udp FACULTAD DE
INGENIERÍA Y CIENCIAS

Bases de Datos Avanzadas: Actividad 5

Dante Hortuvia, Diego lara,
Profesor: Manuel Alba
Ayudante: Cesar Muñoz
Sección: 2

14 de junio 2024

Índice

1. Introducción **3**

2. Desarrollo **4**

2.1. Generación de Datos 4

2.1.1. Creación Clientes 5

2.1.2. Creación Productos 5

2.1.3. Creación Ventas 6

2.2. Inserción de Datos 6

2.3. Consultas 8

2.3.1. Consulta 1 8

2.3.2. Consulta 2 8

2.3.3. Consulta 3 9

3. Conclusión **11**

1. Introducción

El presente informe describe el proceso de generación, carga e inserción de datos en una base de datos Neo4j, así como las consultas realizadas sobre dichos datos. Con el objetivo de analizar la eficiencia y funcionalidad de diferentes métodos de carga, se implementaron dos enfoques principales: uno mediante un script de Python utilizando operaciones por lotes (batch) y otro a través de la importación de archivos CSV a la nube de Neo4j. Además, se crearon scripts específicos para la generación de clientes, productos y ventas, permitiendo una manipulación y análisis detallado de la información. El informe también incluye las consultas realizadas sobre la base de datos para extraer información relevante, como el cliente con mayor gasto, el producto más vendido y los clientes que compraron un producto específico.

2. Desarrollo

2.1. Generación de Datos

Para generar los datos se probaron dos métodos. El primero consistió en crear los datos y cargarlos mediante un script de Python utilizando operaciones por lotes (batch). El segundo método implicó generar un archivo CSV y subirlo a la nube de Neo4j.

```python
from neo4j import GraphDatabase
import time

uri = "neo4j+s://194c036a.databases.neo4j.io"
user = "neo4j"
password = "TiNgrUxHB9DnGCFH_DN_bSzG7TuabZUICT91iFs4rIA"

driver = GraphDatabase.driver(uri, auth=(user, password))

def close_driver(driver):
    driver.close()

def upload_batch(session, batch):
    query = """
    UNWIND $batch AS row
    CREATE (c:Clientes {rut: row.rut, name: row.name})
    """
    session.run(query, batch=batch)

def main():
    data = [{"rut": rut, "name": f"Cliente {rut}"} for rut in range(1, 5001)]

    batch_size = 500

    start_time = time.time()

    with driver.session() as session:
        for i in range(0, len(data), batch_size):
            batch = data[i:i + batch_size]
            upload_batch(session, batch)

    end_time = time.time()
    execution_time = end_time - start_time

    print(f"Tiempo de ejecución: {execution_time} segundos")

if __name__ == "__main__":
    main()
    close_driver(driver)
```

Figura 1: Script batch

El código carga datos en una base de datos Neo4j utilizando el controlador de Python neo4j. Primero, se genera una lista de 5000 registros de clientes, cada uno con un rut y un name. Estos registros se dividen en lotes de 500 y se suben a la base de datos mediante la función upload_batch, que utiliza la consulta UNWIND para crear nodos en la base de datos. La carga en lotes optimiza el proceso, reduciendo el tiempo de ejecución, el cual se mide y se imprime al final del proceso. Finalmente, se cierra la conexión a la base de datos.

```
PS C:\Users\Dante hortuvia\Desktop\Tarea_5> & "C:/Users/Dante hortuvia/AppData/Local/Microsoft/WindowsApps/python3.8.exe" "c:/Users/Dante hortuvia/Desktop/Tarea_5/Neo4j_Crear_CLientes.py"
Tiempo de ejecución: 3.7029855251312256 segundos
```

Figura 2: Tiempo obtenido Script batch

Este método se utilizó como prueba para entender el uso de batch en Neo4j. Aunque fue efectivo, resultó ineficiente en términos de tiempo al generar las ventas, lo que llevó a optar por la carga de datos mediante archivos CSV en Neo4j. Este enfoque resultó mucho más sencillo y rápido para la generación y subida de datos.

