**Sistemas de Información de Gestión y**

**Bussiness Intelligence**

**11.11.2019**

He estado investigando los diferentes tipos de software que podemos aplicar en esta asignatura. En primer lugar, he visitado, tal y como hicimos en clase, DB-Engines (<https://db-engines.com/en/>) para observar qué tipos de bases de datos son las qué mas se utilizan y por qué motivos. Aquí es cuando observé que aparece Neo4j como el número 1 en base de datos de grafos. Como el profesor nos lo mencionó está herramienta en clase decidí buscar información en Google acerca de Neo4j y cómo funciona.

Neo4j es una base de datos de grafos optimizada para relacionar de forma fácil y rápida la información. Esta herramienta permite ejecutar sentencias con 100 millones de nodos en un tiempo corto comparado con otro tipo de base de datos. El lenguaje de esta herramienta es Cypher.

Para entender mejor el funcionamiento de Neo4j he visualizado el siguiente video:

(<https://www.youtube.com/watch?v=k3h_y9w_7l4>**)**

Gracias al vídeo o comprendido como realizar sentencias en Neo4j de forma sencilla. También he entendido que a las relacionas se les trata como ciudadanos de primera clase de forma que tienen la misma relevancia que los propios nodos. Un ejemplo claro sería el siguiente:

C**REATE (m:Player{name:"Lionel Messi"}),**

**(b:Team{name:"Barcelona"})**

**WITH m, b**

**CREATE (m)-[p:PlaysFor]->(b)**

**SET p.since=date("2001-02-01")**

**RETURN m, p, b**

De esta forma tendríamos un jugador llamado Messi que juega en el Barcelona desde el 1-2-2001.

Ahora bien, una vez que he aprendido las bases de Neo4j me descargué el programa para poder realizar mis propios grafos. Una vez todo instalado y configurado utilicé el segundo ejemplo donde se muestra un conjunto de personas y sus amigos:

// Create nodes

CREATE (rob:Person{name:'Roberto'}), (isidro:Person{name:'Isidro'}),

(tony:Person{name:'Antonio'}), (nora:Person{name:'Nora'}),

(lily:Person{name:'Lilian'}), (freddy:Person{name:'Alfredo'}),

(lucas:Person{name:'Lucas'}), (mau:Person{name:'Mauricio'}),

(alb:Person{name:'Albina'}), (reg:Person{name:'Regina'}),

(j:Person{name:'Joaquín'}), (julian:Person{name:'Julián'})

// Create relationships

CREATE

(rob)-[:FriendsWith]->(isidro), (rob)-[:FriendsWith]->(tony), (rob)-[:FriendsWith]->(reg),

(rob)-[:FriendsWith]->(mau), (rob)-[:FriendsWith]->(julian),

(tony)-[:FriendsWith]->(reg), (tony)-[:FriendsWith]->(j),

(alb)-[:FriendsWith]->(reg), (lily)-[:FriendsWith]->(isidro), (lily)-[:FriendsWith]->(j),

(mau)-[:FriendsWith]->(lucas), (lucas)-[:FriendsWith]->(nora), (freddy)-[:FriendsWith]->(nora);

// Query the relationships

MATCH friendships=()-[:FriendsWith]-()

RETURN friendships

Como bien se puede observar para realizar la búsqueda de todos los amigos de todas las personas es simple. Si solo quisiéramos saber los amigos de amigos solo tendríamos que especificar en primer lugar que persona tomamos como referencia para buscar los amigos de sus amigos. Posteriormente añadimos un \*2 dentro de los corchetes de FriendsWith de forma que no cogería todos los amigos sino los amigos que están en el segundo círculo de la persona de referencia.

Para poder conseguir el mejor resultado en la asignatura me dispongo a ver todos los videos del canal Tensor4Dummies de forma que pue3de utilizar tensorflow de forma fácil y también me dispongo a estudiar más en profundidad como programar en Python.

19.11.2019

He empezado el curso de introducción a Neo4j lo cual me permitirá adquirir los conocimientos necesario para realizar la base de datos de grafos de forma correcta.

