m)
```

Relación en cualquier dirección entre n y m.

```
(m)<-[:CONOCE_A]-(n)
```

Existe una relación de tipo CONOCE_A de n a m.

```
(n)-[:KNOWS|TRABAJO]->(m)
```

Existe una relación de tipo CONOCE_A o TRABAJO de n a m.

```
(n)-[r]->(m)
```

Enlaza un identificador a la relación.

```
(n)-[*2]->(m)
```

Caminos de longitud 2.

```
(n)-[*1..5]->(m)
```

Caminos de longitud variable entre 1 y 5.

```
(n)-[*3..]->(m)
```

Caminos de longitud 3 ó más.

```
(n)-[*..5]->(m)
```

Caminos de longitud 5 ó menos.

```
(n)-[*]->(m)
```

Caminos de cualquier longitud.

```
(n)-[:KNOWS]->(m {property: {value}})
```

Recuperar o setear propiedades en cláusulas MATCH, CREATE, CREATE UNIQUE o MERGE.

lengh(path)

`length(path)`

Largo del camino.

Se desea saber si mediante la relación CONOCE_A es posible llegar directa o indirectamente (con caminos en cualquier dirección) desde Mario López hasta Jorge Lupis.

```
MATCH camino=(a {nombre:"Mario", apellido:"López"})-[:CONOCE_A*]-
(b {nombre:"Jorge", apellido:"Lupis"})
RETURN camino
```

*Observar uso de * y ausencia de dirección en la relación.*

```
MATCH camino=(a {nombre:"Mario", apellido:"López"})-[:CONOCE_A*]-
(b {nombre:"Jorge", apellido:"Lupis"})
RETURN length(camino)
```

nodes(path)

`nodes(path)`

Los nodos de un camino como colección.

```
MATCH camino=(a {nombre:"Mario", apellido:"López"})-[:CONOCE_A*]-
(b {nombre:"Jorge", apellido:"Lupis"})
RETURN nodes(camino)
```

relationships(path)

```
relationships(path)
```

Las relaciones de un camino como colección.

```
MATCH camino=(a {nombre:"Mario", apellido:"López"})-[:CONOCE_A*]-
(b {nombre:"Jorge", apellido:"Lupis"})
RETURN relationships(camino)
```

extract

```
MATCH path=(n)-->(m)
RETURN extract(x IN nodes(path) | x.prop)
```

Procesa los nodos de un camino. Examina una colección, ejecuta una expresión para cada elemento, y retorna los resultados en una colección con esos valores.

FOREACH

```
MATCH path = (begin) -[*]-> (end)
FOREACH
  (n IN rels(path) | SET n.marked = TRUE)
```

Ejecuta una operación para cada relación en el camino.

shortestpath()

```
shortestPath((n1:Person)-[*..6]-(n2:Person))
```

Encuentra el camino más corto.

```
allShortestPaths((n1:Person)-->(n2:Person))
```

Encuentra todos los caminos más cortos.

Obtener el camino más corto en la relación CONOCE_A entre Gustavo y Mario, para cualquier dirección y longitud.

```
MATCH (a {nombre:"Gustavo"}), (b {nombre:"Mario"}),
camino=shortestPath((a)-[:CONOCE_A*]->(b))
RETURN camino
```

Lista de personas que trabajan o trabajaron en empresas en las que una persona determinada trabajó, pero que no son sus contactos, para sugerirle nuevos contactos.

```
MATCH (a:Persona)-[:TRABAJO]->()<-[:TRABAJO]-(b:Persona)
WHERE id(a) <> id(b)
AND NOT (a)-[:CONOCE_A]-(b)
RETURN DISTINCT a.nombre, a.apellido, b.nombre, b.apellido
```

Lista de conocimientos que poseen más personas que estudiaron en cada carrera.

