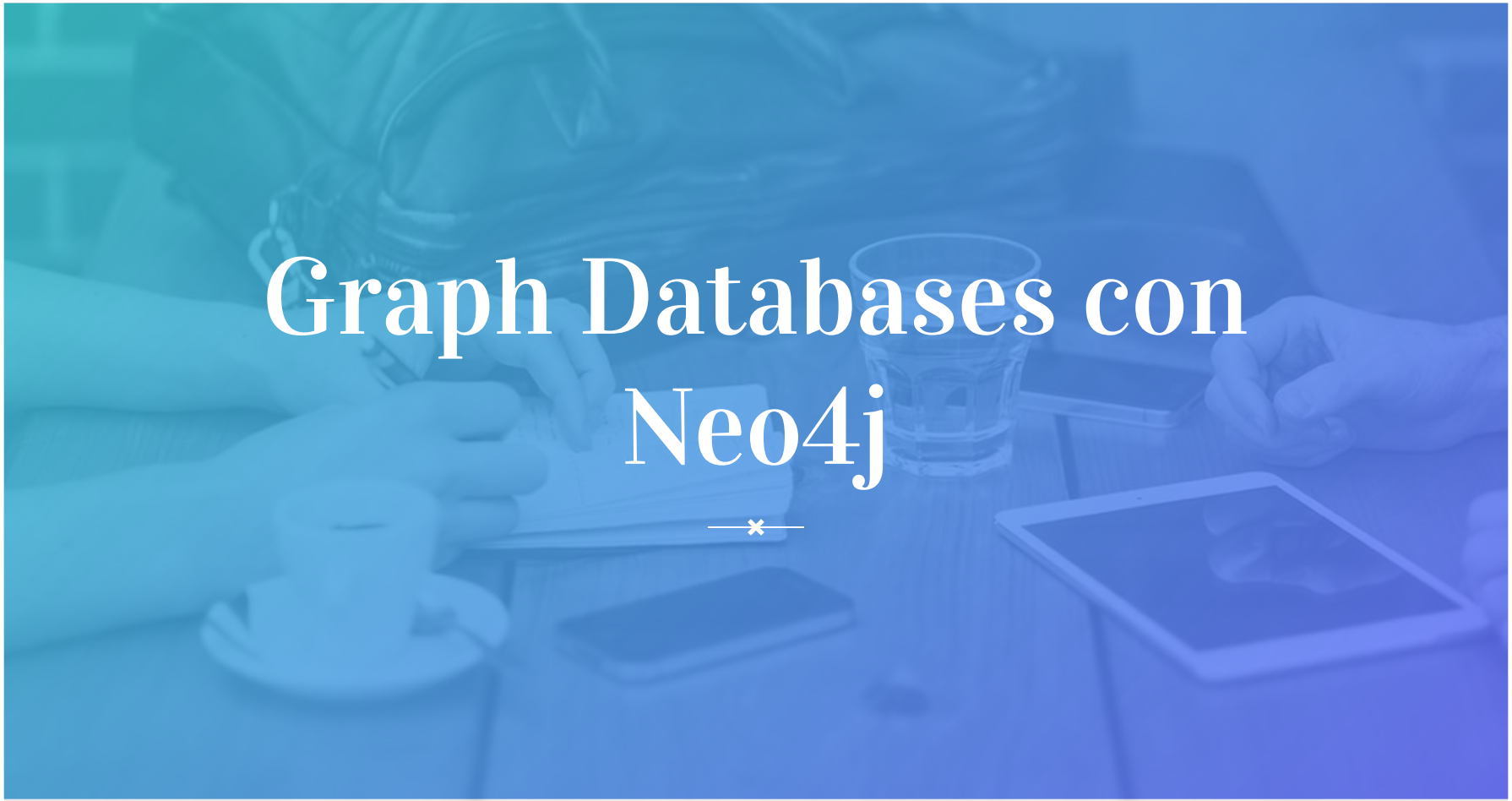


Graph Databases con Neo4j

Hello!

