Intro a Neo4J
Primeros pasos en la base
Ejemplos Alumnos y Préstamos de libros

Versión 1.1
Mayo 2017
Por Fernando Dodino

Índice

1 Ejemplo Alumnos

1.1 Enunciado

Queremos representar un curso de alumnos. Alumno - cursa - Curso.

- Alumno y Curso son nodos y “cursó” la relación.
- el Alumno tiene como atributos legajo, nombre, fecha de ingreso a la facultad, etc.
- cada Curso tiene una materia, el o los profesores que la dictan, el horario de dictado de clases, etc.
- El Alumno participa en n cursos
- Cada curso tiene n alumnos
- ¿Qué propiedades tiene la relación “cursó”? Podemos pensar en las notas, la fecha de firma, las asistencias, etc.

Escribiremos el ejemplo en notación de grafos de Neo4J:

`(a)`	Denota un nodo
`(a)-[r]->(b)`	Define una relación r entre dos nodos que llamamos a y b (a y b no son tipos sino nombres con los que podemos referenciar a los nodos, que sirven dentro del contexto de una consulta)
`(a)-[:CURSA]->(c)`	Indica que dos nodos previamente definidos cumplen la relación CURSA. Sabemos por las características del negocio que son alumnos que CURSAN un determinado curso: la relación va únicamente de alumno hacia curso.

`(a: Alumno)`
`  -[:CURSA]->`
`(c: Curso)`
	Mediante labels definimos restricciones de tipo sobre los nodos alumno y curso

1.2 Definición de nodos, arcos y propiedades

```
(phm {codigo:"K2052"})
```

Define un nodo (que representa un curso) con una propiedad código. Tenemos una variable ***phm*** que referencia a ese nodo. La variable es una referencia cuyo ciclo de vida está atado a la ejecución de una consulta en Cypher (solo es válida mientras ejecutamos una instrucción, luego la referencia se pierde).

Podemos etiquetar a phm como un nodo Curso, si le asignamos un *label*:

```
(phm: Curso {codigo:"K2052"})
```

```
(miguel
   {
      nombre: "Miguel Angel Gagliardo",
      legajo: 199532
   }
)
```

Definimos un nodo (que representa un alumno) con dos propiedades, tenemos una variable ***miguel*** que referencia a ese nodo temporalmente.

Ahora agregaremos la relación CURSA con las notas, relación que va desde el nodo alumn@ hacia el curso:

```
(miguel)-[:CURSA {notas:[7, 10]}]->(phm)
```

1.3 Primer ejemplo

Te recomendamos que sigas

- esta presentación
- el ejemplo completo de Alumnes en Github
- y para consultas más elaboradas está esta pregunta de Stack Overflow.

2 Ejemplo préstamos de libros

2.1 Enunciado

Vamos a modelar los préstamos de libros. Aparecen como entidades:

El libro que tiene

- autor
- título

El usuario que tiene

- nombre

Y el préstamo relaciona libro y usuario, donde además queremos saber

- fecha en que lo prestamos
- fecha en que lo devolvió

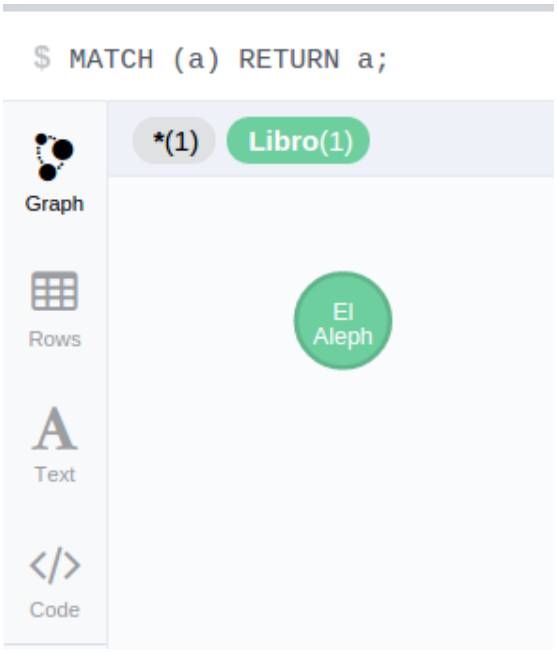
2.2 Definición de nodos libro

Una base de grafos se compone de nodos y relaciones. Los nodos tienen labels y propiedades. El label permite definir un tipo. Creamos nuestro primer nodo, el libro "El Aleph"

