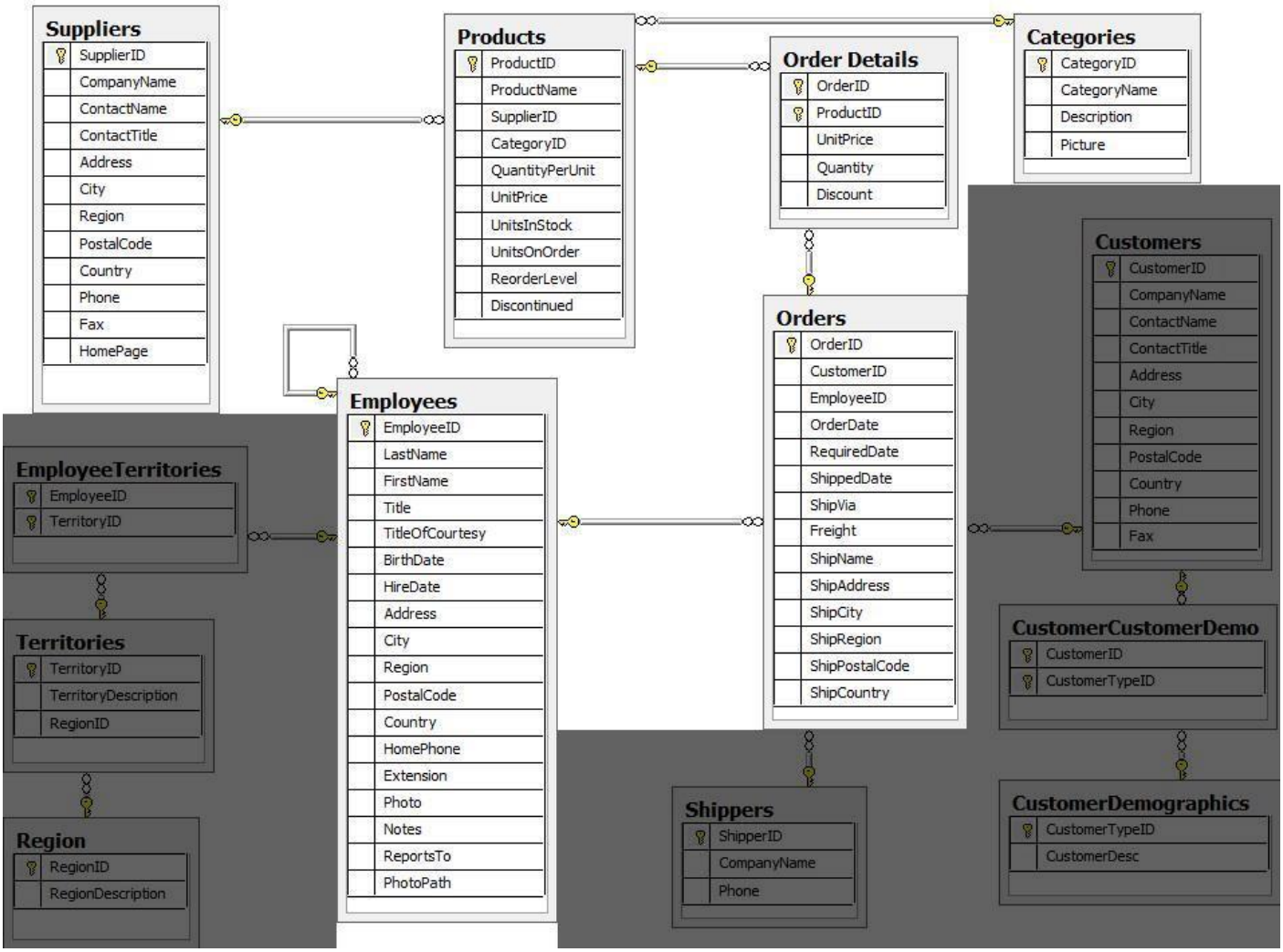
Como migrar una base de datos relacional "POSTGRES a Neo4J"

Nombre: Luis Orellana P.