2.1.1. Creación Clientes

Para crear los clientes se utiliza un script de python que genera clientes. El Script genera nformación de 5000 clientes, cada uno con un ID único, un nombre y un RUT único de 9 dígitos. La función generar_ruts_unicos asegura la unicidad de los RUTs mediante la generación aleatoria y el uso de un conjunto. Este DataFrame facilita la manipulación y el análisis de los datos de los clientes.

```python
def generar_ruts_unicos(n):
    ruts = set()
    while len(ruts) < n:
        rut = str(random.randint(1000000000, 9999999999))
        ruts.add(rut)
    return list(ruts)

clientes = pd.DataFrame({
    'Cliente_ID': np.arange(1, 5001),
    'Nombre': [f'Cliente {i}' for i in range(1, 5001)],
    'RUT': generar_ruts_unicos(5000)
})
```

Figura 3: Script para crear csv de clientes

2.1.2. Creación Productos

Se utiliza un script que genera productos, cada producto tiene un ID único (del 1 al 50), un nombre en el formato 'Producto X' y un precio. Los precios se generan aleatoriamente utilizando np.random.uniform para obtener valores entre 10 y 100, redondeados a dos decimales.

```python
productos = pd.DataFrame({
    'Producto_ID': np.arange(1, 51),
    'Nombre': [f'Producto {i}' for i in range(1, 51)],
    'Precio': np.round(np.random.uniform(10, 100), 2)
})
```

Figura 4: Script para crear csv de producto

2.1.3. Creación Ventas

Se utiliza un script que genera las ventas, el script genera un DataFrame de Pandas con datos de ventas simuladas para 5000 clientes. Para cada cliente, asigna un rango de productos (10 por cliente) y genera una o más ventas con una fecha aleatoria en el último año. Cada venta incluye un ID de venta, ID de cliente, ID de producto, cantidad vendida (entre 1 y 10) y fecha de venta en formato 'YYYY-MM-DD'. Los datos de ventas se almacenan en una lista y se convierten en un DataFrame al final.

```python
ventas = []
for cliente_id in range(1, 5001):
    producto_inicio = ((cliente_id - 1) // 1000) * 10 + 1
    producto_fin = producto_inicio + 9
    for producto_id in range(producto_inicio, producto_fin + 1):
        fecha_venta = datetime.now() - timedelta(days=random.randint(0, 365))
        ventas.append({
            'Venta_ID': len(ventas) + 1,
            'Cliente_ID': cliente_id,
            'Producto_ID': producto_id,
            'Cantidad': random.randint(1, 10),
            'Fecha': fecha_venta.strftime('%Y-%m-%d')
        })
ventas = pd.DataFrame(ventas)
```

Figura 5: Script para crear csv de Ventas

2.2. Inserción de Datos

Para la insercion de datos se hace mediante Neo4j en import donde se suben los 3 csv, luego de establecer sus Primary key se hacen las conexiones entre los nodos mediante la siguentes consultas, para conectar ventas con produtos es

```
MATCH (v:Venta)
MATCH (p:Producto {Produto_ID: v.Producto_ID})
MERGE (v)-[:Ventas_productos]->(p)
```

Listing 1: Conexion de productos y ventas

y para conetar ventas con Clientes se utiliza la siguiente :

```
MATCH (v:Ventas)
MATCH (c:Clientes_ID {Clientes_ID: v.Clientes_ID})
MERGE (c)-[:Ventas_Clientes]->(v)
```

Listing 2: Conexion de Clientes y ventas

con esto se generaron las conexiones con el nodo ventas, la operacion total se demoro 16 segundos en subir los 3 csv.