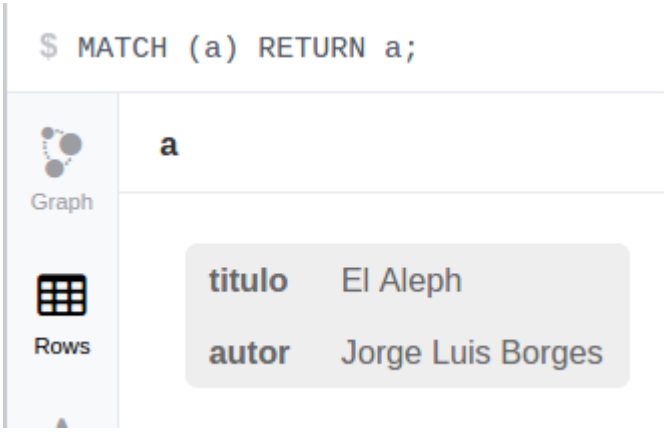
```
CREATE (elAleph:Libro
   {autor:'Jorge Luis Borges',
    titulo: 'El Aleph'}
)
```

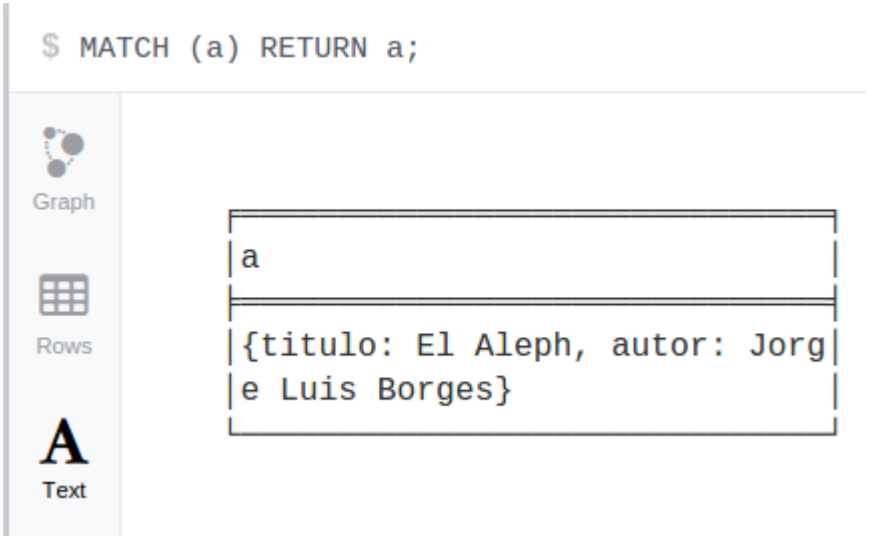
Una vez creado, podemos buscarlo en la base de grafos mediante Cypher:

```
MATCH (a) RETURN a;  // busca todos los nodos
```



Tenemos varias formas de visualizar la información:





En principio tenemos conceptos que matchean bastante bien el modelo de objetos:

Concepto en Neo4J	Concepto en Objetos
nodo	objeto
label	tipo/clase
propiedades: un mapa con valores	el mismo concepto

Seguimos creando otros 5 libros:

