Como poder migrar desde una base de datos transaccional a BDOG (DE POSTGRES A neo4J)

Al derivar un modelo de gráfico a partir de un modelo relacional, debemos tener en cuenta un par de pautas generales.

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

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

- No hay nulos. Las entradas de valor no existentes (propiedades) simplemente no están presentes.
- Describe las relaciones con más detalle.
- Cualquiera de los modelos se puede normalizar más.

- Exportación de tablas relacionales a CSV

Una de las mas factibles es exportando nuestros datos de la base de datos relacional a archivos CSV.

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.

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;

- 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 MERGE palabra 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.

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 schema await(un comando de shell) para bloquear hasta que se llenen.

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);

- Consultar el gráfico

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;