Las ventajas de utilizar estos tipos de base de datos respecto a utilizar bases de datos relacionales:

- Es más intuitiva a la hora de buscar en la base de datos algún dato.

- Es más ágil dado que da una mayor facilidad de modificar algún dato de la base de datos.

Los compenetes de la base de datos relacional en una de grafos pasarían a ser los siguientes elementos:

- Las líneas o registros de las tablas pasan a ser nodos.

- Los joins pasan a ser relaciones.

- Los nombres de las tablas pasarán a ser Etiquetas/Labels.

- Cada columna de una tabla será una property.

Neo4j permite crear una base de datos de ejemplo acerca de peliculas para poder aprender Cypher de forma rápida. Para acceder al ejemplo se debe utilizar el comando:

:play Movie-Graph.

Dejaré justo de aquí en adelante varias cosas del tutorial de Neo4j para, si me hace falta, pueda revisar la información en busca de resolver alguna duda:

() //un nodo

()--() // 2 nodos tienen una relación

()-->() // el primer nodo tiene uan relación hacía el Segundo nodo

()<--() // el segundo nodo tiene una relación hacia el primero

Query con varias relaciones (Tom Hanks actua en y dirige):

MATCH (p:Person {name: 'Tom Hanks'})-[:ACTED\_IN | :DIRECTED]->(m:Movie)

RETURN p.name, m.title

Buscar el nombre de los actores de Matrix sin necesidad de obtener la información de la película:

MATCH (p:Person)-[:ACTED\_IN]->(:Movie {title: 'The Matrix'})

RETURN p.name

Buscar personas que estén relacionadas de alguna forma con la película de Matrix:

MATCH (p:Person)-->(m:Movie {title: 'The Matrix'})

RETURN p, m

Búsqueda del nombre de una persona que haya puntuada la película del código da vinci:

MATCH (p:Person)-[:REVIEWED {rating: 65}]->(:Movie {title: 'The Da Vinci Code'})

RETURN p.name

Query para obtener los nodos de personas que siguen a otras personas porque es para cualquier dirección:

MATCH (p1:Person)-[:FOLLOWS]-(p2:Person {name:'Angela Scope'})

RETURN p1, p2

Búsqueda con la sentencia WHERE:

MATCH (p)-[:ACTED\_IN]->(m)

WHERE p:Person AND m:Movie AND m.title='The Matrix'

RETURN p.name

Para devolver informacion de la película con un campo tagline existente

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name='Jack Nicholson' AND exists(m.tagline)

RETURN m.title, m.tagline

Uso de palabras clave cuando se trata de comparar cadenas de texto (STARTS WITH, ENDS WITH, and CONTAINS):

MATCH (p:Person)-[:ACTED\_IN]->()

WHERE p.name STARTS WITH 'Michael'

RETURN p.name

La comparación a la hroa de usar cadenas tiene en cuenta si son mayúsculas o no las letras, para evitar eso podemos usar toLower:

MATCH (p:Person)-[:ACTED\_IN]->()

WHERE toLower(p.name) STARTS WITH 'michael'

RETURN p.name

Testing with regular expressions

También se puede hacer búsquedas donde se haga una búsqueda parcial por ejemplo cuando buscamos las personas cuyo nombre empiece por Tom:

MATCH (p:Person)

WHERE p.name =~'Tom.\*'

RETURN p.name

Misma búsqueda que antes, pero excluyendo las personas que hayan dirigido la película:

MATCH (p:Person)-[:WROTE]->(m:Movie)

WHERE NOT exists( (p)-[:DIRECTED]->(m) )

RETURN p.name, m.title

Búsqueda de los nodos y relaciones donde Gene Hackman actúa en una película junto con otro actor y dicho actor dirige la película

MATCH (gene:Person)-[:ACTED\_IN]->(m:Movie)<-[:ACTED\_IN]-(other:Person)

WHERE gene.name= 'Gene Hackman'

AND exists( (other)-[:DIRECTED]->(m) )

RETURN gene, other, m