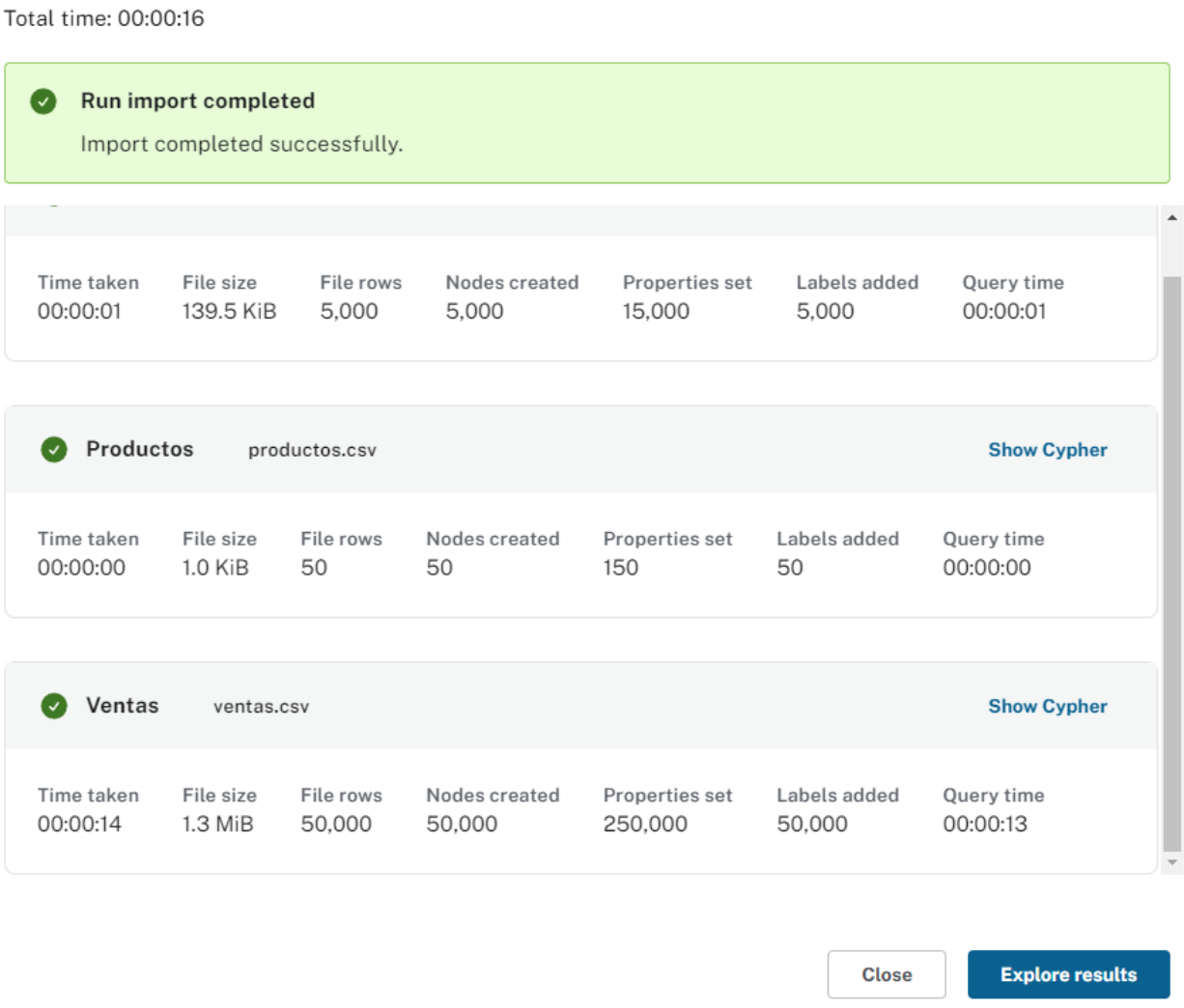

Figura 6: Tiempo de insercion de los csv

2.3. Consultas

2.3.1. Consulta 1

```
MATCH (c:Clientes)-[:Ventas_Clientes]->(v:Ventas)-[:Ventas_producto]
->(p:Productos)
WITH c, sum(v.Cantidad * p.Precio) as Gasto
WITH max(Gasto) as max
MATCH (c:Clientes)-[:Ventas_Clientes]->(v:Ventas)-[:Ventas_producto]
->(p:Productos)
WITH c, sum(v.Cantidad * p.Precio) as Gasto, max
WHERE max = Gasto
RETURN c.Nombre, Gasto , max
```

Listing 3: Nombre del mejor cliente y monto total comprado

Esta consulta encuentra al cliente que ha gastado más en total en sus compras. Primero, calcula el gasto total de cada cliente sumando la cantidad de productos comprados multiplicada por su precio. Luego, determina el gasto máximo entre todos los clientes y finalmente retorna el nombre del cliente cuyo gasto total coincide con este máximo, junto con el gasto total y el máximo gasto.

c.Nombre	Gasto	max
"Cliente2509"	5725.51	5725.51

Figura 7: Resultado de la Query 1

Started streaming 1 record after 2 ms and completed after 175 ms.