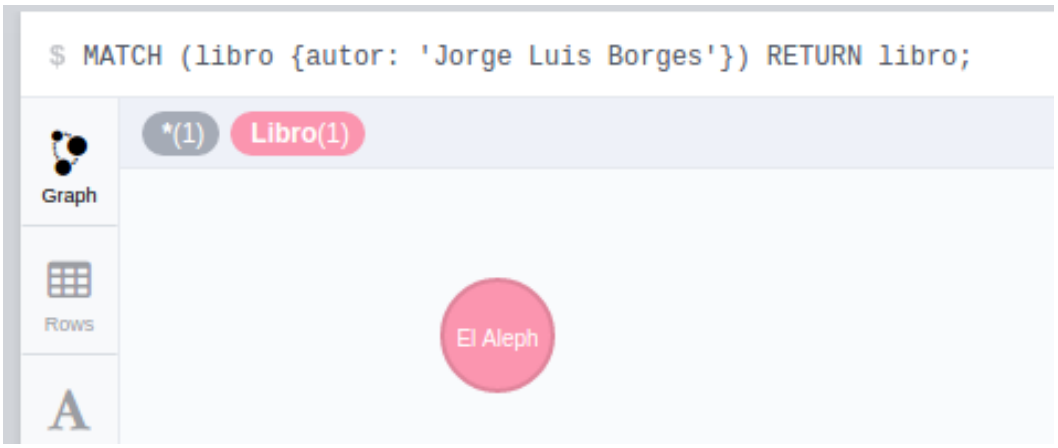
```
CREATE (noHabra:Libro {autor:'Osvaldo Soriano', titulo: 'No habrá más penas ni olvido'})

CREATE (cienAnios:Libro {autor:'Gabriel García Márquez', titulo: '100 años de soledad'})

CREATE (novelaPeron:Libro {autor:'Tomás Eloy Martínez', titulo: 'La novela de Perón'})

CREATE (porQuien:Libro {autor:'Ernest Hemingway', titulo: '¿Por quién doblan las campanas?'})
```

Ahora tenemos los 5 libros ubicables como nodos

```
MATCH (libro: Libro) RETURN libro; // busca todos los nodos libro
```

Podemos buscar libros de Borges:

```
MATCH (libro {autor: 'Jorge Luis Borges'}) RETURN libro;
```



También libros que contengan la palabra “de” en el título:

```
MATCH (libro: Libro) WHERE libro.titulo =~ '.*de.*' RETURN libro;
```

En el último query pedimos nodos Libro específicamente. Por ahora no hay diferencia, a futuro sí vamos a tener diferentes nodos.

2.3 Nodos usuario

Tendremos en principio cuatro usuarios: Lampone, Ravenna, Medina y Santos. Los creamos:

```
CREATE (lampone:Usuario {nombre:'Pablo Lampone'})

CREATE (medina:Usuario {nombre:'Gabriel Medina'})

CREATE (santos:Usuario {nombre:'Mario Santos'})

CREATE (ravenna:Usuario {nombre:'Emilio Ravenna'})
```

Al crear estos cuatro nodos, nuestra consulta en Cypher de todos los nodos ahora devuelve tanto libros como usuarios:

Podemos discriminar los nodos usuarios pidiendo el label específico:

```
MATCH (usuario: Usuario) RETURN usuario;
```

2.4 Relaciones

Pero además de los nodos, las bases de grafos trabajan con relaciones (también llamadas arcos), que tienen

- dirección, algo que comparten con el modelo de objetos. Cada relación tiene una flecha que marca el nodo origen y el destino, para que una relación sea bidireccional eso implica generar dos relaciones.
- propiedades, algo que objetos en principio no tiene.

En nuestro caso, vamos a modelar al préstamo como una relación, no un nodo. En esa relación participan dos propiedades:

1. fecha de préstamo
2. fecha de devolución

En nuestro ejemplo vamos a establecer

- que a Santos le prestamos El Aleph
- que Medina se llevó 100 años de soledad y lo devolvió
- y que Ravenna tiene el libro "¿Por quién doblan las campanas?"

Un detalle importante, Neo4J no soporta Date como tipo de dato, utilizaremos un String con formato dd/mm/yyyy.