Soy Pedro Diez

Hoy me han liado para la charla, pero yo normalmente soy Ingeniero de Backend Senior en AttackIQ. Trabajo sobre todo python y frameworks como FastAPI y Django.

: pedro.d.diez@gmail.com

: @pedroddiez

: PedroDDiez

¿Qué es una Graph Database?

¿Alguien se atreve? ¿Alguna idea por ahí?

“

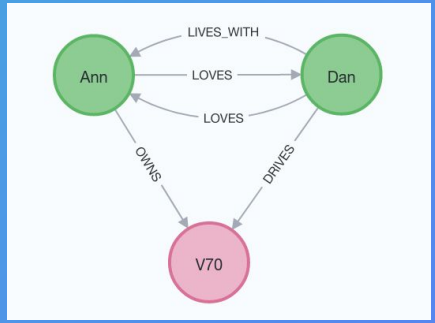
Una base de datos de Grafos almacena nodos y relaciones en vez de tablas o documentos.

¿Qué es una Graph Database?

- Almacena nodos y relaciones
- Define propiedades asociadas a estos
- No hace falta definir un modelo.
- Podemos trabajar con ellas como con una pizarra para bocetos

¿Qué es una Graph Database?

Hoy en día, para saber qué está pasando hay que procesar y comprender grandes conjuntos de conexiones.

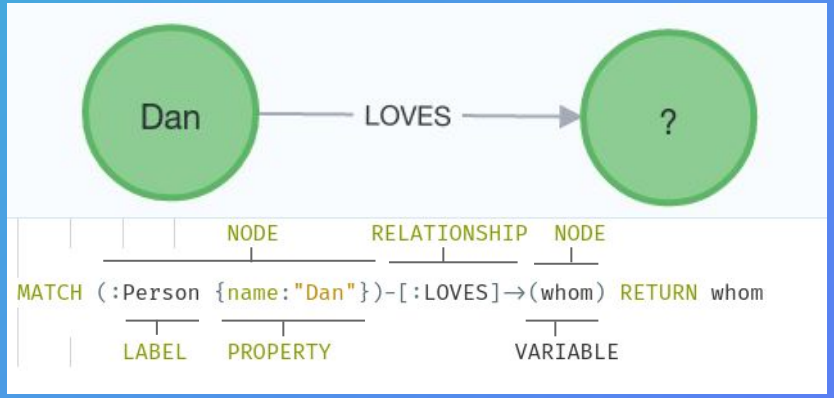


¿Por qué una Graph Database?

En bases de datos relacionales también existen relaciones y se puede navegar por ellas con JOINS, pero estos pueden volverse ineficientes cuando las relaciones crecen.

El modelo de grafo con propiedades

En Neo4j la información se organiza en nodos, relaciones y propiedades



El modelo de grafo con propiedades

Los ***nodos*** son las entidades en el grafo

- Se pueden etiquetar con *labels* que representan los distintos roles de tu dominio
- Pueden tener cualquier número de pares clave-valor, o *properties*
- Asociado a las *labels* podemos tener metadata como índices, restricciones...

El modelo de grafo con propiedades

Las ***relaciones*** son conexiones con nombre y dirección entre dos nodos.

- Siempre tienen dirección, tipo y pueden tener properties
- Los nodos pueden tener todas las relaciones que sean necesarias
- Se pueden navegar en cualquier dirección

Veamos un ejemplo

NorthWind dataset

Utilizaremos el dataset de NorthWind, donde tenemos datos relacionados a un comercio, con clientes, pedidos , productos, stock...