Figura 8: Tiempo de la Query 1

2.3.2. Consulta 2

```
MATCH (v:Ventas)-[rr:Ventas_producto]->(p:Productos)
WHERE date(v.Fecha) >= date("2024-01-01")
AND date(v.Fecha) <= date("2024-12-31")
WITH p, sum(v.Cantidad) as CantidadT
WITH max(CantidadT) as max
MATCH (v:Ventas)-[rr:Ventas_producto]->(p:Productos)
WHERE date(v.Fecha) >= date("2024-01-01")
```

```
AND date(v.Fecha) <= date("2024-12-31")
WITH p, sum(v.Cantidad) as CantidadT, max
WHERE max = CantidadT
RETURN p.Nombre, p.ProductoID, CantidadT;
```

Listing 4: Código y Nombre del producto con la mayor cantidad acumulada de ventas en un rango de tiempo variable

Esta consulta encuentra el producto más vendido en 2024. Primero, filtra las ventas de 2024 y suma la cantidad vendida de cada producto. Luego, determina la cantidad máxima vendida entre todos los productos. Finalmente, retorna el nombre y el ID del producto cuya cantidad vendida coincide con este máximo, junto con la cantidad total vendida.

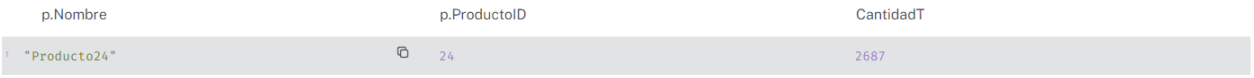

Figura 9: Resultado de la Query 2

Figura 10: Tiempo de la Query 2

2.3.3. Consulta 3

```
MATCH (c:Clientes)-[r:Ventas_Clientes]->(v:Ventas)-[rr:Ventas_producto]
->(p:Productos)
WHERE p.Nombre = "Producto2"
RETURN c.Nombre,c.RUT,p.Nombre;
```

Listing 5: Nombre y Rut de todos los clientes que han comprado un producto específico

Esta consulta encuentra todos los clientes que han comprado "Producto2". Primero, busca las relaciones entre clientes, ventas y productos. Luego, filtra para seleccionar solo las ventas del producto llamado "Producto2". Finalmente, retorna el nombre y RUT del cliente junto con el nombre del producto.

	c.Nombre	c.RUT	p.Nombre
1	"Cliente1"	5373966394	"Producto2"
2	"Cliente2"	7162064026	"Producto2"
3	"Cliente3"	7027693704	"Producto2"
4	"Cliente4"	7328115076	"Producto2"
5	"Cliente5"	9279714917	"Producto2"
6	"Cliente6"	9201299226	"Producto2"
7	"Cliente7"	6193892966	"Producto2"
8	"Cliente8"	1441105283	"Producto2"
9	"Cliente9"	4035437259	"Producto2"

Figura 11: Resultado de la Query 3

Figura 12: Tiempo de la Query 3

Nodo	Tiempo De ejecucion
Clientes	1 segundo
Productos	>1 segundo
Ventas	13 segundos

Resultados del tiempo al crear los nodos en Neo4j.

Consulta	Tiempo De ejecucion
Consulta 1	175 ms
Productos	166 ms
Ventas	68 ms

Resultados del tiempo al ejecutar las querys.

Como podemos observar, los tiempos de ejecución son bajos, lo que indica que Neo4j es una base de datos muy eficiente para localizar información durante la ejecución de consultas. Esto la convierte en una opción altamente eficaz para almacenar y recuperar datos rápidamente.

3. Conclusión

En conclusión, la comparación entre los métodos de carga de datos demostró que la utilización de archivos CSV en la nube de Neo4j es más eficiente y sencilla en comparación con el enfoque batch mediante un script de Python. La importación de archivos CSV no solo optimizó el tiempo de ejecución, sino que también simplificó el proceso de carga de grandes volúmenes de datos. Las consultas realizadas sobre la base de datos Neo4j permitieron obtener información valiosa sobre los clientes y productos, destacando la capacidad de Neo4j para manejar y analizar grandes conjuntos de datos de manera eficiente. Este análisis reafirma la eficacia de Neo4j como una herramienta poderosa para la gestión y consulta de bases de datos orientadas a grafos.