Diagrama entidad-relacion



Pautas generales

- Una fila es un nodo
- Un nombre de tabla es un nombre de etiqueta
- Una combinación o clave externa es una relación

Pasos para mapear nuestro modelo relacional a un gráfico

Filas a nodos, nombres de tablas a etiquetas

Con estos principios en mente, podemos mapear nuestro modelo relacional a un gráfico con los siguientes pasos:

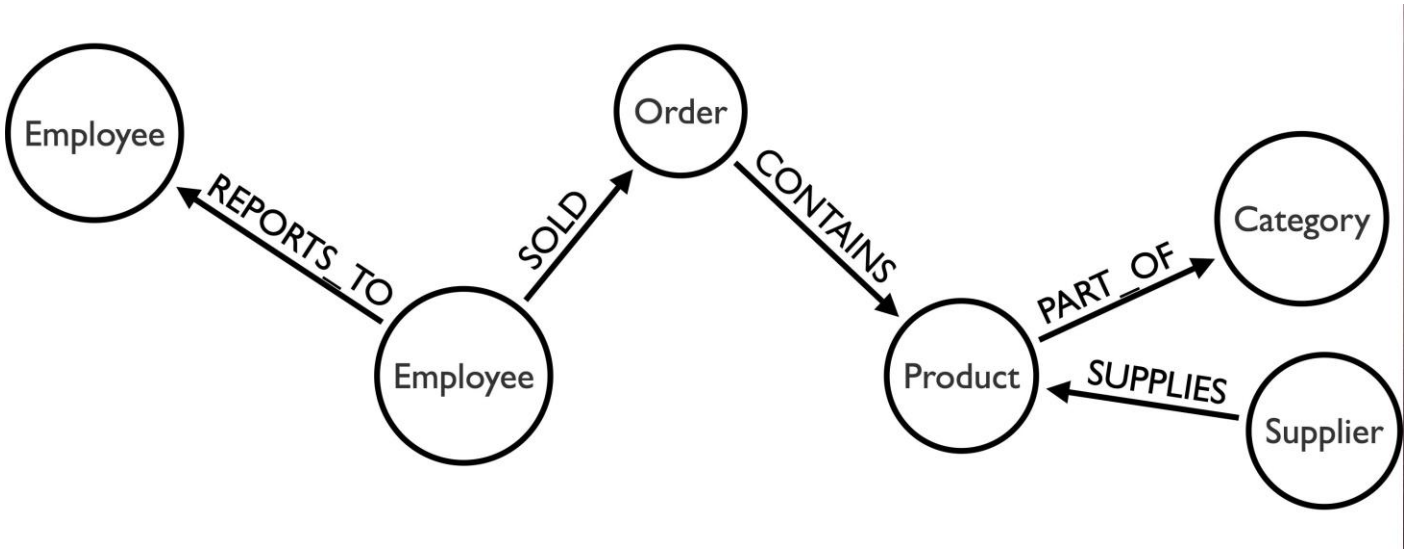
Filas a nodos, nombres de tablas a etiquetas

1. Cada fila de nuestra Orderstabla se convierte en un nodo en nuestro gráfico con Orderla etiqueta.
2. Cada fila de nuestra Productstabla se convierte en un nodo con Productla etiqueta.
3. Cada fila de nuestra Supplierstabla se convierte en un nodo con Supplierla etiqueta.
4. Cada fila de nuestra Categoriestabla se convierte en un nodo con Categoryla etiqueta.
5. Cada fila de nuestra Employeestabla se convierte en un nodo con Employeela etiqueta.