```
MATCH (a:Persona)-[:POSEE]->(c:Conocimiento), (a)-[:ESTUDIO]-
>(d:Carrera)
RETURN d.nombre, c.nombre, count(distinct a) AS cantidad
ORDER by d.nombre, cantidad DESC
```

Ranking de los primeros 2 conocimientos que poseen más personas egresadas de la carrera "Ing en Sistemas de Información"

```
MATCH (a:Persona)-[:POSEE]->(c:Conocimiento),
 (a)-[e:ESTUDIO]->(d:Carrera {nombre: "Ing en Sistemas de Información"})
WHERE e.estado = "Completo"
WITH d.nombre as carrera, c.nombre as conocim, count(DISTINCT a) AS
cantidad
RETURN carrera, conocim, max(cantidad) as maximo
ORDER BY carrera, maximo DESC LIMIT 2;
```

Ranking de las 3 personas más populares: las que son conocidas por más personas en la red

```
MATCH (a:Persona)<-[:CONOCE_A]-(b:Persona)
WITH a as persona, count(DISTINCT b) AS cantidad
RETURN persona, cantidad
ORDER BY cantidad DESC LIMIT 3;
```

Otras funciones

Funciones para predicados

```
ALL(identificador en colección WHERE predicado)
ANY(identificador en colección WHERE predicado)
NONE(identificador en colección WHERE predicado)
SINGLE(identificador en colección WHERE predicado)
EXISTS(patrón o propiedad)
```

```
coalesce(n.property, {defaultValue})
```

La primera expresión no nula.

```
timestamp()
```

Milisegundos desde la medianoche del 01/01/1970 UTC.

```
id(node_or_relationship)
```

Id interno de una relación o nodo.

```
head(expresión)
```

El primer elemento de una colección

```
last(expresión)
```

El último elemento de una colección

`toInt({expr})`

Convierte la entrada a un entero de ser posible. De lo contrario, retorna NULL.

`toFloat({expr})`

Convierte la entrada a un número de punto flotante si es posible. De lo contrario, retorna NULL.

`toString({expr})`

Convierte el argumento a una cadena de caracteres.

Cláusulas generales

RETURN

```
RETURN *
```

Devuelve el valor de todos los identificadores. Puede devolver nodos, relaciones o propiedades.

```
RETURN n AS columnName
```

Usa alias para la columna resultante.

Obtener de las Instituciones el dato de ubicación denominándolo ciudad.

```
MATCH (n: Institucion)
RETURN n.ubicacion AS ciudad
```

```
RETURN DISTINCT n
```

Devuelve filas únicas.

ORDER BY

```
ORDER BY n.propiedad
```

Ordenamiento de resultados.

```
ORDER BY n.propiedad DESC
```

Ordenamiento en orden descendente.

Obtener apellido y nombre de todas las personas de la red, ordenado por apellido descendente.

```
MATCH (a:Persona)
RETURN a.apellido, a.nombre
ORDER BY a.apellido DESC
```

En los ordenamientos, NULL siempre quedará al final en orden ascendente y al principio en orden descendente.

LIMIT

```
LIMIT {limit_number}
```

Limita el número de resultados.

Ídem anterior, pero limitando a 4 resultados.

```
MATCH (a:Persona)
RETURN a.apellido, a.nombre
ORDER BY a.apellido DESC
LIMIT 4
```

SKIP

```
SKIP {skip_number}
```

Saltea un número de resultados.

```
SKIP {skip_number} LIMIT {limit_number}
```

Saltea resultados y limita el número.

Ídem anterior pero salteando los primeros 3

```
MATCH (a:Persona)
RETURN a.apellido, a.nombre
ORDER BY a.apellido DESC
SKIP 3 LIMIT 4
```

WITH

La cláusula WITH separa explícitamente partes del query, permitiendo declarar qué identificadores arrastrar hacia la parte siguiente. Un uso habitual es para limitar la cantidad de resultados que pasarán a otra cláusula MATCH. Otro uso es para filtrar valores agregados.

Obtener la lista de personas y cantidad de conocimientos registrados de cada una de ellas, para las que registraron más de 3 conocimientos.

```
MATCH (a: Persona)-[:POSEE]->(c:Conocimiento)
WITH a, count(c) AS conocimientos
WHERE conocimientos > 3
RETURN a.apellido, a.nombre, conocimientos
```

También se puede combinar con ORDER BY, SKIP y LIMIT.

Adicionalmente, otro uso puede ser para separar cláusulas de lectura y escritura del grafo.

UNION

Combina los resultados de varios queries.