Este dataset podemos descargarlo en
https://github.com/neo4j-documentation/developer-resources/raw/gh-pages/data/northwind/northwind.zip

Diagrama entidad-relación de NorthWind

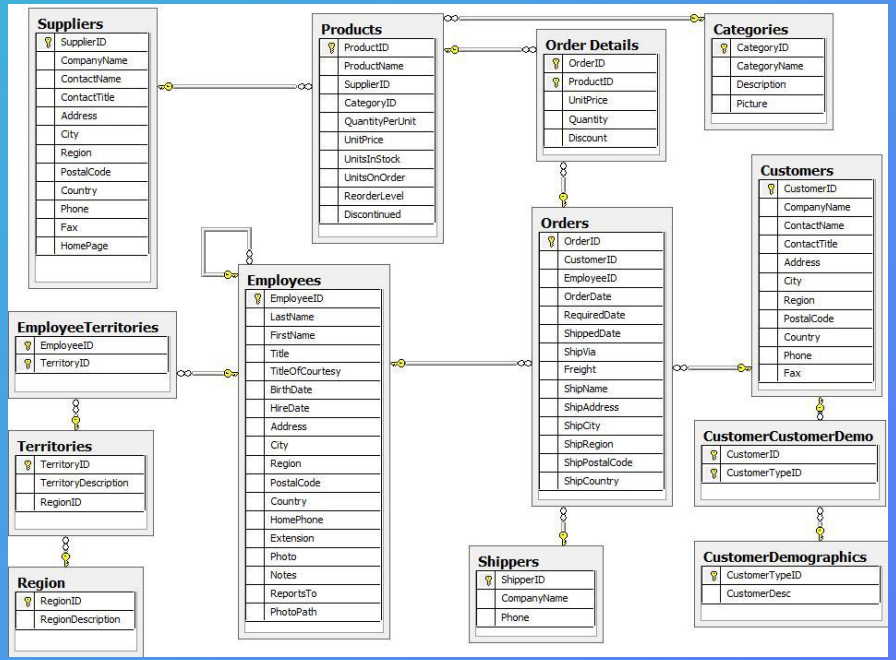
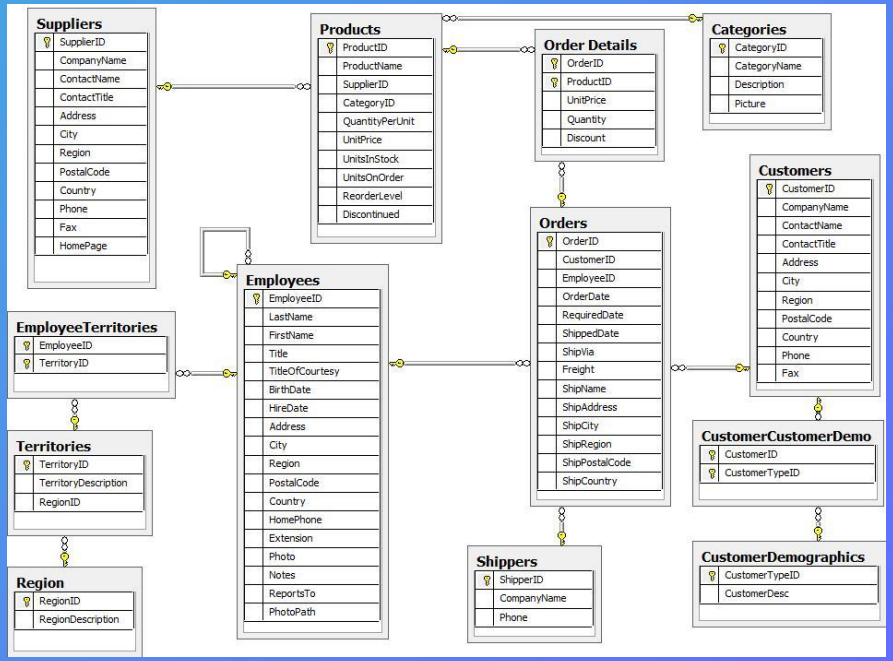


Diagrama entidad-relación de NorthWind

Solo vamos a usar una parte del dataset: Suppliers, Products, Categories, Employees, Orders y ~~Order_details~~



Desarrollando el Modelo del Grafo

Empezaremos traduciendo el modelo de datos relacional a uno de grafo. Este modelo estará relacionado con las necesidades del negocio.

