GESTIÓN/ALMACENAMIENTO DE INFORMACIÓN NO ESTRUCTURADA: BASES DE DATOS ORIENTADAS A GRAFOS

Sergio Soler Rocha
Universidad Nacional de Educación a Distancia

1. Despliegue de Neo4j usando Docker en Windows

Para la implementación de la base de datos Neo4j, se utilizó Docker en Windows como entorno de despliegue. Los pasos seguidos fueron los siguientes:

1. **Instalación de Docker Desktop:** Se descargó e instaló Docker Desktop para Windows desde el sitio oficial de Docker.
2. **Configuración de Docker:** Se habilitó la opción de usar el motor basado en WSL 2 en las configuraciones de Docker Desktop.
3. **Ejecución del contenedor Neo4j:** Se utilizó el siguiente comando en la terminal para iniciar Neo4j:

   ```
   docker run --name neo4j -p 7474:7474 -p 7687:7687 -v %USERPROFILE%/neo4j/data:/data neo4j
   ```

   Este comando configura el contenedor Neo4j, haciendo el servicio accesible a través de `http://localhost:7474` y almacenando los datos de forma persistente.

2. Creación de la Base de Datos

Utilizando los datos del archivo `grafo.txt`, se siguieron los siguientes pasos para crear y manipular la base de datos:

1. **Carga de Datos:** Se cargaron los datos en la base de datos Neo4j mediante la ejecución de los comandos Cypher provistos en el archivo `grafo.txt`.
2. **Consulta de Datos:** Se realizaron consultas específicas para verificar la integridad de los datos y su correcta manipulación. Por ejemplo, se visualizó algunos nodos con la siguiente sentencia:

   ```
   MATCH (n) RETURN n LIMIT 10;
   ```

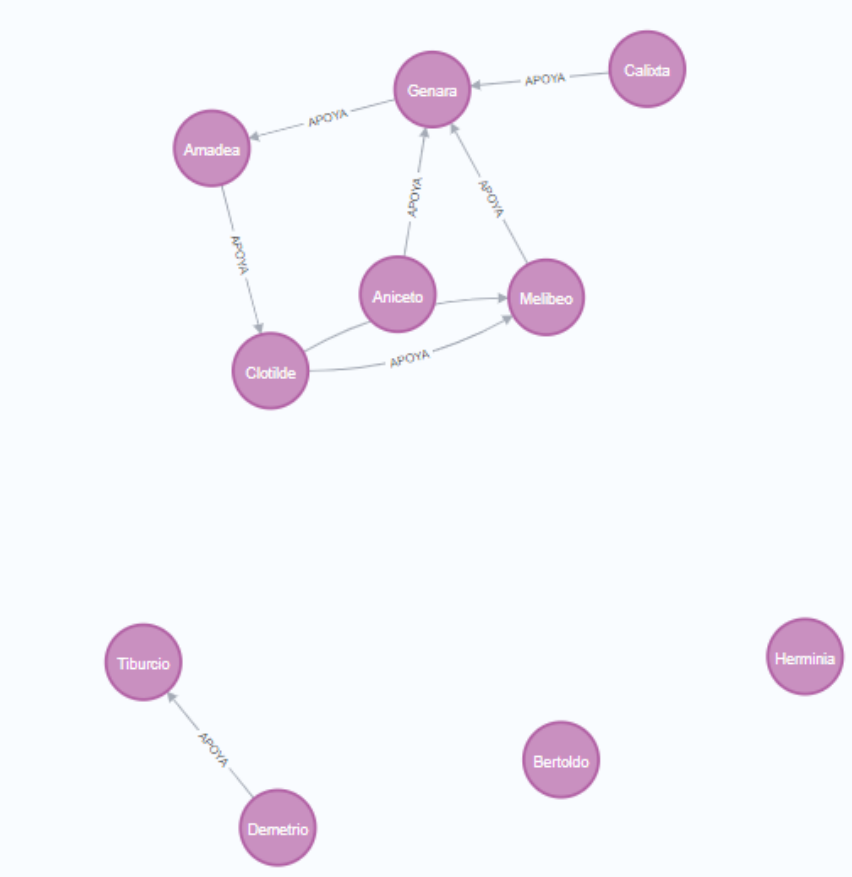

Figura 1: Salida de la consulta MATCH (n) RETURN n LIMIT 10;

3. Consultas en la Base de Datos Neo4j

En esta sección, se describen las consultas implementadas para obtener información específica de la base de datos utilizando el lenguaje de consulta Cypher de Neo4j.

3.1. Consulta de Opiniones Positivas

La primera consulta busca identificar los nombres de los clientes que han emitido opiniones positivas sobre un producto específico cuyo precio es de 4000 euros.

```
MATCH (c:Cliente)-[:GOOD]->(p:Producto {precio: 4000})
RETURN c.nombre AS NombreCliente
```

```
MATCH (c:Cliente)-[:GOOD]→(p:Producto {precio: 4000}) RETURN c.nombre AS NombreCliente
```

NombreCliente
"Aniceto"
"Calixta"
"Tiburcio"
"Demetrio"

Figura 2: Salida de la consulta 3.1

Esta consulta utiliza el patrón de coincidencia `MATCH` para encontrar relaciones de tipo `GOOD` entre clientes y el producto de precio 4000, retornando los nombres de los clientes correspondientes.