Se une a las relaciones

- Une entre Suppliersy se Productsconvierte en una relación nombrada SUPPLIES(donde el proveedor suministra el producto).
- Unir entre Productsy se Categoriesconvierte en una relación nombrada PART_OF(donde el producto es parte de una categoría).
- Unirse entre Employeesy se Ordersconvierte en una relación nombrada SOLD(donde el empleado vendió un pedido).
- La unión entre Employeessí (relación unaria) se convierte en una relación nombrada REPORTS_TO(donde los empleados tienen un gerente).
- Unir con tabla de unión (Order Details) entre Ordersy Productsse convierte en una relación llamada CONTAINScon propiedades de unitPrice, quantityy discount(donde pedido contiene un producto).

Si dibujamos nuestra traducción en la pizarra, tenemos este modelo de datos de gráfico.



¿En qué se diferencia el modelo gráfico del modelo relacional?

- No hay nulos.
- Describe las relaciones con más detalle.
- Cualquiera de los modelos se puede normalizar más

Exportación de tablas relacionales a CSV

Se deberá tomar los datos de las tablas relacionales y ponerlos en otro formato para cargarlos en el gráfico. Un formato común que muchos sistemas pueden manejar en un archivo plano de valores separados por comas (CSV), así que veamos cómo exportar tablas relacionales desde una base de datos PostgreSQL a archivos CSV para que podamos crear nuestro gráfico.

El comando 'copiar' de PostgreSQL nos permite ejecutar una consulta SQL y escribir el resultado en un archivo CSV. Podemos ensamblar un breve script .sql de estos comandos de copia, como se muestra a continuación.

export_csv.sql Sql

COPY (SELECT *FROM customers) TO '/tmp/customers.csv' WITH CSV header; COPY (SELECT* FROM suppliers) TO '/tmp/suppliers.csv' WITH CSV header; COPY (SELECT *FROM products) TO '/tmp/products.csv' WITH CSV header; COPY (SELECT* FROM employees) TO '/tmp/employees.csv' WITH CSV header; COPY (SELECT * FROM categories) TO '/tmp/categories.csv' WITH CSV header;

COPY (SELECT * FROM orders LEFT OUTER JOIN order_details ON order_details.OrderID = orders.OrderID) TO '/tmp/orders.csv' WITH CSV header;

Luego, podemos ejecutar ese script en nuestra base de datos Northwind con el comando psql -d northwind < export_csv.sql, y creará los archivos CSV individuales enumerados en nuestro script.

Importando los datos usando Cypher

Después de haber exportado nuestros datos desde PostgreSQL, usaremos el comando LOAD CSV de Cypher para transformar el contenido del archivo CSV en una estructura de gráfico. Primero, probablemente querremos colocar nuestros archivos CSV en un directorio de fácil acceso. Con Neo4j Desktop, podemos colocarlos en el directorio de importación de la base de datos local (instrucciones detalladas que se encuentran en nuestra guía de importación de escritorio). De esta manera, podemos usar el file:///prefijo en nuestras declaraciones Cypher para ubicar los archivos. También podemos colocar los archivos en otro directorio local o remoto (admite HTTPS, HTTP y FTP) y especificar la ruta completa en nuestras declaraciones Cypher. Dado que estamos

usando Neo4j Desktop en este ejemplo, usaremos la carpeta de importación para la base de datos y la ruta de nuestros archivos CSV puede comenzar con el file:///prefijo.

Ahora que tenemos nuestros archivos a los que podemos acceder fácilmente, podemos usar el LOAD CSVcomando de Cypher para leer cada archivo y agregar declaraciones Cypher después para tomar los datos de fila / columna y transformarlos en el gráfico.