Lo que sí necesitamos es acceder a los nodos para poder relacionarlos. Probamos nuestro query que encuentra a Mario Santos:

```
MATCH (santos: Usuario {nombre: 'Mario Santos'}) RETURN santos;
```

También el libro El Aleph:

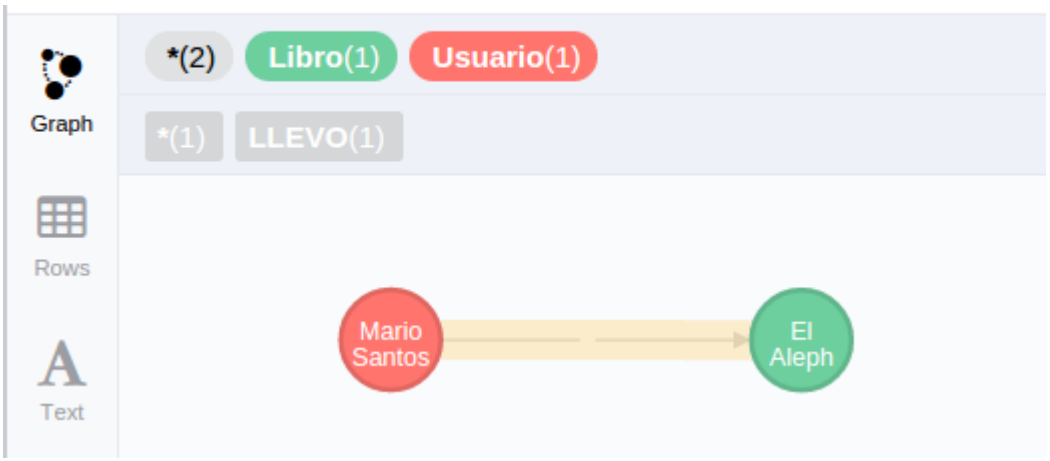
```
MATCH (elAleph: Libro {titulo: 'El Aleph'}) RETURN elAleph;
```

Ahora relacionaremos a Mario Santos con El Aleph:

```
MATCH
    (santos: Usuario {nombre: 'Mario Santos'}),
    (elAleph: Libro {titulo: 'El Aleph'})
CREATE (santos)
           -[:LLEVO {fechaPrestamo: '01/05/2016'}]->
       (elAleph);
```

Para visualizar la relación, debemos buscar los nodos usuario relacionados con nodos libro:

```
MATCH (usuario:Usuario)-[llevo]->(libro: Libro) RETURN usuario,
libro;
```



Vemos que hay

- 1 nodo libro en verde
- 1 nodo usuario en rojo
- 1 relación llevó

Si queremos ver información sobre la relación, debemos retornar la información:

```
MATCH (usuario:Usuario)-[llevo]->(libro: Libro) RETURN usuario, llevo, libro;
```

Agregamos ahora la información del libro que se llevó y devolvió Medina...

```
MATCH
        (medina: Usuario {nombre: 'Gabriel Medina'}),
        (cienAnios: Libro {titulo: '100 años de soledad'})
CREATE (medina)
           -[:LLEVO {fechaPrestamo: '04/05/2016', fechaDevolucion:
'14/06/2016'}]->
       (cienAnios);
```

Y por último Emilio Ravenna se llevó ¿Por quién doblan las campanas?

