Universidad Nacional Autónoma de México

Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas

Bases de Datos No Estructurados

Práctica 4: Neo4j

Autores :

- **López Sánchez Misael**
- **Tapia López José de Jesús.**

Nos introducimos al uso de las bases de datos de tipo grafo, en este caso `Neo4j` de la cual crearemos una aplicación de recomendación de productos de **PROFECO**

OBJETIVO:

> Crear una base de datos de tipo grafo con los datos, diseñada para una aplicación que permita:

- Dado un estado y un producto, buscar lugares en los que pueda encontrarlo
- Dado un estado y una tienda, verificar si tiene algún incumplimiento con algun producto.
- Dado un estado y un producto, buscar alternativas sin incumplimiento de ese producto o categoría.

> Además, la aplicación debe de ser capaz de:
>
> - Llevar un registro de las compras y lugares que ha hecho cada usuario
> - Recomendar a un usuario una tienda en donde pueda encontrar en un solo lugar, lo que compra en diferentes tiendas.
> - Encontrar los estados con mayor y menor incumplimiento relativo al número de tiendas que tiene.
> - Recomendar productos a un usuario según lo que otros usuarios han comprado, i.e dado un usuairo y un producto obntener todos los productos de esa misma categoria que ha comprado otros usuarios que tambień han comprado ese mismo producto.

INTRODUCCIÓN

Neo4j es una base de datos orientada a grafos; la cual, como su nombre lo indica, usa grafos para representar datos y las relaciones entre ellos. Dichos datos los podemos representar como en Teoría de Gráficas, mediante nodos y aristas; en las que los nodos son los datos y las aristas son las relaciones entre datos. Cabe destacar que Neo4j usa grafos con etiquetas, en las que podemos asignar propiedades tanto a nodos como relaciones. Tanto los nodos como las aristas se pueden etiquetar. Las etiquetas se pueden usar para restringir las búsquedas. Por lo tanto, podemos hacer una analogía con una Base de Datos Relacional (BDR):

1. Nodos: Es como una tabla en BDR donde almacenamos los datos.
2. Relaciones: Es la conexión entre los datos que se asignó entre dos nodos.
3. Propiedades: Son son más que etiquetas que se pueden unir tanto a Nodos como a Relaciones.

Características principales:

- Base de datos transaccional compatible con ACID con almacenamiento y procesamiento de gráficos nativos
- Las bases de datos de gráficos garantizan la coherencia a través de las transacciones. No permiten relaciones colgantes: el nodo inicial y el nodo final siempre tienen que existir, y los nodos solo se pueden eliminar si no tienen ninguna relación asociada a ellos.
- Admite dos tipos de API Java: API Cypher y API Java nativa para desarrollar aplicaciones Java. Por tanto, proporciona un servidor robusto y un Lenguaje específico de dominio (DSL) llamado Cypher que se usa para crear, actualizar, consultar, explorar y descubrir nuevos datos y relaciones.
- Se puede decir que de cierta forma los datos son semi-estructurados.
- Puede importar datos.
- Está diseñado para optimizar la gestión, el almacenamiento y el recorrido rápidos de nodos y relaciones.
- Ofrece un rendimiento constante en tiempo real, lo que permite a las empresas crear aplicaciones para cumplir con los desafíos de datos en evolución actual.

Y es gracias a esto que podemos extraer valor agregado a los datos con gran rendimiento y de una forma ágil y flexible.

In [1]:

```
# Bibliotecas que utilizamos para trabajar.
import pandas as pd
import numpy as np
import requests
```

In [ ]:

```
pip install -U ipython
```

In [ ]:

```
#Importamos el documento proporcionado (lo cargamos en
# GitHub para leerlo más rápido
url = "https://raw.githubusercontent.com/jodejetalo99/BDNE/master/Profeco_2015.csv"

#Importamos los datos
dataProfeco = pd.read_csv(url,
        error_bad_lines=False, encoding='latin1')
# Este es el data frame con los datos originales del archivo Profeco_2015.csv
dataProfeco.head()
```

In [ ]:

```
# Nos deshacemos de las variables que no nos importan
dataProfeco=dataProfeco.drop(columns=['Unnamed: 0','ANIO'
  ,'UNIDAD_ADMINISTRATIVA','NUMERO_DE_EXPEDIENTE_DE_VERIFICACION',
                'FECHA_DE_VISITA','NORMA_OFICIAL_MEXICANA_VERIFICADA'])
dataProfeco.head()
```

Ajuste y modificaciones al documento para generar

nodos.

El siguiente pre-procesamiento lo realizamos para poder generar las relaciones entre productos y tiendas.
Por lo tanto, con los datos brindados en la práctica, generaremos a continuación una tabla productos y una tabla de las tiendas, en las que cada una tendrá sus correspondientes variables. Posteriormente, ambas tablas las exportaremos como *CSV*.
NOTA: Cuando mencionamos tablas, nos referimos a DataFrames.

Creador de identificadores únicos.