3.2. Búsqueda de Circuitos de Apoyo

La segunda consulta identifica clientes que forman un circuito de relaciones de apoyo, es decir, secuencias que comienzan y terminan en el mismo cliente.

```
MATCH cycle=(c:Cliente)-[:APOYA*]->(c2:Cliente)-[:APOYA*]->(c)
RETURN DISTINCT c.nombre AS NombreCliente
```

```
MATCH cycle=(c:Cliente)-[:APOYA*]→(c2:Cliente)-[:APOYA*]→(c) RETURN DISTINCT c.nombre AS NombreCliente
```

NombreCliente
"Melibeo"
"Genara"
"Amadea"
"Clotilde"

Figura 3: Salida de la consulta 3.2

Utilizando una búsqueda de ciclo con el operador `APOYA*`, esta consulta devuelve nombres de clientes que participan en al menos un ciclo de apoyo.

3.3. Adición de Relación de Apoyo

La tercera consulta asegura que Demetrio muestre su apoyo a Genara, creando nodos si no existen y estableciendo una relación única de apoyo entre ellos.

```
MERGE (demetrio:Cliente {nombre: 'Demetrio'})
MERGE (genara:Cliente {nombre: 'Genara'})
MERGE (demetrio)-[r:APOYA]->(genara)
RETURN demetrio, genara, r
```

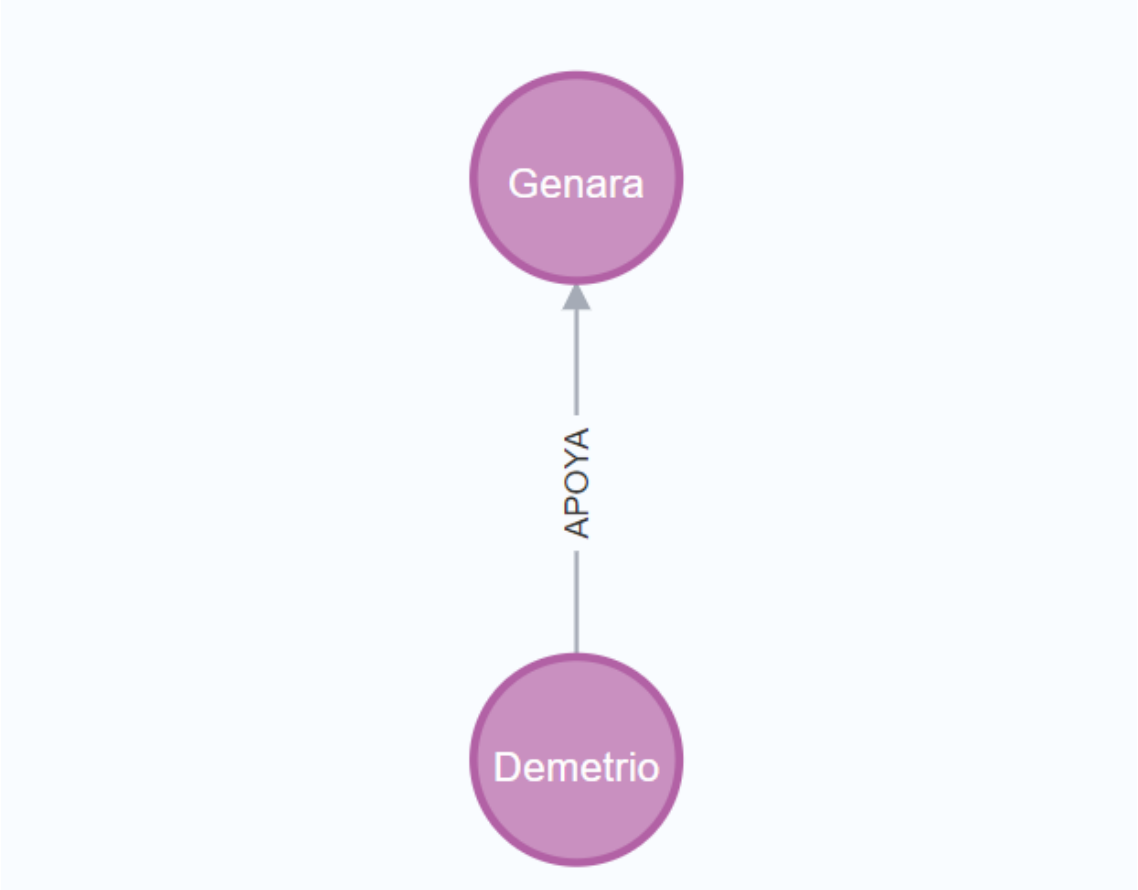

Figura 4: Salida de la consulta 3.3

La consulta usa el comando `MERGE` para evitar duplicaciones tanto de nodos como de relaciones, garantizando la existencia de una única relación de `APOYA` entre Demetrio y Genara.