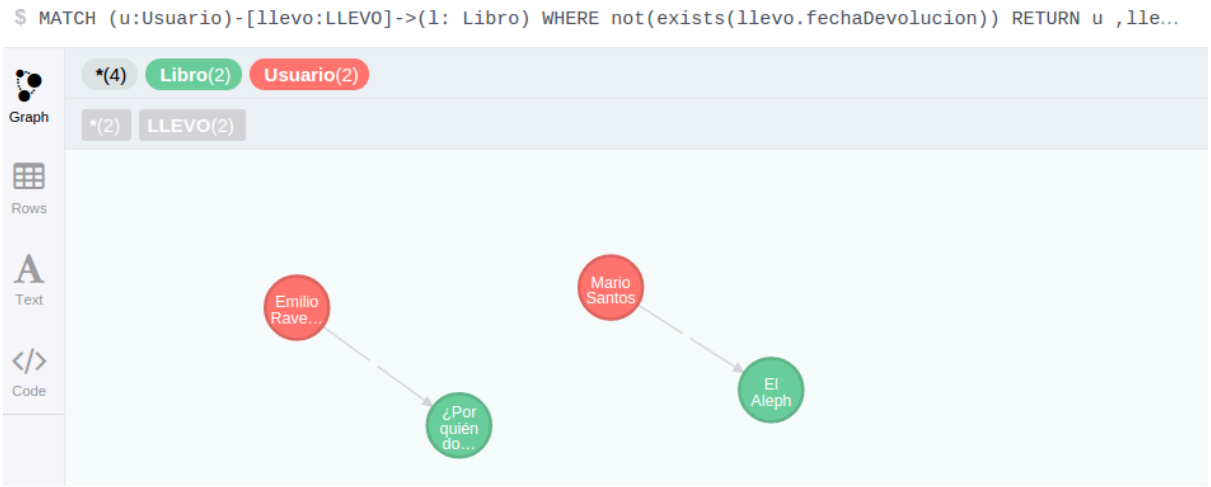
```
MATCH (ravenna: Usuario {nombre: 'Emilio Ravenna'}),
      (porQuien: Libro {titulo: '¿Por quién doblan las campanas?'})
CREATE (ravenna)
          -[:LLEVO {fechaPrestamo: '22/05/2016'}]->
```

```
        (porQuien);
```

2.5 Consultas

Podemos ver qué préstamos están pendientes, son aquellas relaciones LLEVO entre un nodo Usuario y un nodo Libro que no tienen la propiedad fecha de devolución:

```
MATCH (usuario:Usuario)-[llevo:LLEVO]->(libro: Libro)
      WHERE not(exists(llevo.fechaDevolucion))
RETURN usuario, llevo, libro;
```



La consulta pregunta por la existencia de un campo, en lugar de hacer la pregunta por un valor nulo en el campo fechaDevolucion, por la naturaleza desestructurada de las bases de grafos.

También podemos saber qué libros leyó un usuario:

```
MATCH (usuario:Usuario {nombre: 'Mario Santos'})
      -[llevo:LLEVO]->
      (libro: Libro)
RETURN usuario, llevo, libro;
```

Qué usuarios leyeron un libro:

```
MATCH (usuario: Usuario)-[llevo:LLEVO]->(libro: Libro {titulo: '100
años de soledad'})
      RETURN usuario, llevo, libro;
```

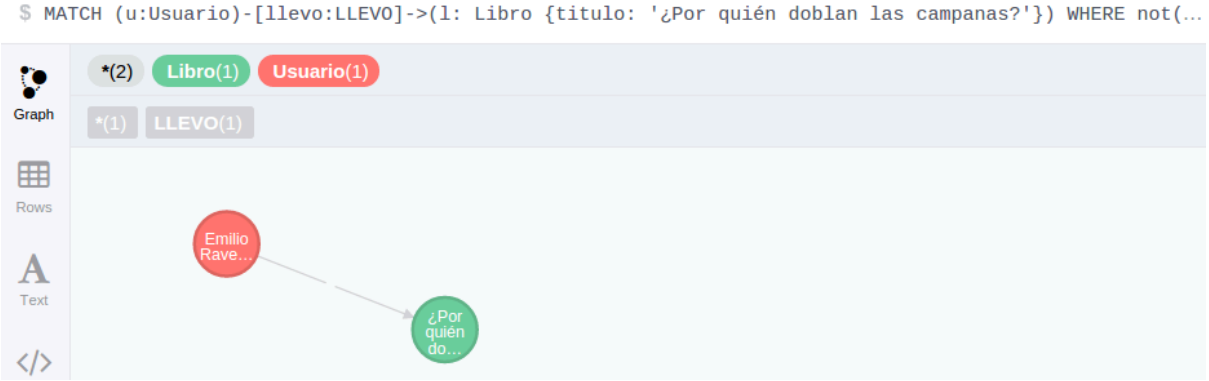
¿Y quién tiene un libro?

```
MATCH (usuario:Usuario)-[llevo:LLEVO]->(libro: Libro {titulo: '100
años de soledad'})
      WHERE not(exists(llevo.fechaDevolucion))
      RETURN usuario, llevo, libro;
```