Para propósitos de esta práctica nosotros decidimos realizar un pre-procesamiento de la información del csv `Profeco_2015.csv`, y lo que decidimos fue partir el csv en dos tablas principales:

- tabla_productos -> p:Producto (en Neo4j)
- tabla_tiendas -> t:Tienda (en Neo4j)

Esto con el principal objetivo de poder fragmentar la información para poder crear indentificadores (id's) para identificar de mananera unica a cada tienda y a cada producto; y para crear la relación ***un producto se encuentra en una tienda***

Caso tabla_productos

En el caso de la tabla producto decidimos crear los índices utilizando 3 columnas:

- DESCRIPCION_DEL_PRODUCTO
- RAZON_SOCIAL_VISITADA
- MUNICIPIO

De esta manera, logramos identificar a cada uno de nuestros productos de forma única, principalmente por DESCRIPCION_DEL_PRODUCTO, pues existen diferentes productos catalogados con el mismo nombre, pero al leer la descripción son diferentes.

Caso tabla_tiendas

Para el pre-procesamiento de la creación de los identificadores de la tabla_tiendas, usamos un procedimiento análogo, en este caso agrupamos un identificador unico utilizando las columnas:

- RAZON_SOCIAL_VISITADA
- MUNICIPIO

De esta forma, logramos identificar de manera única por municipio a las diferentes tiendas existentes en la base de datos.

El objetivo principal de todos esto fue para poder fácilitar las consultas y los registros dentro de cada uno de los nodos que implementamos para diseñar nuestra base de datos que se explican más adelante.

In [ ]:

```
# Creamos una tabla de productos

# nos quedamos por un momento con la columna municipio en esta tabla
# pues puede haber entidades en las que
# la misma tienda (sucursal) puede estar diferentes municipios
tabla_productos = dataProfeco[["TIPO_DE_PRODUCTO",
  "DESCRIPCION_DEL_PRODUCTO","RAZON_SOCIAL_VISITADA","MUNICIPIO",
  "INCUMPLIMIENTO","CLASIFICACION_RESULTADO_PRODUCTO"]]
tabla_productos['ID_PRODUCTO'] = tabla_productos.groupby(['DESCRIPCION_DEL_PRODUCTO
    ,'RAZON_SOCIAL_VISITADA','MUNICIPIO']).ngroup()

tabla_productos.sort_values(by="ID_PRODUCTO",ascending=True)
# tabla_productos = tabla_productos.drop_duplicates()
# Checamos que tenemos todos nuestros productos identificados
# por tienda correctamente
tabla_productos = tabla_productos.sort_values(by="ID_PRODUCTO",
                ascending=True)
tabla_productos.head()
```

In [ ]:

```
# Creamos la tabla de las tiendas

tabla_tiendas = dataProfeco[["RAZON_SOCIAL_VISITADA",
        "GIRO_COMERCIAL","MUNICIPIO","ENTIDAD"]]
tabla_tiendas['ID_RAZON_SOCIAL_VISITADA'] = (tabla_tiendas.groupby(
    ['RAZON_SOCIAL_VISITADA','MUNICIPIO']).ngroup())
#tabla_tiendas = tabla_tiendas.drop_duplicates()
#tabla_tiendas.ID_TIENDA =tabla_tiendas.ID_TIENDA.astype(str)
tabla_tiendas = tabla_tiendas.sort_values(by="ID_RAZON_SOCIAL_VISITADA",
                                          ascending=True)
tabla_tiendas.head()
```

Al crear las relaciones de los id's como lo hicimos, se repetian id's por lo que con el siguiente comando simplemente los eliminamos y nos quedamos con la ifnormación lista para poder ser usada para nuestros fines.

In [0]:

```
tabla_tiendas = tabla_tiendas.drop_duplicates()
tabla_productos = tabla_productos.drop_duplicates()
```

In [0]:

```
# estas últimas do tablas (dataframes) los exportamos como csv
#tabla_productos.to_csv('productos2.csv')
#tabla_tiendas.to_csv('tiendas.csv')
```

Sección donde agregamos los datos a Neo4j

Se importan desde un csv a Neo4j

In [ ]:

```
# Instalamos la biblioteca py2neo que nos permitiran trabajar
# la conección entre NEO4j y Python
!pip install py2neo
```

In [ ]:

```
# Instalamos Neo4j para trabajar con el.
# download 3.5.8 or neo4j-enterprise-4.0.0-alpha09mr02-unix
!curl https://neo4j.com/artifact.php?name=neo4j-community-3.5.8-unix.tar.gz
    -o neo4j.tar.gz
# decompress and rename
!tar -xf neo4j.tar.gz  # or --strip-components=1
!mv neo4j-community-3.5.8 nj
# disable password, and start server
!sed -i '/#dbms.security.auth_enabled/s/^#//g' nj/conf/neo4j.conf
!nj/bin/neo4j start
```

Estos CSV creados anteriormente, los cuales exportamos a google dirve, los subimos al GitHub de José de Jesús; y los leemos desde ahí para facilitarnos la carga de los datos en nuestra base de datos de tipo grafos. Además, notemos que cargamos otros dos CSV, los cuales los inventamos de tal forma que nos sirvan para los objetivos de la práctica.