Por lo general, podemos usar estas guías:

- Una fila es un nodo
- Una tabla (su nombre) es una etiqueta
- Un join o foreign key es una relación

Desarrollando el Modelo del Grafo

En nuestro caso convertiremos las siguientes filas a nodos:

- *Orders* como nodos *Order*
- *Products* como nodos *Product*
- *Suppliers* como nodos *Supplier*
- *Categories* como nodos *Category*
- *Employees* como nodos *Employee*

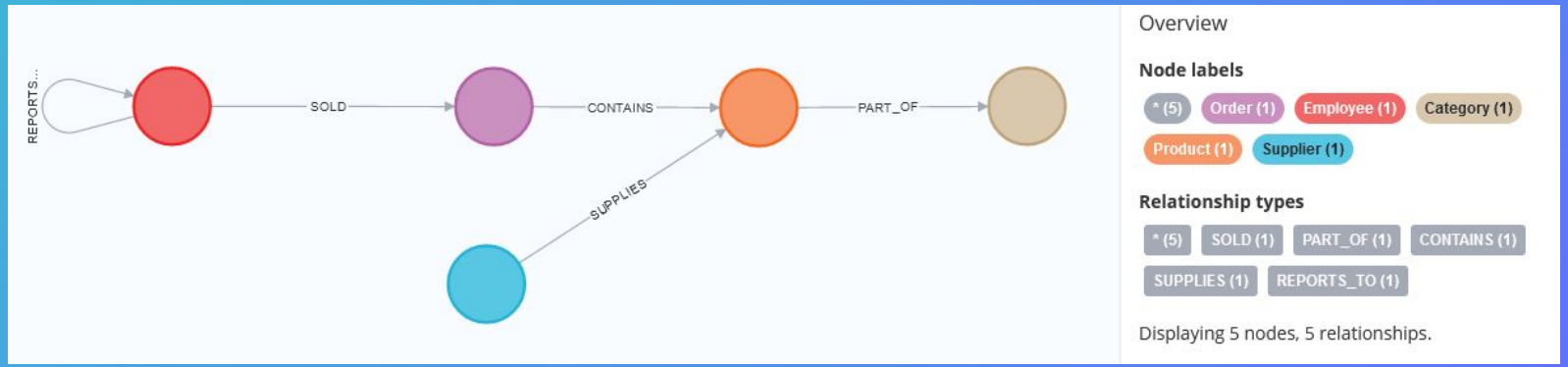
Desarrollando el Modelo del Grafo

En nuestro caso crearemos las siguientes relaciones:

- Entre *Suppliers* y *Products* crearemos *SUPPLIES*
- Entre *Products* y *Categories* crearemos *PART_OF*
- Entre *Employees* y *Orders* crearemos *SOLD*
- Entre *Empleados* crearemos *REPORTS_TO*
- Entre *Orders* y *Productos* crearemos *CONTAINS* con propiedades como *quantity, discount, ...*

Desarrollando el Modelo del Grafo

Una vez que el grafo esté creado, tendrá una estructura similar a esta:



Importando los datos con Cypher

Ahora que tenemos las nociones básicas, vamos a importar datos.

Utilizaremos los siguientes archivos del repo:

- create_nodes.cypher
- create_relationships.cypher
- create_indices.cypher

Importando los datos con Cypher

Creando nodos con *create_nodes.cypher*

```
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
```

Importando los datos con Cypher

Creando relaciones con *create_relationships.cypher*

```
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
neo4j$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/PedroDDiez/d2403112fca28d9ac186e0efc67e8240/raw/f42c314615…
```