El script Cypher completo está disponible en Github para que lo copie y ejecute, pero repasaremos cada sección a continuación para explicar qué hace cada pieza del script.

```
// Create orders LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row MERGE (order:Order {orderID: row.OrderID}) ON CREATE SET order.shipName = row.ShipName;

// Create products LOAD CSV WITH HEADERS FROM 'file:///products.csv' AS row MERGE (product:Product {productID: row.ProductID}) ON CREATE SET product.productName = row.ProductName, product.unitPrice = toFloat(row.UnitPrice);

// Create suppliers LOAD CSV WITH HEADERS FROM 'file:///suppliers.csv' AS row MERGE (supplier:Supplier {supplierID: row.SupplierID}) ON CREATE SET supplier.companyName = row.CompanyName;

// Create employees LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row MERGE (e:Employee {employeeID:row.EmployeeID}) ON CREATE SET e.firstName = row.FirstName, e.lastName = row.LastName, e.title = row.Title;

// Create categories LOAD CSV WITH HEADERS FROM 'file:///categories.csv' AS row MERGE (c:Category {categoryID: row.CategoryID}) ON CREATE SET c.categoryName = row.CategoryName, c.description = row.Description;
```

Es posible que observe que no hemos importado todas las columnas de campo en nuestro archivo CSV. Con nuestras declaraciones, podemos elegir qué propiedades se necesitan en un nodo, cuáles pueden omitirse y cuáles pueden necesitar ser importadas a otro tipo de nodo o relación. También puede notar que usamos la MERGEpalabra clave , en lugar de CREATE. Aunque estamos bastante seguros de que no hay duplicados en nuestros archivos CSV, podemos utilizarlos MERGEcomo una buena práctica para garantizar entidades únicas en nuestra base de datos.

Para conjuntos de datos comerciales o empresariales muy grandes, es posible que encuentre errores de memoria insuficiente, especialmente en máquinas más pequeñas. Para evitar estas situaciones, puede anteponer la instrucción con la USING PERIODIC COMMITsugerencia de consulta para confirmar los datos en lotes. Esta práctica no es una recomendación estándar para conjuntos de datos más pequeños, pero solo se recomienda cuando existen problemas de memoria.

Una vez creados los nodos, debemos crear las relaciones entre ellos. Importar las relaciones significará buscar los nodos que acabamos de crear y agregar una relación entre esas entidades existentes. Para asegurarnos de

que la búsqueda de nodos esté optimizada, querremos crear índices para cualquier propiedad de nodo que queramos usar en las búsquedas (a menudo, la identificación u otro valor único).

También queremos crear una restricción (también crea un índice con ella) que no permitirá que se creen pedidos con el mismo ID, evitando duplicados. Finalmente, como los índices se crean después de que se insertan los nodos, su población ocurre de forma asincrónica, por lo que usamos el schema await(un comando de shell) para bloquear hasta que se llenen.

Ejecutar en Neo4j

- CREATE INDEX product_id FOR (p:Product) ON (p.productID);
- CREATE INDEX product_name FOR (p:Product) ON (p.productName);
- CREATE INDEX supplier_id FOR (s:Supplier) ON (s.supplierID);
- CREATE INDEX employee_id FOR (e:Employee) ON (e.employeeID);
- CREATE INDEX category_id FOR (c:Category) ON (c.categoryID);
- CREATE CONSTRAINT order_id ON (o:Order) ASSERT o.orderID IS UNIQUE;
- schema await

Con los nodos e índices iniciales en su lugar, ahora podemos crear las relaciones de pedidos a productos y pedidos a empleados.

// Create relationships between orders and products LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row MATCH (order:Order {orderID: row.OrderID}) MATCH (product:Product {productID: row.ProductID}) MERGE (order)-[op:CONTAINS]->(product) ON CREATE SET op.unitPrice = toFloat(row.UnitPrice), op.quantity = toFloat(row.Quantity);

// Create relationships between orders and employees LOAD CSV WITH HEADERS FROM "file:///orders.csv" AS row MATCH (order:Order {orderID: row.OrderID}) MATCH (employee:Employee {employeeID: row.EmployeeID}) MERGE (employee)-[:SOLD]->(order);

A continuación, cree relaciones entre productos, proveedores y categorías:

// Create relationships between products and suppliers LOAD CSV WITH HEADERS FROM "file:///products.csv" AS row MATCH (product:Product {productID: row.ProductID}) MATCH (supplier:Supplier {supplierID: row.SupplierID}) MERGE (supplier)-[:SUPPLIES]->(product);