Claro, como Medina devolvió el libro esta consulta no devuelve filas. En cambio si preguntamos por ¿Por quién doblan las campanas?

```
MATCH (usuario:Usuario)-[llevo:LLEVO]->(libro: Libro {titulo: '¿Por
quién doblan las campanas?'})
      WHERE not(exists(llevo.fechaDevolucion))
      RETURN usuario, llevo, libro;
```

Aparece Ravenna...

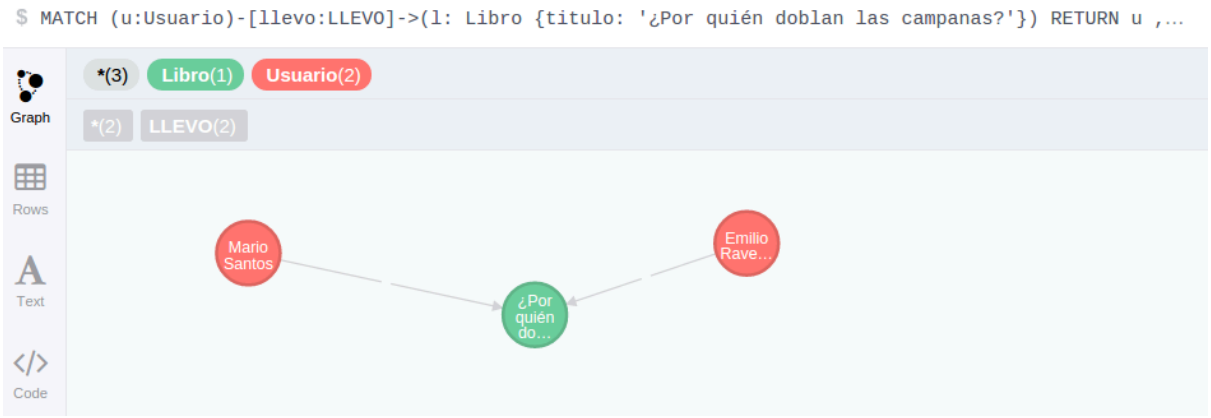


De hecho si Ravenna devuelve el libro y se lo lleva Santos...

```
MATCH (u:Usuario)-[llevo:LLEVO]->(l: Libro {titulo: '¿Por quién
doblan las campanas?'})
      WHERE not(exists(llevo.fechaDevolucion))
      SET llevo.fechaDevolucion = '07/06/2016';

MATCH (santos: Usuario {nombre: 'Mario Santos'}),
      (porQuien: Libro {titulo: '¿Por quién doblan las campanas?'})
      CREATE (santos)-[:LLEVO {fechaPrestamo:
'08/06/2016'}]->(porQuien);
```

Podemos ver el historial de los que fueron leyendo ¿Por quién doblan las campanas?

```
MATCH (usuario:Usuario)-[llevo:LLEVO]->(libro: Libro {titulo: '¿Por
quién doblan las campanas?'})
      RETURN usuario, llevo, libro;
```



Es decir, si bien la relación *llevó* es dirigida de un nodo origen a uno destino, podemos conocer a partir de ellas información de los usuarios o de los libros.

3 Más información

Recomendamos

- https://docs.spring.io/spring-data/neo4j/docs/6.0.x/reference/html/#auditing (el ejemplo de Kevin Bacon)
- http://neo4j.com/docs/cypher-refcard/current/
- https://neo4j.com/docs/cypher-manual/current/introduction/
- https://neo4j.com/developer/cypher-query-language/