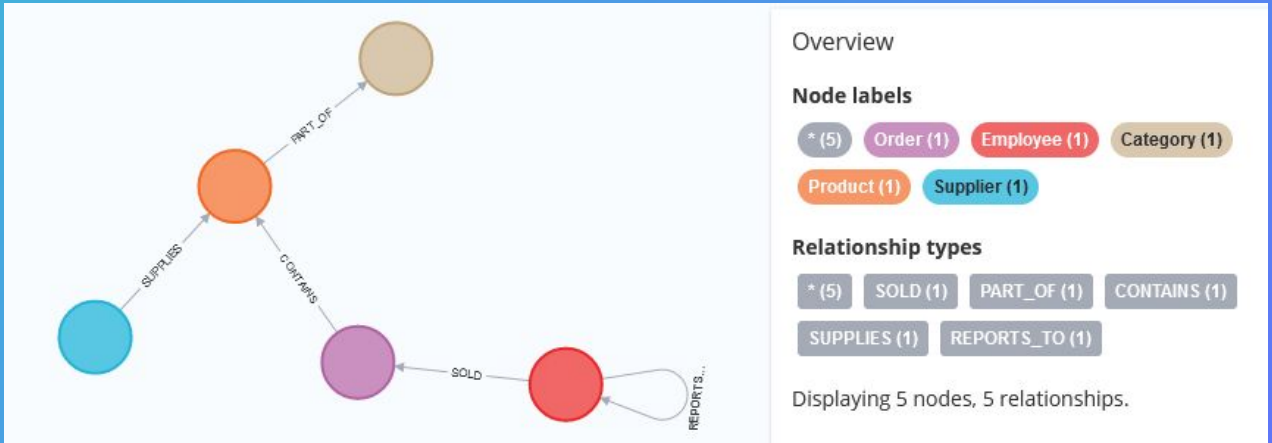
Importando los datos con Cypher

Creando los índices con *create_indices.cypher*

```
neo4j$ CREATE INDEX product_id FOR (p:Product) ON (p.productID)
neo4j$ CREATE INDEX product_name FOR (p:Product) ON (p.productName)
neo4j$ CREATE INDEX supplier_id FOR (s:Supplier) ON (s.supplierID)
neo4j$ CREATE INDEX employee_id FOR (e:Employee) ON (e.employeeID)
neo4j$ CREATE INDEX category_id FOR (c:Category) ON (c.categoryID)
neo4j$ CREATE CONSTRAINT order_id FOR (o:Order) REQUIRE o.orderID IS UNIQUE
neo4j$ CALL db.awaitIndexes()
```

Importando los datos con Cypher

Ya tenemos todo importado, miremos que pinta tiene con `CALL db.schema.visualization()`



Consultando el grafo

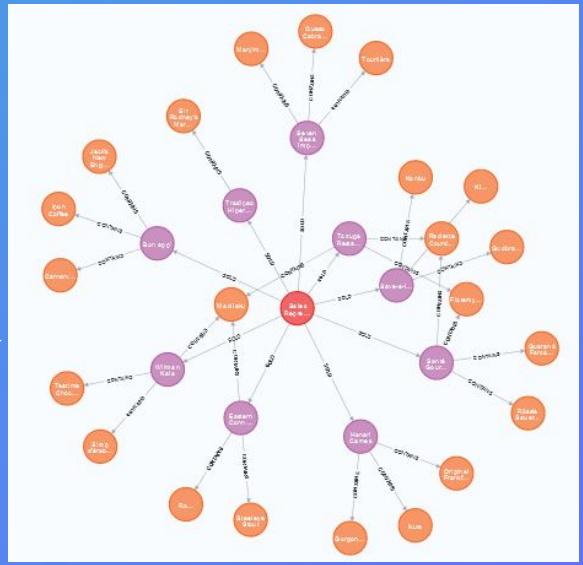
Ahora que tenemos datos, podemos empezar a escribir queries.

Veamos algunos ejemplos.