4. Implementación de Base de Datos en Neo4j

Para explorar las capacidades de Neo4j en la gestión de redes complejas, se diseñó una base de datos que modela una red social de música. Esta base de datos incluye entidades como Usuarios, Artistas, Canciones, y Eventos, interconectadas a través de diversas relaciones que reflejan las interacciones en la red.

4.1. Estructura de la Base de Datos

La base de datos se compone de los siguientes nodos y relaciones:

- **Usuarios**: Fans de la música que pueden seguir artistas, asistir a eventos, y recomendar canciones.
- **Artistas**: Creadores de música que pueden tener seguidores y cuyas canciones son interpretadas en eventos.
- **Canciones**: Obras musicales interpretadas por artistas.
- **Eventos**: Eventos musicales donde se presentan diversas canciones.

Para facilitar la ejecución y prueba de diversas consultas en nuestro entorno Neo4j, se creó un archivo denominado 'bd_musica.txt'. Este archivo contiene el código necesario para la construcción de la base de datos de música, incluyendo la definición de nodos y relaciones entre ellos.

4.2. Consultas Implementadas

4.2.1. Consulta 1: Usuarios con Gustos Musicales Similares

Esta consulta identifica pares de usuarios que siguen a los mismos artistas, lo que podría indicar que tienen gustos musicales similares. Esta información es útil para funciones de recomendación de amigos o para agrupar usuarios en eventos. La consulta utiliza el patrón MATCH para encontrar pares de usuarios (u1 y u2) que siguen al menos a un artista en común. Utiliza la cláusula WHERE para asegurarse de que no se comparen los usuarios consigo mismos (u1 <> u2). Los artistas comunes se recolectan y se devuelven junto con los nombres de los usuarios, ordenando los resultados por el número de artistas comunes en orden descendente.

```
MATCH (u1:Usuario)-[:SIGUE]->(a:Artista)<-[:SIGUE]-(u2:Usuario)
WHERE u1 <> u2
RETURN u1.nombre, u2.nombre, collect(a.nombre) AS ArtistasComunes
ORDER BY size(ArtistasComunes) DESC
LIMIT 10;
```

	u1.nombre	u2.nombre	ArtistasComunes
1	"Alice"	"Emma"	["Daft Punk"]
2	"Emma"	"Dave"	["Daft Punk"]
3	"Alice"	"Dave"	["Daft Punk"]
4	"Emma"	"Alice"	["Daft Punk"]
5	"Dave"	"Alice"	["Daft Punk"]
6	"Frank"	"Bob"	["Taylor Swift"]
7			

Figura 5: Salida de la consulta 1

4.2.2. Consulta 2: Artistas más Populares por Seguidores

Esta consulta calcula y lista a los artistas según el número de usuarios que los siguen, lo que ayuda a identificar a los artistas más populares en la red. Se realiza un MATCH para conectar usuarios con artistas que siguen, y se utiliza la función count() para contar el número total de seguidores por artista. Los resultados se ordenan en orden descendente para resaltar a los artistas más populares.

```
MATCH (u:Usuario)-[:SIGUE]->(a:Artista)
RETURN a.nombre, count(u) AS NumeroDeSeguidores
ORDER BY NumeroDeSeguidores DESC
LIMIT 10;
```

	a.nombre	NumeroDeSeguidores
1	"Daft Punk"	3
2	"Taylor Swift"	3
3	"Billie Eilish"	2

Figura 6: Salida de la consulta 2

4.2.3. Consulta 3: Eventos Recomendados para un Usuario Basado en sus Gustos

Esta consulta proporciona recomendaciones personalizadas de eventos a un usuario basadas en los artistas que sigue. Se busca a un usuario específico y se identifican los eventos donde actúan los artistas que el usuario sigue. La consulta asegura que se devuelvan eventos únicos y se listan los artistas que participan en cada evento, lo que es útil para personalizar la experiencia del usuario en la plataforma.

```
MATCH (u:Usuario {nombre: 'Alice'})-[:SIGUE]->(a:Artista),
(a)-[:INTERPRETA]->(s:Cancion)-[:FEATURED]->(e:Evento)
RETURN DISTINCT e.nombre, e.fecha, collect(a.nombre) AS Artistas
ORDER BY e.fecha;
```

	e.nombre	e.fecha	Artistas
1	"Festival de Música"	"2021-08-21"	["Daft Punk"]

Figura 7: Salida de la consulta 3

5. Conclusión

Este proyecto ha sido una excelente oportunidad para explorar el potencial de las bases de datos de grafos para analizar y gestionar datos complejos en un contexto social y musical. Nos ha permitido ver de primera mano cómo las tecnologías de bases de datos avanzadas pueden transformar nuestra capacidad para entender y mejorar las interacciones entre usuarios y contenidos. Con cada consulta y cada nodo creado, hemos ampliado nuestra comprensión de las posibles aplicaciones prácticas de Neo4j.