```
MATCH ...
UNION
MATCH ...
```

Retorna la unión (distinta) de todos los resultados de los queries.

```
MATCH ...
UNION ALL
MATCH ...
```

Retorna la unión de todos los resultados de los queries, incluyendo filas duplicadas.

Obtener los nombres de las personas que registraron un conocimiento o bien una carrera (en cualquier estado).

```
MATCH (p1:Persona)-[:ESTUDIO]->(ca:Carrera)
RETURN p1.apellido
UNION ALL
MATCH (p1:Persona)-[:POSEE]->(co:Conocimiento)
RETURN p1.apellido
```

Ídem, pero eliminando los duplicados.

```
MATCH (p1:Persona)-[:ESTUDIO]->(ca:Carrera)
RETURN p1.apellido
UNION
MATCH (p1:Persona)-[:POSEE]->(co:Conocimiento)
RETURN p1.apellido
```

Obtenemos también cuál es esa carrera o ese conocimiento

```
MATCH (p1:Persona)-[:ESTUDIO]->(ca:Carrera)
RETURN p1.apellido, ca.nombre
UNION
MATCH (p1:Persona)-[:POSEE]->(ca:Conocimiento)
RETURN p1.apellido, ca.nombre
```

Esquema

Neo4J 2.0 introdujo un esquema opcional para el grafo, basado en el concepto de las etiquetas. Las etiquetas se usan en la especificación de los índices y para definir restricciones en el grafo. Juntos, los índices y las restricciones son el esquema del grafo.

CREATE INDEX

```
CREATE INDEX ON :Etiqueta(propiedad)
```

Crea un índice para la etiqueta y propiedad especificadas. Los índices se crean en background, por lo que no están disponibles en forma inmediata.

```
MATCH (n:Etiqueta) WHERE n.propiedad = {valor}
```

Se puede usar un índice para comparaciones de igualdad. Sin embargo una comparación del tipo: lower(n.name) = {value} no usará un índice.

```
MATCH (n:Person)
USING INDEX n:Person(name)
WHERE n.name = {value}
```

Se puede forzar el índice a usar.

Crear un índice para los nodos de label Persona, sobre el campo apellido.

```
CREATE INDEX ON :Persona(apellido)
```

DROP INDEX

```
DROP INDEX ON :Person(name)
```

Elimina el índice para la etiqueta y propiedad especificadas.

CREATE CONSTRAINT

```
CREATE CONSTRAINT ON (p:Etiqueta)
       ASSERT p.propiedad IS UNIQUE
```

Crea una restricción de unicidad para esa etiqueta y propiedad. Si se actualiza o crea cualquier nodo con una propiedad que ya exista, fallará la operación de escritura. Este constraint creará su correspondiente índice.

Crear una restricción de unicidad para Persona y apellido.

```
CREATE CONSTRAINT ON (p:Persona)
       ASSERT p.apellido IS UNIQUE
```

¿Qué responde el motor?

Crear una para Empresa y nombre.

```
CREATE CONSTRAINT ON (p:Empresa)
       ASSERT p.nombre IS UNIQUE
```

Crear un nodo empresa con el nombre Accenture.

```
CREATE (n:Empresa {nombre: "Accenture"})
RETURN n
```

DROP CONSTRAINT

```
DROP CONSTRAINT ON (p:Etiqueta)
     ASSERT p.propiedad IS UNIQUE
```

Elimina la restricción de unicidad y el índice para la etiqueta y propiedad indicadas.

Consideraciones generales de performance

Utilizar parámetros en vez de literales cuando sea posible. Esto permite que Cypher reutilice los queries en vez de tener que parsearlos y construir nuevos planes de ejecución.

Siempre establecer límites superiores para los patrones de longitud variable. Es fácil que un query se convierta en un desastre y toque todos los nodos en el grafo por error.

Devolver sólo los datos necesarios. Evitar devolver nodos y relaciones completos. En cambio, sólo devolver lo que se necesita.

Manual de Referencia

http://docs.neo4j.org/chunked/milestone/index.html