Consultando el grafo

Busquemos una muestra de empleados que vendieron pedidos con sus productos asociados:

```
MATCH
(e:Employee)-[rel:SOLD]->(o:Order)-[rel2:CONTAINS]->(p:Product
) RETURN e, rel, o, rel2, p LIMIT 25;
```

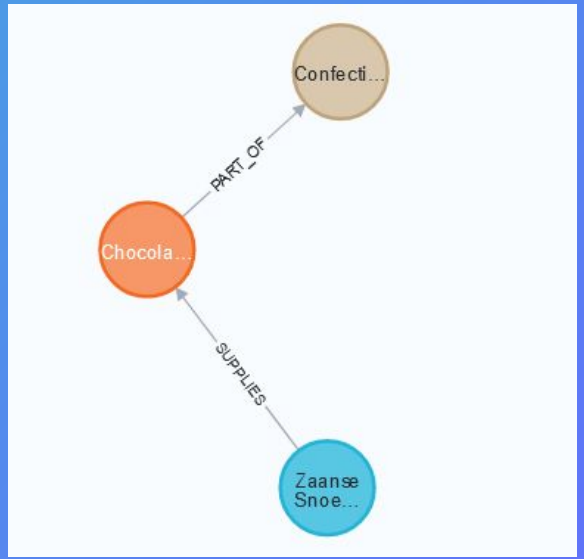


Consultando el grafo

Busquemos el proveedor y categoría de un producto:

```
MATCH (s:Supplier)-[r1:SUPPLIES]->(p:Product {productName:
'Chocolade'})-[r2:PART_OF]->(c:Category)

RETURN s, r1, p, r2, c;
```



Consultando el grafo

Quien ha hecho mayor cross-selling de “Chocolade”:

```
MATCH (choc:Product {productName:'Chocolade'})<-[:CONTAINS]-(:Order)<-[:SOLD]-(employee),
      (employee)-[:SOLD]->(o2)-[:CONTAINS]->(other:Product)
RETURN employee.employeeID as employee, other.productName as otherProduct, count(distinct o2) as
count ORDER BY count DESC LIMIT 5;
```

employee	otherProduct	count
4	"Gnocchi di nonna Alice"	14
3	"Gumbär Gummibärchen"	12
4	"Pâté chinois"	12
1	"Flotemysost"	12
1	"Pavlova"	11

Consultando el grafo

Como se estructura la organización:

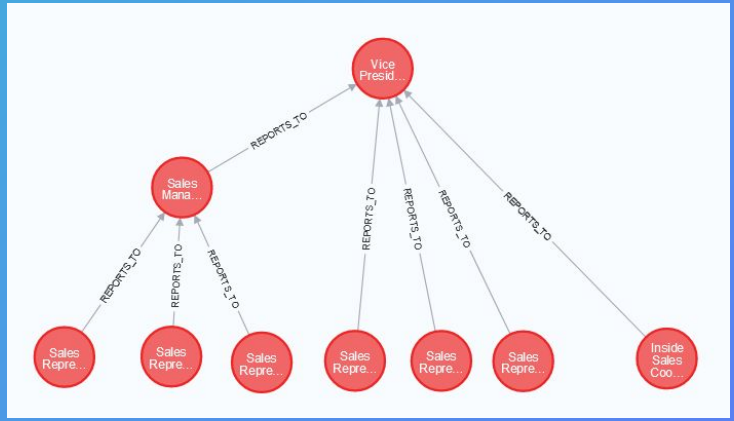
```
MATCH (e:Employee)<-[:REPORTS_TO]-(sub)
RETURN e.employeeID AS manager, sub.employeeID AS employee;
```

manager	employee
"2"	"5"
"2"	"3"
"2"	"1"
"2"	"8"
"2"	"4"
"5"	"9"
"5"	"6"
"5"	"7"

Consultando el grafo

Como se estructura la organización (mejor así):

```
MATCH (e:Employee)<-[:REPORTS_TO]-(sub)
RETURN e.employeeID AS manager, sub.employeeID AS employee;
```



Thanks!

Creo que con esto os podéis hacer una idea.

¿Alguna pregunta?

Puedes escribirme a @pedroddiez & pedro.d.diez@gmail.com