// Create relationships between products and categories LOAD CSV WITH HEADERS FROM "file:///products.csv" AS row MATCH (product:Product {productID: row.ProductID}) MATCH (category:Category {categoryID: row.CategoryID}) MERGE (product)-[:PART_OF]->(category);

Por último, crearemos la relación 'REPORTS_TO' entre empleados para representar la estructura de informes:

// Create relationships between employees (reporting hierarchy) LOAD CSV WITH HEADERS FROM "file:///employees.csv" AS row MATCH (employee:Employee {employeeID: row.EmployeeID}) MATCH (manager:Employee {employeeID: row.ReportsTo}) MERGE (employee)-[:REPORTS_TO]->(manager);

También puede ejecutar todo el script a la vez usando bin/neo4j-shell -path northwind.db -file import_csv.cypher.
Ahora podemos consultar el gráfico resultante para averiguar qué nos puede decir sobre nuestros datos recién importados.

Consultar el grafico

Podríamos comenzar con un par de consultas generales para verificar que nuestros datos coincidan con el modelo que diseñamos anteriormente en la guía. A continuación, se muestran algunos ejemplos de consultas:

//find a sample of employees who sold orders with their ordered products MATCH (e:Employee)-[rel:SOLD]->(o:Order)-[rel2:CONTAINS]->(p:Product) RETURN e, rel, o, rel2, p LIMIT 25;

//find the supplier and category for a specific product MATCH (s:Supplier)-[r1:SUPPLIES]->(p:Product {productName: 'Chocolade'})-[r2:PART_OF]->(c:Category) RETURN s, r1, p, r2, c;

Una vez que estemos seguros de que los datos se alinean con nuestro modelo de datos y todo parece correcto, podemos comenzar a realizar consultas para recopilar información y conocimientos para las decisiones comerciales. Una pregunta que podría interesarnos es la siguiente:

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;

Parece que el empleado nº 4 estaba ocupado, ¡aunque el empleado nº 1 también lo hizo bien!

empleado	otro producto	contar
4	Gnocchi di nonna Alice	14
4	Paté chinois	12
1	Flotemysost	12
3	Gumbär Gummibärchen	12
1	Pavlova	11

También nos gustaría responder a la siguiente pregunta:

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

gerente	empleado
2	3
2	4
2	5
2	1
2	8
5	9
5	7
5	6

Tenga en cuenta que el empleado No. 5 tiene personas que le reportan pero también reporta al empleado No. 2.

Investiguemos eso un poco más:

MATCH path = (e:Employee)<-[:REPORTS_TO*]-(sub) WITH e, sub, [person in NODES(path) | person.employeeID][1..-1] AS path RETURN e.employeeID AS manager, path as middleManager, sub.employeeID AS employee ORDER BY size(path);

gerente	Gerente de nivel medio	empleado
2	[]	3
2	[]	4
2	[]	5
2	[]	1
2	[]	8
5	[]	9
5	[]	7
5	[]	6
2	[5]	9
2	[5]	7
2	[5]	6

MATCH (e:Employee) OPTIONAL MATCH (e)-[:REPORTS_TO*0..]-(sub)-[:SOLD]- (-[:REPORTS_TO*0..]-(sub)-[:SOLD]-)(order) RETURN e.employeeID as employee, [x IN COLLECT(DISTINCT sub.employeeID) WHERE x <> e.employeeID] AS reportsTo, COUNT(distinct order) AS totalOrders ORDER BY totalOrders DESC;

empleado	informes a	totalOrders
2	[8,1,5,6,7,9,4,3]	830
5	[6,7,9]	224
4	[]	156
3	[]	127
1	[]	123
8	[]	104
7	[]	72
6	[]	67
9	[]	43

Bibliografia

https://neo4j.com/developer/guide-importing-data-and-etl/ (https://neo4j.com/developer/guide-importing-dataand-etl/)

In [ ]: