UNIVERSIDAD VERACRUZANA

Facultad de Estadística e Informática

MAESTRÍA EN SISTEMAS INTERACTIVOS CENTRADOS EN EL USUARIO

Nombre: Diana Rocha Botello

Asignatura: Gestión de Datos

Proyecto: Neo4j

Fecha: 17/12/2018

Contenido

1. Introducción

En la actualidad se demandan cada vez más recursos tecnológicos para cubrir las necesidades de los sistemas actuales, por lo que han surgido evoluciones en los sistemas de software y por consecuencia en las bases de datos; las cuales según (Gómez, 2007, p.18) son un conjunto de datos que pertenecen al mismo contexto, almacenados sistemáticamente para su posterior uso, es una colección de datos estructurados según un modelo que refleje las relaciones y restricciones existentes en el mundo real. Los datos que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y descripción han de ser únicas estando almacenadas junto a los mismos.

Para facilitar el manejo de una base de datos se desarrollaron los sistemas gestores de bases de datos (SGBD). Un SGBD según (Silberschatz, 2002, p.1) consiste en una colección de datos inter-relacionados y un conjunto de programas para acceder a dichos datos. El objetivo principal de un SGBD es brindar una forma de almacenar y recuperar la información de la base de datos de forma práctica y eficiente, es decir, de forma sencilla y haciendo obteniendo los resultados deseados con el mínimo de recursos. Los SGB cuentan con herramientas para crear la estructura, manipular los datos, recuperar la información, realizar copias de seguridad, entre otras; también debe contar con algunos lenguajes para indicar que se realicen acciones específicas como lo son:

- Lenguaje de definición de datos (DDL por sus siglas en inglés).
- Lenguaje de manipulación de datos (DML por sus siglas en inglés).
- Lenguaje de consulta de datos (DQL por sus siglas en inglés).
- Lenguaje de control de datos (DCL por sus siglas en inglés).

En 1970 Codd publicó un artículo en donde definió el modelo relacional y llegó a convertirse en el principal modelo de datos para las aplicaciones de procesamiento de datos, en la actualidad se han ido desplazando por otros tipos de bases de datos pero aún se siguen utilizando. Para el modelo relacional se deben de considerar los siguientes conceptos básicos:

- Tabla o relación
- Atributos. Son las columnas de las tablas.
- Tuplas. Son las filas de las tablas.
- Grado. Es la cantidad de columnas o atributos.
- Cardinalidad. Número de tuplas.
- Claves o llaves. Las claves pueden ser candidatas, primarias, alternativas o foráneas.

El modelo relacional tiene restricciones entre las cuales se pueden mencionar las siguientes:

- No pueden existir tuplas repetidas dentro de una misma tabla.
- Una tupla debe corresponder con un elemento del mundo real.
- Cada tabla debe tener un nombre único.

- Un atributo sólo puede tener un valor en la tupla.
- El orden de las tuplas y atributos no importa.

El modelo relacional utiliza principalmente SQL como lenguaje de consulta y permite el uso de transacciones con propiedades ACID, que son:

- Atomicidad. Se refiere a que las operaciones dentro de una transacción se ejecutan como una sola unidad.
- Consistencia. Se refiere a que se debe de preservar la consistencia de los datos.
- Aislamiento. Se refiere a que deben ser independientes entre sí.
- Durabilidad. Se refiere a que los cambios en los datos que hagan las transacciones deben verse reflejados.

Posteriormente surgieron bases de datos orientadas a objetos (OODBMS por sus siglas en inglés) para cubrir la necesidad de abordar tipos de datos complejos que las bases de datos existentes no permitían y utiliza el lenguaje de consulta de objetos (OQL por sus siglas en inglés), el cual es un lenguaje declarativo del tipo de SQL, que permite realizar consultas de modo eficiente sobre bases de datos orientadas a objetos. Está basado en SQL-92, el cual le proporciona un enorme conjunto de la sintaxis de la sentencia SELECT.

Algunas características del Sistema Gestor de Bases de Datos Orientado a objetos (SGBDOO) para este tipo de bases de datos son:

- Soporta objetos complejos.
- Identidad de objeto
- Encapsulamiento
- Soporta tipos o clases
- Los tipos o clases pueden heredar de superclases
- Permite la sobrecarga
- Lenguaje de manipulación de datos complejo
- Persistencia de datos
- Soporta el manejo de grandes cantidades de datos.
- Soporta concurrencia
- Permite la recuperación de datos
- Facilidad para hacer consultas

También surgieron las bases de datos objeto-relacional, las cuales extienden el modelo de datos relacional proporcionando un sistema de tipos enriquecido que incluye tipos colección y orientación a objetos; Oracle permite implementar este tipo de bases de datos.

Las necesidades siguieron creciendo y para resolver los problemas que limitaban a las bases de datos SQL, se comenzaron a implementar bases de datos NoSQL. Las bases de datos NoSQL cuentan con las siguientes características:

- Manejan grandes volúmenes de información.

- Dan mayor velocidad de respuesta.
- Su esquema es flexible.
- Las consultas son simples.
- Tienen habilidad de distribución.
- Su concurrencia es débil porque sus transacciones no implementan las propiedades ACID.
- Permite el acceso a bajo nivel.

Existen cuatro tipos de modelos de datos NoSQL:

1. Por clave-valor. Este modelo utiliza una clave para identificar un valor.
2. Orientados a columnas. Estos modelos guardan los datos por columna.
3. Orientados a documentos. Se apoya del modelo clave-valor y los documentos pueden ser en formato XML o JSON. XBase y Mongo son bases de datos orientadas a documentos.
4. Orientados a grafos.

En el presente trabajo se pretende abordar un ejemplo de base de datos NoSQL orientada a grafos llamada Neo4j. Una base de datos orientada a grafos es aquella que permite almacenar la información como nodos de un grafo y sus respectivas relaciones con otros nodos, permitiendo así aplicar la teoría de grafos para recorrer la base de datos; son muy útiles para guardar información en modelos con muchas relaciones como redes y conexiones sociales. Cada nodo consta de un grado que indica el número de aristas que tiene, a su vez un grafo puede ser dirigido o no dirigido, dependiendo de si las aristas tienen nodos origen y nodos destino. El uso de este tipo de bases de datos depende altamente de la lógica de negocio donde se encuentre involucrada la información a almacenar, ya que no puede aplicar en todos los escenarios, o tal vez no se podría aprovechar su potencial en unos u otros contextos. En el modelo de estructura de grafos se tienen dos tipos de elementos: los nodos y las relaciones, cada uno de ellos contienen sus propiedades tipo clave valor; asimismo las relaciones agregan una característica y es que pueden ser o no dirigidos.

2. Características de Neo4j

Las bases de datos orientadas a grafos (BDOG) ayudan a encontrar relaciones y dar sentido completo. Una de las más conocidas es Neo4j, un servicio implementado en Java. Su primera versión fue lanzada en febrero de 2010 y en estos momentos está bajo dos tipos de licencia: una licencia comercial por un lado y una Affero General Public License (AGPL) por otro. Su desarrolladora es la compañía Neo Technology, una startup sueca con sede en San Francisco.

Neo4j usa grafos para representar datos y las relaciones entre ellos. Un grafo se define como cualquier representación gráfica formada por vértices (se ilustran mediante círculos) y aristas (se muestran mediante líneas de intersección). Existen diversos tipos de grafos, entre los cuales se pueden mencionar los siguientes:

- **Grafos no dirigidos**. Los nodos y las relaciones son intercambiables, su relación se puede interpretar en cualquier sentido. Las relaciones de amistad en la red social Facebook, por ejemplo, son de este tipo.
- **Grafos dirigidos**. Los nodos y las relaciones no son bidireccionales por defecto. Las relaciones en Twitter son de este tipo. Un usuario puede seguir a determinados perfiles en esta red social sin que ellos le sigan a él.
- **Grafos con peso**. En este tipo de gráficas las relaciones entre nodos tienen algún tipo de valoración numérica. Eso permite luego hacer operaciones.
- **Grafos con etiquetas**. Estos grafos llevan incorporadas etiquetas que pueden definir los distintos vértices y también las relaciones entre ellos. En Facebook podríamos tener nodos definidos por términos como ‘amigo’ o ‘compañero de trabajo’ y la relaciones como ‘amigo de’ o ‘socio de’.
- **Grafos de propiedad**. Es un grafo con peso, con etiquetas y donde podemos asignar propiedades tanto a nodos como relaciones (por ejemplo, cuestiones como nombre, edad, país de residencia, nacimiento). Es el más complejo.

Como todos los tipos de bases de datos, existen ventajas y desventajas; a continuación se presenta una lista de las ventajas de utilizar bases de datos orientadas a grafos:

- Rendimiento. Las bases de datos orientadas a grafos tienen mejor rendimiento en comparación con las relacionales y algunas no relacionales. La clave reside en que aunque las consultas de datos aumenten exponencialmente, el rendimiento no desciende. Estas bases de datos optimizan mucho el proceso de consultas.
- Agilidad. Este tipo de bases de datos tienen agilidad en la gestión de datos, ejemplo de ello es que si se quisiera llevar al límite sus capacidades, tendríamos que superar un volumen total de 34.000 millones de nodos (datos), 34.000 millones de relaciones entre esos datos, 68.000 millones de propiedades y 32.000 tipos de relaciones.
- Flexibilidad y escalabilidad. Cuando los desarrolladores de una empresa trabajan con grandes datos, buscan flexibilidad y escalabilidad. Las bases de

datos orientadas a grafos aportan mucho en este sentido porque cuando aumentan las necesidades, las posibilidades de añadir más nodos y relaciones a un grafo ya existente son enormes.

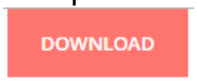
- Tolerante a fallos. Cuando una máquina o máquinas dentro de su clúster falle, se recupera con las máquinas que aún funcionen.

Por otro lado utilizar este tipo de bases de datos también tiene sus desventajas, entre las cuales se pueden mencionar las siguientes:

- Es eventualmente consistente. Cuando se implementa una base de datos distribuida y sólo exista un servidor maestro que la replique, los esclavos pueden aún no tener visibles las actualizaciones de manera inmediata.
- Modelo de datos no estandarizado.
- Lenguaje no estandarizado. Neo4j utiliza un lenguaje de consulta y manipulación de datos llamado Cypher, el cual no se encuentra estandarizado.

3. Descarga e instalación

Para instalar Neo4j es necesario ingresar a su página oficial https://neo4j.com. Una vez que se encuentra en su página da clic en el botón de la esquina superior derecha



y se mostrará la siguiente pantalla:

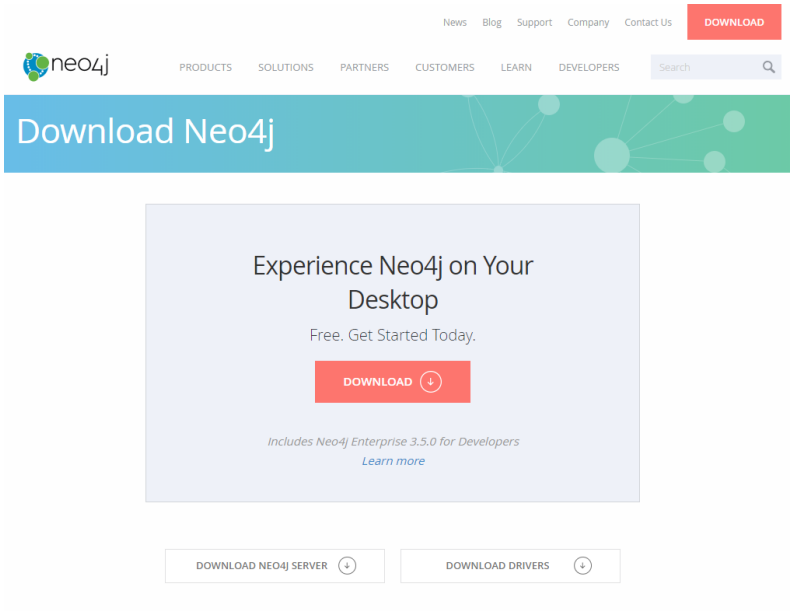


Figura 1. Página de descargas de Neo4j.

Posteriormente se da clic en el botón 

para conocer las versiones y los sistemas operativos para los que puede descargar Neo4j (Figura 2).

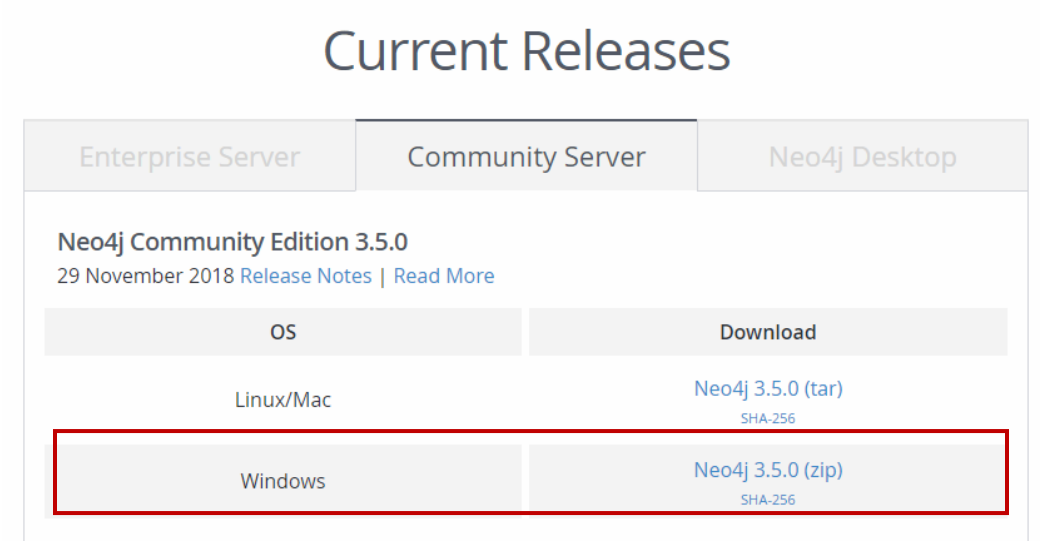


Figura 2. Opciones de descarga de Neo4j.

En este caso seleccionó Neo4j Community Server para Windows como se aprecia en la Figura 2.

Una vez descargado el archivo .zip, es necesario comprobar que nuestro equipo de cómputo tenga instalado Java y se haya indicado en las variables de entorno del sistema la variable JAVA_HOME. En mi caso se instaló:

- JDK-8u192-windows-x64
- JavaSetup8u191

Y se configuró la variable de entorno como se muestra en la Figura 3:

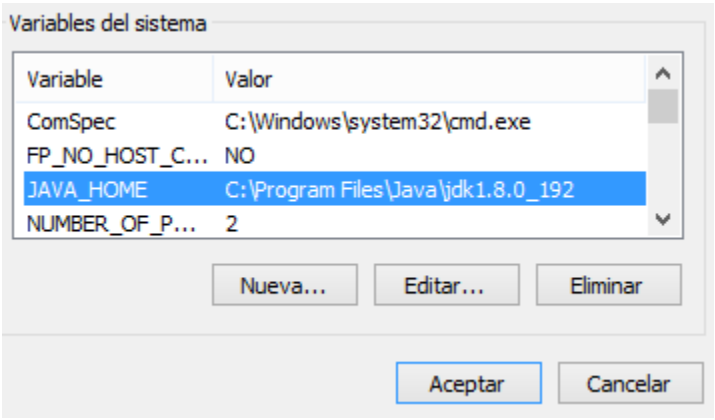


Figura 3. Configuración de las variables de entorno.

Después se procede a descomprimir el archivo .zip y se crea una variable Neo4j en C:, como se muestra en la Figura 4:

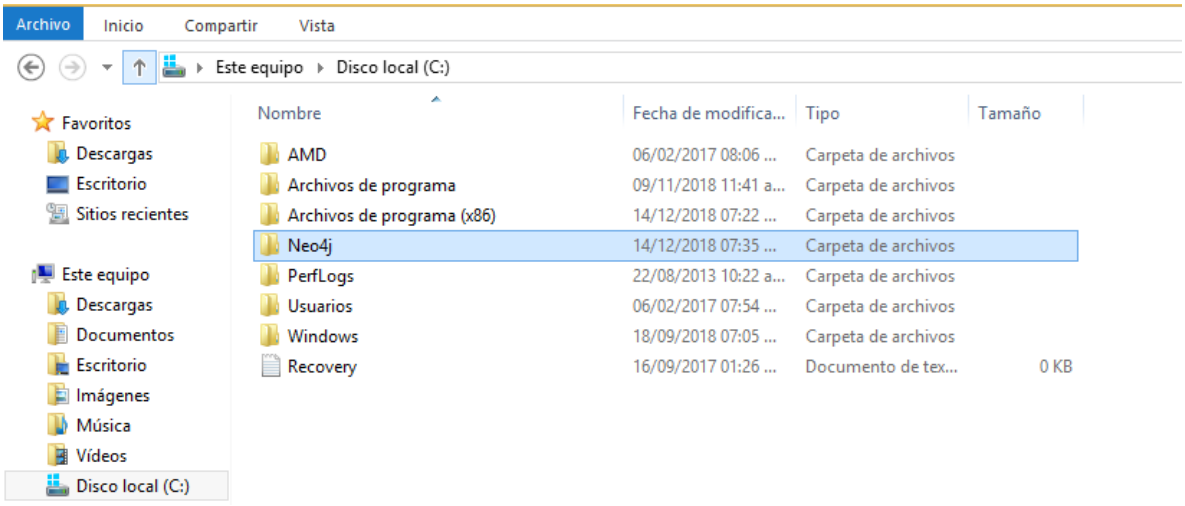


Figura 4. Creación de la carpeta neo4j.

Dentro de esa carpeta se copian los archivos que se descomprimieron del .zip (Figura 5).

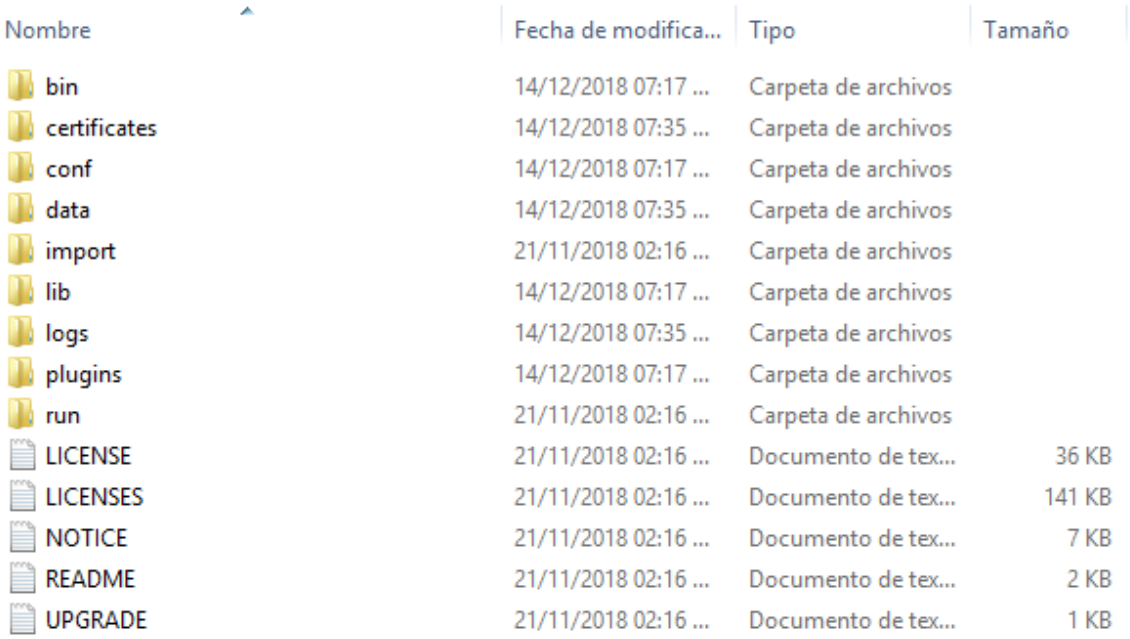


Figura 5. Copiar archivos a la carpeta de neo4j.

Posteriormente se abre una consola como administrador y se coloca en la carpeta de c:\Neo4j\bin como se muestra en la Figura 6:

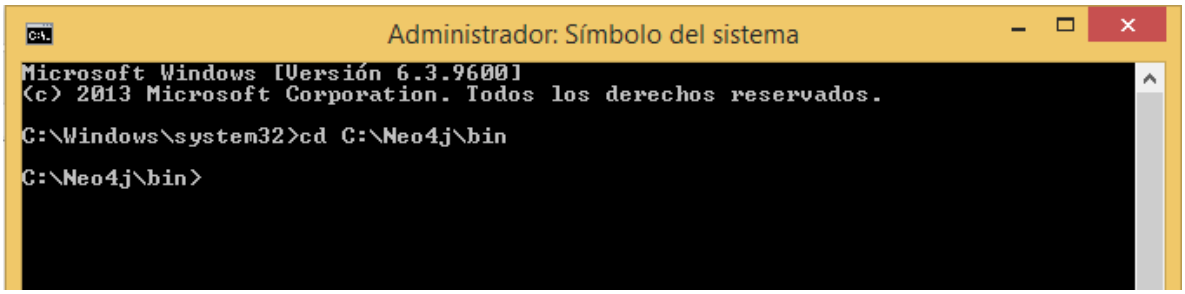


Figura 6. Ubicarse en la carpeta de neo4j.

Una vez ubicado en la ruta antes mencionada, debe ejecutar el siguiente comando en consola para instalar el servicio:

neo4j install-service

Para iniciar el servicio se ejecuta el comando *neo4j start*, en ocasiones el servicio no inicia a la primera por lo que se debe ejecutar el comando una vez más (Figura 7).

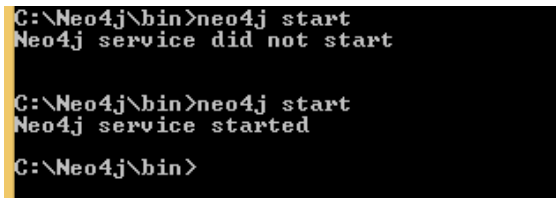


Figura 7. Iniciar el servicio.

Después de debe acceder mediante el navegador a la dirección http://localhost:7474/browser/ como se muestra en la Figura 8:

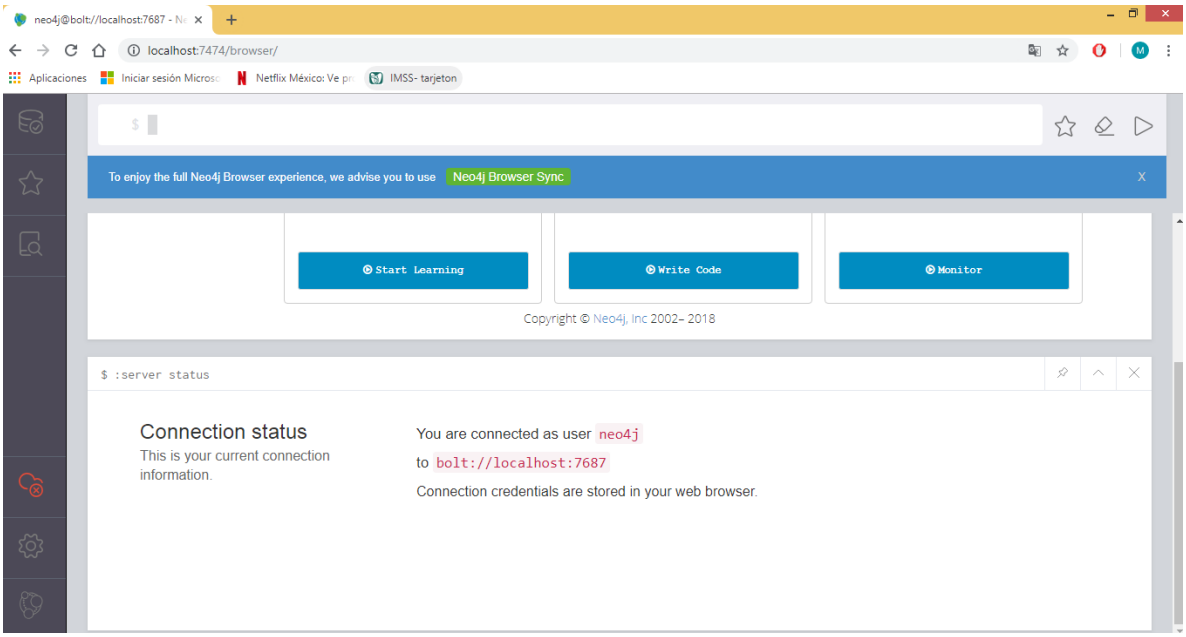


Figura 8. Acceso al servicio vía web.

Cuando se inicia por primera vez el servicio se debe ingresar el usuario “neo4j” y la contraseña “neo4j”, una vez ingresados te pide una contraseña nueva.

4. Definición de datos

En esta sección se describen ejemplos para crear nodos y relaciones, modificar nodos y relaciones y borrar nodos.

a. Crear nodos

Para crear nodos se utiliza la siguiente sentencia:
CREATE (VariableX: EtiquetaX {propiedad$_1$:valor$_1$, propiedad$_2$:valor$_2$,…, propiedad$_n$:valor$_n$})

En donde la VariableX se refiere a la categoría del nodo, EtiquetaX es el nombre de la categoría a la que pertenece el nodo, las propiedades son los atributos del nodo y el valor se refiere al valor que tomará el atributo del nodo.

En este caso se ejecutaron las siguientes sentencias para crear dos autores y cuatro libros:

CREATE(aut:autor{nombre:'Gabriel',apellido_paterno:'García',apellido_materno:'Márquez',pais:'Colombia'})

```
$ CREATE (aut:autor{nombre:'Gabriel',apellido_paterno:'García',apellido_materno:'Márquez',pais:'Colombia'})
```

Table

Added 1 label, created 1 node, set 4 properties, completed after 3120 ms.

CREATE(aut:autor{nombre:'Jhon',apellido_paterno:'Katzenbach',apellido_materno:'L',pais:'Estados Unidos'})

```
$ CREATE (aut:autor{nombre:'Jhon',apellido_paterno:'Katzenbach',apellido_materno:'L',pais:'Estados Unidos'})
```

Table

Added 1 label, created 1 node, set 4 properties, completed after 4 ms.

CREATE (lib:libro{codigo:'1',titulo:'El psicoanalista'})

```
$ CREATE (lib:libro{codigo:'1',titulo:'El psicoanalista'})
```

Table

Added 1 label, created 1 node, set 2 properties, completed after 144 ms.

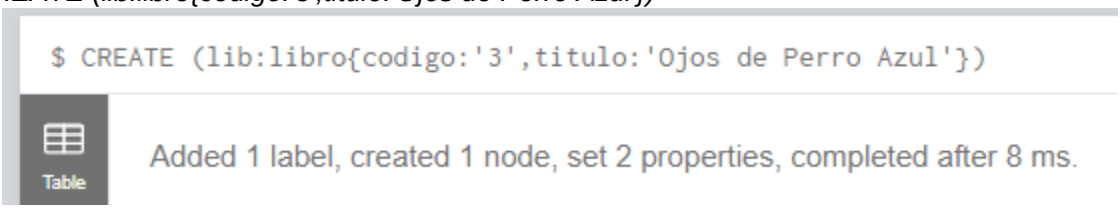
CREATE (lib:libro{codigo:'2',titulo:'Cien Años de Soledad'})

```
$ CREATE (lib:libro{codigo:'2',titulo:'Cien Años de Soledad'})
```

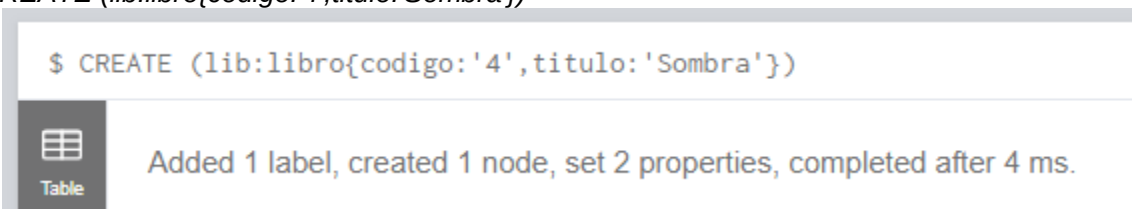
Table

Added 1 label, created 1 node, set 2 properties, completed after 8 ms.

CREATE (lib:libro{codigo:'3',titulo:'Ojos de Perro Azul'})



CREATE (lib:libro{codigo:'4',titulo:'Sombra'})



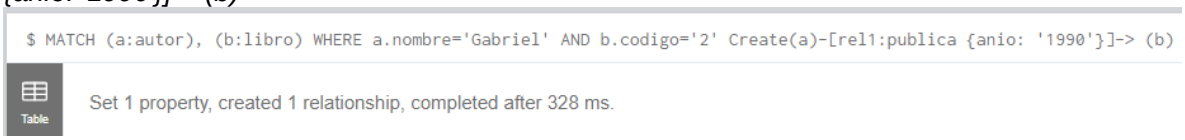
b. Crear relaciones

Para crear relaciones se utiliza la siguiente sentencia:
MATCH (x1: Etiqueta1), (x2: Etiqueta2) … CREATE (x1) - [VariableR: EtiquetaR { propiedad: valor }] -> (x2)

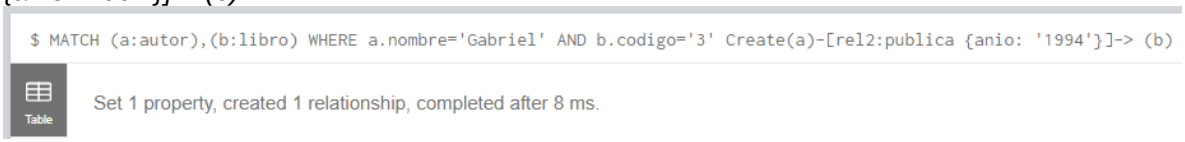
En donde x1 y x2 son dos variables cualquiera que se referirán a una etiqueta que exista (Etiqueta1 y Etiqueta2), la VariableR contendrá variables extras y a la relación en sí, la EtiquetaR será el grupo al cual pertenece la relación, la propiedad y valor son atributos y valores de la relación.

En este caso se ejecutaron las siguientes sentencias para crear relaciones entre los dos autores y los cuatro libros:

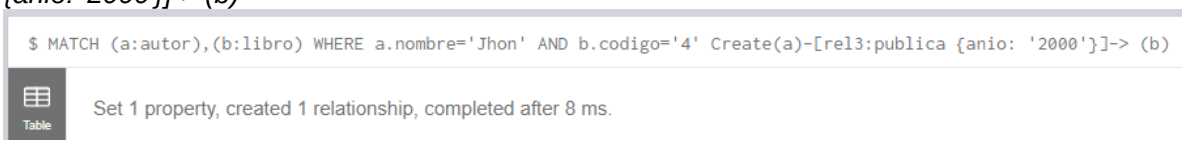
MATCH (a:autor), (b:libro) WHERE a.nombre='Gabriel' AND b.codigo='2' Create(a)-[rel1:publica {anio: '1990'}]-> (b)



MATCH (a:autor),(b:libro) WHERE a.nombre='Gabriel' AND b.codigo='3' Create(a)-[rel2:publica {anio: '1994'}]-> (b)



MATCH (a:autor),(b:libro) WHERE a.nombre='Jhon' AND b.codigo='4' Create(a)-[rel3:publica {anio: '2000'}]-> (b)



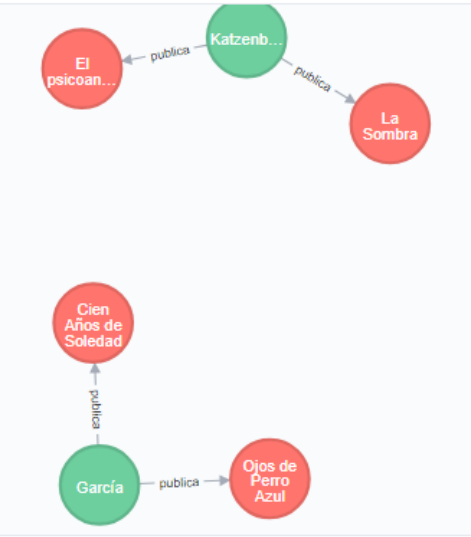
MATCH (a:autor),(b:libro) WHERE a.nombre='Jhon' AND b.codigo='1' Create(a)-[rel4:publica {anio: '1999'}]-> (b)

```
$ MATCH (a:autor),(b:libro) WHERE a.nombre='Jhon' AND b.codigo='1' Create(a)-[rel4:publica {anio: '1999'}]-> (b)
```

Table

Set 1 property, created 1 relationship, completed after 8 ms.

Finalmente las relaciones quedaron de la siguiente manera:



c. Modificar nodos

Para modificar valores de las propiedades de los nodos es necesario definir una sentencia que siga la sintaxis que se presenta a continuación:
MATCH (VariableX: EtiquetaX {propiedad: valor}) SET VariableX.propiedad = 'Valor'

En donde la VariableX se refiere a la variable que contendrá la categoría del nodo, la EtiquetaX es la categoría del nodo, la propiedad y valor son el atributo y valor de atributo por el cual se seleccionará el nodo y finalmente se selecciona la propiedad del nodo que se desea cambiar, se coloca un signo de igual (‘=’) y se escribe el nuevo valor para la propiedad seleccionada.

En este caso se modificó el título de un libro y se utilizó la siguiente sentencia:

MATCH (lib:libro{codigo:'4'}) SET lib.titulo='La Sombra'

```
$ MATCH (lib:libro{codigo:'4'}) SET lib.titulo='La Sombra'
```

Table

Set 1 property, completed after 28 ms.

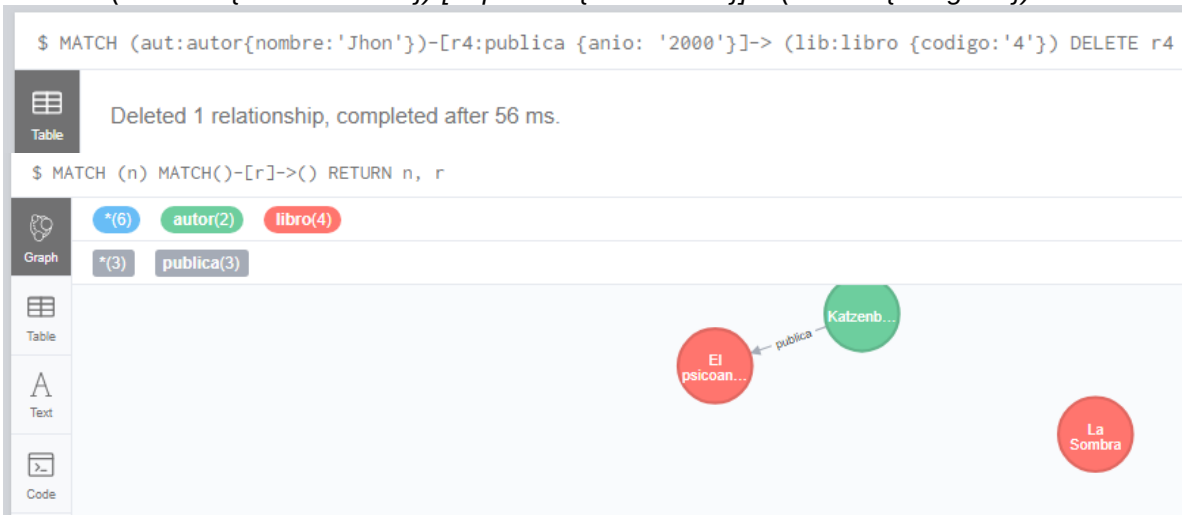
d. Borrar nodos

Para eliminar un nodo se necesita primero eliminar las dependencias que existan hacia él, en caso de existir alguna, y posteriormente se podrá eliminar el nodo sin problemas. Para eliminar la relación entre nodos y eliminar un nodo se utilizan las siguientes sentencias:
MATCH (Variable1: Etiqueta1 { propiedad: valor }) - [VariableR: EtiquetaR { propiedad: valor }] -> (Variable2: Etiqueta2 { propiedad: valor }) DELETE VariableR

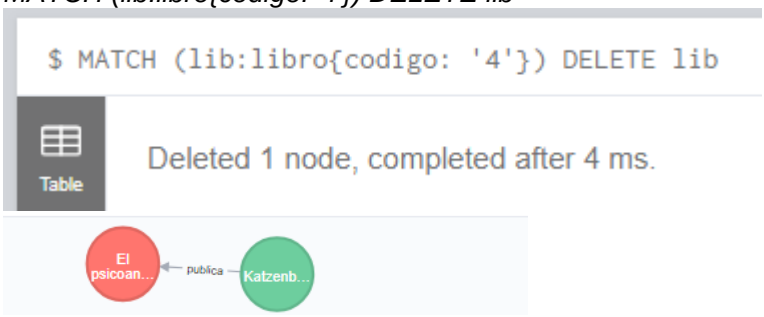
MATCH (VariableX: EtiquetaX { propiedad: valor }) DELETE VariableX

Las sentencias utilizadas para eliminar un nodo libro se muestran a continuación:

MATCH (aut:autor{nombre:'Jhon'})-[r4:publica {anio: '2000'}]-> (lib:libro {codigo:'4'}) DELETE r4



MATCH (lib:libro{codigo: '4'}) DELETE lib

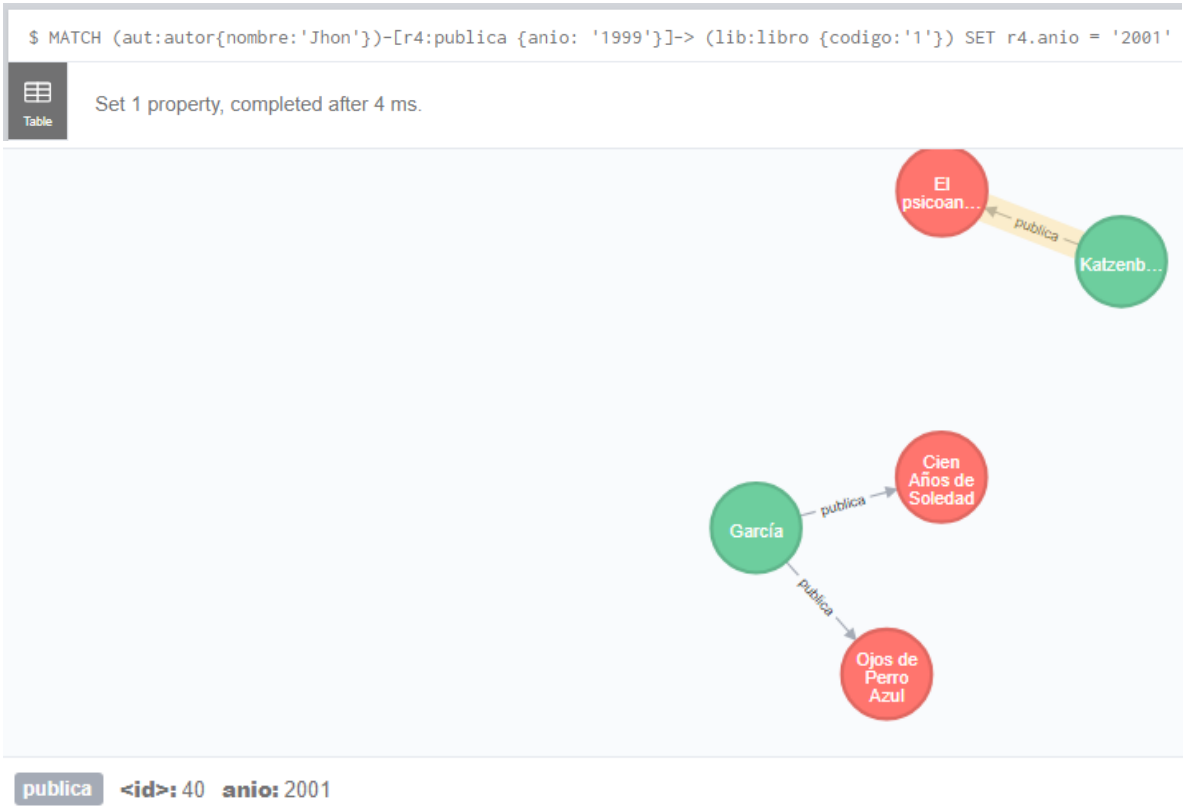


e. Modificar relaciones

La sentencia para modificar algún valor de las propiedades de la relación es similar al de modificar nodos y la sintaxis es como se presenta a continuación:
MATCH (Variable1: Etiqueta1 { propiedad: valor }) - [VariableR: EtiquetaR { propiedad: valor }] -> (Variable2: Etiqueta2 { propiedad: valor }) SET VariableR.propiedad = 'Valor'

Para comprender mejor la sintaxis anterior se modificó el valor de una propiedad de la relación entre un autor y un libro. La sentencia utilizada para realizarlo fue la siguiente:

MATCH (aut:autor{nombre:'Jhon'})-[r4:publica {anio: '1999'}]-> (lib:libro {codigo:'1'}) SET r4.anio = '2001'



5. Consultas

a. Consultas con match

Para realizar consultas con match se tienen dos sentencias básicas que son:
MATCH (x1: Etiqueta1 { propiedad: valor }), (x2: Etiqueta2 { propiedad: valor }) RETURN x1, x2
MATCH (x1: Etiqueta1 { propiedad: valor }) --> (x2: Etiqueta2 { propiedad: valor }) RETURN x1, x2

Para comprender mejor como funcionan estas sentencias se presentan dos ejemplos:

MATCH (a:autor{nombre:'Jhon'}), (b:libro {codigo:'1'}) RETURN a,b

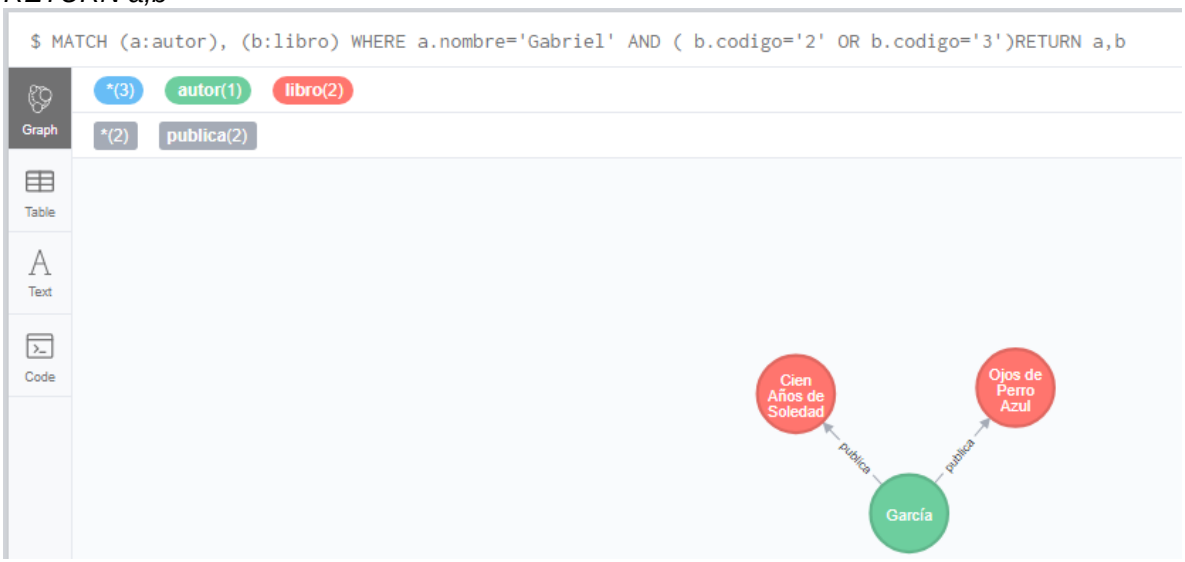
MATCH (a:autor{nombre:'Gabriel'})--> (b:libro) RETURN a,b

b. Consultas con where

Neo4j también cuenta con la sentencia Where para completar las consultas con match y brinda la posibilidad de utilizar operadores lógicos como en las consultas Where en SQL (AND, OR). Para realizar una consulta con where se sigue la sintaxis que se presenta a continuación:
MATCH (x1: Etiqueta1 { propiedad: valor }), (x2: Etiqueta2 { propiedad: valor }) WHERE x1.propiedad = 'valor' OperadorLogico x2.propiedad = 'valor' RETURN x1, x2

Para comprender mejor como funciona esta sentencia se presenta un ejemplo con nuestros nodos de autores y libros:

MATCH (a:autor), (b:libro) WHERE a.nombre='Gabriel' AND (b.codigo='2' OR b.codigo='3') RETURN a,b



c. Funciones de agregación

Neo4j cuenta con algunas funciones de agregación dentro de las cuales se pueden destacar las siguientes:

- avg(): Obtiene el promedio de un conjunto de valores numéricos.
  MATCH (v: etiqueta) RETURN avg(v.propiedad)
- collect(): Convierte en un arreglo de elementos un grupo específico.
  MATCH (v: etiqueta) RETURN collect(v.propiedad)
- count(): Cuenta el número de elementos de un grupo.
  MATCH (v: etiqueta) RETURN v.propiedad, count(*)
- max(): Obtiene el número mayor de un grupo.
  MATCH (v: etiqueta) RETURN max(v.propiedad)
- min(): Obtiene el número menor de un grupo.
  MATCH (v: etiqueta) RETURN min(v.propiedad)
- stDev(): Obtiene la desviación estándar de una muestra de valores, también existe stDevP() para obtener la desviación estándar de una población de valores.
  MATCH (v: etiqueta) RETURN stDev(v.propiedad)
- Sum(): Suma un grupo de valores numéricos.
  MATCH (v: etiqueta) RETURN sum(v.propiedad)

6. Conexión a la base de datos desde un lenguaje de programación

Para conectarse desde un lenguaje de programación a la base de datos, se utilizó Java con el IDE Netbeans para hacer un pequeño ejemplo de la conexión mediante el driver de neo4j que es compatible con Java a la base de datos de libros y autores que se ha definido durante el desarrollo de este documento.

En primer lugar se creó una aplicación de Java y una clase llamada *conexión_neo4j.java* (*Véase Anexo a*) que se muestra en la Figura 9:

```
public class conexion_neo4j {
    Connection conectar=null;
    public Connection conexion(){
        Connection conectar=null;
        try {
            // EN ESTA SECCIÓN SE CARGA EL DRIVER PARA HACER LA CONEXIÓN
            Class.forName("org.neo4j.jdbc.Driver");
            conectar = (Connection) DriverManager.getConnection("jdbc:neo4j:bolt://localhost/?user=neo4j,password=123456,scheme=basic");
            System.out.println("CONEXIÓN EXITOSA CON NEO4J PUBLICACIONES");
                // MENSAJE QUE INDICA QUE LA CONEXIÓN SE ESTABLECIÓ CORRECTAMENTE
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("CONEXIÓN EXITOSA CON NEO4J");
            JOptionPane.showMessageDialog(null, "FALLÓ LA CONEXIÓN CON NEO4J PUBLICACIONES"+e);
        }
        return conectar;

    }
}
```

Figura 9. Conexión a la base de datos con Java.

Este ejemplo sólo contempla la creación nodos de la categoría libros, para lo cual se generó una clase llamada *AgregarLibro.java* (*Véase Anexo b*) y su función principal es la de agregar nodos de libros como se puede apreciar en la Figura 10.

```
    public void nuevoNodo(){
        String codigo,titulo;
        String newNodo;

        codigo = id.getText();
        titulo = nom.getText();

        newNodo = "CREATE(lib:libro{codigo:'"+codigo+"', titulo:'"+titulo+"'})";



        try {
            PreparedStatement pst=BD.prepareStatement(newNodo);
            int n=pst.executeUpdate();
            if (n>0){
                JOptionPane.showMessageDialog(null, "SE HA AGREGADO UN NUEVO LIBRO");
                id.setText("");
                nom.setText("");
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    }
```

Figura 10. Código para agregar nodos de libros.

Se omitió la explicación de todo el resto del código dado que sólo es interés de este proyecto mostrar un ejemplo de conexión a una base de datos implementada en Neo4j y no la implementación de la interfaz gráfica; sin embargo a continuación se muestra el funcionamiento de una interfaz gráfica sencilla que se implementó para probar la funcionalidad de agregar libros.

En la figura 11 se muestra el formulario para agregar libros que consta del código y título del libro:

Figura 11. Pantalla para agregar libros.

En la Figura 12 se muestra el mensaje informativo para confirmar que se ha agregado el nodo del libro exitosamente:

Figura 12. Mensaje de información cuando el libro se agrega exitosamente.

Finalmente se puede consultar en nuestra interfaz de administración de Neo4j (*http://localhost:7474/browser/*), que se ha agregado el nodo del libro con título *La sombra del viento* (Figura 13)

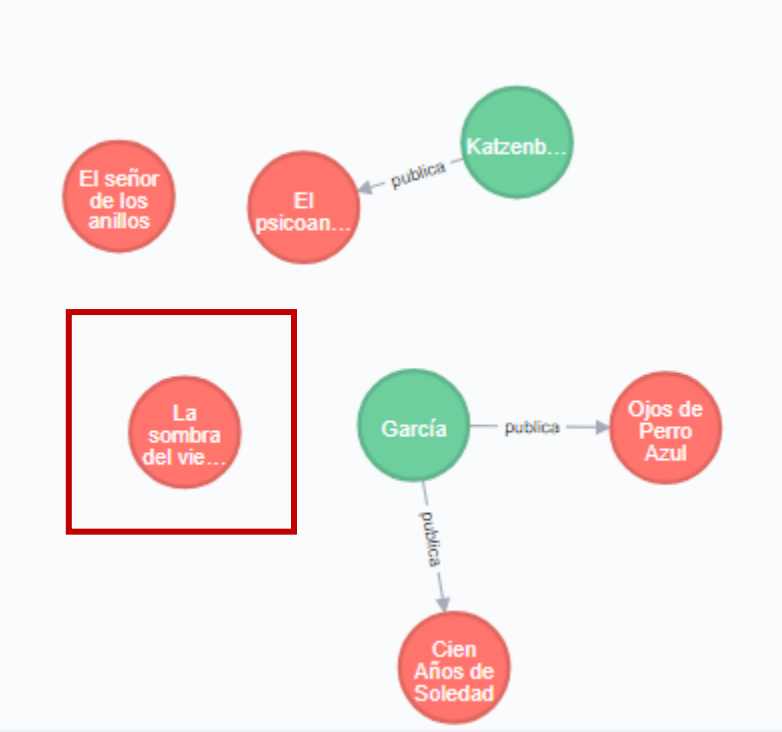


Figura 13. Consulta de los nodos existentes, incluyendo el nuevo nodo.

7. Anexos

Se incluyeron como anexos las dos clases de java para la conexión a la base de datos mediante un lenguaje de programación.

a. Clase conexión_neo4j

```
package interfaz;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;
import org.neo4j.driver.internal.*;
import org.neo4j.driver.v1.*;
import org.neo4j.driver.v1.types.*;
import org.neo4j.jdbc.*;

public class conexion_neo4j {
    Connection conectar=null;
    public Connection conexion(){
        Connection conectar=null;
        try {
              // EN ESTA SECCIÓN SE CARGA EL DRIVER PARA HACER LA CONEXIÓN
            Class.forName("org.neo4j.jdbc.Driver");
            conectar = (Connection) DriverManager.getConnection("jdbc:neo4j:bolt://localhost/?user=neo4j,password=123456,scheme=basic");
            System.out.println("CONEXIÓN EXITOSA CON NEO4J PUBLICACIONES");
               // MENSAJE QUE INDICA QUE LA CONEXIÓN SE ESTABLECIÓ CORRECTAMENTE
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("CONEXIÓN EXITOSA CON NEO4J");
            JOptionPane.showMessageDialog(null, "FALLÓ LA CONEXIÓN CON NEO4J PUBLICACIONES"+e);
        }
        return conectar;

    }
}
```

b. Clase AgregarLibro

```
package interfaz;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import org.neo4j.driver.v1.types.Node;

public class AgregarLibro extends javax.swing.JDialog {
conexion_neo4j Conexion = new conexion_neo4j();
Connection BD= Conexion.conexion();

    public AgregarLibro(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }

    public void nuevoNodo(){
       String codigo,titulo;
       String newNodo;
       codigo = id.getText();
       titulo = nom.getText();
       newNodo = "CREATE(lib:libro{codigo:'"+codigo+"', titulo:'"+titulo+"'})";
       try {
            PreparedStatement pst=BD.prepareStatement(newNodo);
            int n=pst.executeUpdate();
            if (n>0){
               JOptionPane.showMessageDialog(null, "SE HA AGREGADO UN NUEVO LIBRO");
               id.setText("");
               nom.setText("");
            }
        } catch (SQLException ex) {
           JOptionPane.showMessageDialog(null, ex.getMessage());
        }
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        id = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        nom = new javax.swing.JTextField();
        acep = new javax.swing.JButton();
        jButton4 = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setSize(new java.awt.Dimension(800, 400));
        jLabel3.setText("Código");
        jLabel4.setText("Título");
        acep.setText("Agregar Libro");
        acep.addActionListener(new java.awt.event.ActionListener() {
```

```
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            acepActionPerformed(evt);
        }
    });

    jButton4.setText("Cancelar");
    jButton4.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton4ActionPerformed(evt);
        }
    });

    jLabel2.setIcon(new javax.swing.ImageIcon(getClass().getResource("/iconos/book.png"))); // NOI18N
    jLabel2.setToolTipText("");
    jLabel8.setFont(new java.awt.Font("Tahoma", 1, 24)); // NOI18N
    jLabel8.setText("AGREGAR LIBROS");
    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(72, 72, 72)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabel4)
                        .addComponent(jLabel3))
                    .addGap(66, 66, 66)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(nom, javax.swing.GroupLayout.PREFERRED_SIZE, 272,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 52,
Short.MAX_VALUE)
                            .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(id, javax.swing.GroupLayout.PREFERRED_SIZE, 151,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addGap(0, 0, Short.MAX_VALUE))))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(acep, javax.swing.GroupLayout.PREFERRED_SIZE, 150,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(82, 82, 82)
                    .addComponent(jButton4, javax.swing.GroupLayout.PREFERRED_SIZE, 120,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jLabel1)))
            .addContainerGap())
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 202,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jLabel8)
            .addGap(0, 0, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 141,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(layout.createSequentialGroup()
                    .addGap(59, 59, 59)
                    .addComponent(jLabel8)))
            .addGap(20, 20, 20)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(116, 116, 116)
                    .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 41,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(36, 36, 36))
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(id, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel3))
                    .addGap(56, 56, 56)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(nom, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel4))
                    .addGap(61, 61, 61)))
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(0, 29, Short.MAX_VALUE)
                    .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(77, 77, 77))
                .addGroup(layout.createSequentialGroup()
                    .addGap(17, 17, 17)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(acep)
                        .addComponent(jButton4))
                    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
        );
        pack();
    }// </editor-fold>

    private void acepActionPerformed(java.awt.event.ActionEvent evt) {
    String Query;
    Query = "MATCH (libro) WHERE libro.codigo = {1} RETURN libro";
    try {
        PreparedStatement pst= BD.prepareStatement(Query);
        pst.setString(1, id.getText());
        ResultSet rs = pst.executeQuery();
        int cont = 0;
        while(rs.next()){
            cont++;
        }
        if (cont > 0) {
            JOptionPane.showMessageDialog(null, "EL CÓDIGO QUE INGRESÓ YA ESTÁ SIENDO USADO");
                id.setText("");
       nom.setText("");
        }else{
            String idlong = id.getText();
            int idtam = idlong.length();
            if(idtam > 0){
              nuevoNodo();
            }else{
```

```
            JOptionPane.showMessageDialog(null, "FALTA INGRESAR EL CÓDIGO DEL LIBRO");
        }
      }
      } catch (SQLException e) {
          JOptionPane.showMessageDialog(null, e.getMessage());
      }
   }

   private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
     id.setText("");
     nom.setText("");
     this.setVisible(false);
   }
   public static void main(String args[]) {
       try {
          for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels())
{
              if ("Nimbus".equals(info.getName())) {
                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
                 break;
              }
          }
      } catch (ClassNotFoundException ex) {
          java.util.logging.Logger.getLogger(AgregarLibro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
      } catch (InstantiationException ex) {
          java.util.logging.Logger.getLogger(AgregarLibro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
      } catch (IllegalAccessException ex) {
          java.util.logging.Logger.getLogger(AgregarLibro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
      } catch (javax.swing.UnsupportedLookAndFeelException ex) {
          java.util.logging.Logger.getLogger(AgregarLibro.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
      }
      /* Create and display the dialog */
      java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
              AgregarLibro dialog = new AgregarLibro(new javax.swing.JFrame(), true);
              dialog.addWindowListener(new java.awt.event.WindowAdapter() {
                  @Override
                  public void windowClosing(java.awt.event.WindowEvent e) {
                      System.exit(0);
                  }
              });
              dialog.setVisible(true);
          }
      });
   }
   private javax.swing.JButton acep;
   private javax.swing.JTextField id;
   private javax.swing.JButton jButton4;
   private javax.swing.JLabel jLabel1;
   private javax.swing.JLabel jLabel2;
   private javax.swing.JLabel jLabel3;
   private javax.swing.JLabel jLabel4;
   private javax.swing.JLabel jLabel7;
   private javax.swing.JLabel jLabel8;
   private javax.swing.JTextField nom;
   // End of variables declaration
}
```

8. Referencias

Silberschatz, A., Korth, H., & Sudarshan, S. (2002). *Fundamentos de Bases de Datos* 4° Edición. Madrid, España: McGraw-Hill.

Elmasri, R., Navathe, S. (2004). *Fundamentals of Database Systems*, 4th Edition. Boston, USA: Pearson Education Inc.

García-Molina H., Ullman, J. D., Widom J. (2002). *Database Systems: The Complete Book*. Upper Saddle River, USA: Prentice-Hall.

Date, C. J. (2003). *An Introduction to Database Systems*, 8th Edition. USA: Addison Wesley.

Carralero Colmenar, Nieves (2012). *ODMG – Estandarizando Bases de datos OO*. Revista Digital Sociedad de la Información N° 38.

Marqués, Merche (2002). *Diseño de Sistemas de Base de Datos.*

Pinilla, C., Bello, M., & Peña, C. (2017). *Bases de datos orientadas a grafos*. *Tia*, 1 53-160.

BBVAOpen4U. (2018). *Neo4j: qué es y para qué sirve una base de datos orientada a grafos*. Recuperado de: https://bbvaopen4u.com/es/actualidad/neo4j-que-es-y-para-que-sirve-una-base-de-datos-orientada-grafos [Consultado el 16 Diciembre de 2018]

Neo4j. (2018). *4.3. Aggregating functions.* Recuperado de: https://neo4j.com/docs/cypher-manual/current/functions/aggregating/ [Consultado el 16 Diciembre de 2018]