In [11]:

```
from py2neo.data import Node, Relationship
from py2neo import Graph, NodeMatch
graph = Graph("bolt://localhost:7687")

# Importamos los datos y creamos nuestros nodos
graph.run("LOAD CSV WITH HEADERS FROM ordenes_neo.csv' AS
          'row CREATE (n:Orden) SET n = row")
graph.run("LOAD CSV WITH HEADERS FROM productos.csv' AS
          'row CREATE (n:Producto) SET n = row")
graph.run("LOAD CSV WITH HEADERS FROM tiendas.csv' AS
          'row CREATE (n:Tienda) SET n = row")
graph.run("LOAD CSV WITH HEADERS FROM usuarios_neo.csv' AS
          'row CREATE (n:Usuario) SET n = row")
```

Out[11]:

```
<py2neo.database.Cursor at 0x7fa8a8442f60>
```

```
<div style="text-align: justify"> Creamos las relaciones entre nodos uniendo las
tablas de acuerdo a sus atributos. </div>
```

In [12]:

```
# Hacemos las relaciones entre los nodos

graph.run("MATCH (u:Usuario),(o:Orden) WHERE
          'u.ID_USUARIO = o.ID_USUARIO CREATE (u)-[:Hace]->(o)")

graph.run("MATCH (o:Orden),(t:Tienda) WHERE
 'o.ID_RAZON_SOCIAL_VISITADA = t.ID_RAZON_SOCIAL_VISITADA CREATE
          '(o)-[:Se_hace_en]->(t)")

graph.run("MATCH (o:Orden),(p:Producto) WHERE
 'o.ID_PRODUCTO = p.ID_PRODUCTO CREATE (o)-[:Tiene]->(p)")

graph.run("MATCH (p:Producto),(t:Tienda) WHERE
 'p.RAZON_SOCIAL_VISITADA = t.RAZON_SOCIAL_VISITADA CREATE
          '(p)-[:Se_encuentra_en]->(t)")
```

Out[12]:

```
<py2neo.database.Cursor at 0x7fa8a9fb1048>
```

In [0]:

```
#graph.run("MATCH (n) DETACH DELETE n")
```

Descipción de los Nodos

Como se habrá observado en la base anterior, nuestra base de datos de tipo grafo tiene como tal 4 tipos de nodos, los cuales interactuan entre sí para poder almacenar la información y poder realizar las consultas. Dichos nodos los mostramos a continuación y colocamos los atributos que posee cada uno de ellos.

1.- **u:Usuario** >> el cual posee los datos de los consumidores

- ID_USUARIO
- NOMBRE
- APELLIDO
- EDAD

2.- **o:Orden** >> Contiene la información detallada de los usuarios por órdenes:

- ID_ORDEN
- ID_USUARIO
- CANTIDAD
- ID_PRODUCTO
- ID_RAZON_SOCIAL_VISITADA

3.- **p:Producto** >> Contiene la información de los productos identificados de manera única como lo hicimos en la tabla_productos:

- ID_PRODUCTO
- TIPO_DE_PRODUCTO
- DESCRIPCION_DEL_PRODUCTO
- RAZON_SOCIAL_VISITADA
- MUNICIPIO
- INCUMPLIMIENTO
- CLASIFICACION_RESULTADO_PRODUCTO

4.- **t:Tienda** >> Contiene la información de la ubicación y descrispción de cada una de las tiendas:

- ID_RAZON_SOCIAL_VISITADA
- RAZON_SOCIAL_VISITADA
- GIRO_COMERCIAL
- MUNICIPIO
- ENTIDAD

Es por eso que las relaciones fueron creadas del modo en que se presentó en la celda anterior:

- Un **usuario** *hace* una **orden**
- Una **orden** *se hace* en una **tienda**
- Una **orden** *tiene* **productos**
- Un **producto** *se encuentra* en una **tienda**

APLICACIÓN

La siguiente es la aplicación principal de la actividad sobre productos requerida utilizando *Neo4j*. Cuenta con un menú con las siguientes opciones:
1.- **Ingresar a un nuevo usuario** - Se pueden dar de alta a nuevos usuarios dentro de la base de datos.

2.- **Ingresar una nueva compra de un usuario** - Los usuarios pueden ingresar nuevas compras que ellos mismos hayan realizado.

3.- **Buscar productos por estados** - El usuario brinda un estado y un producto, para devolverle lugares (Específicamente le muestra el nombre de la tienda y el municipio) donde pueda encontrarlo.

4.- **Revisar incumplimientos por estados y tiendas**- El usuario indica un estado y una tienda, y le muestra si tiene algun incumplimiento con un producto.

5.- **Localizar alternativas de productos** - El usuario brinda un estado y un producto, y le enseña alternativas sin incumplimiento de ese producto o categoría. Es decir, dado que algunos productos tienen incumplimiento, es mejor buscarlos en otras tiendas que no tengan incumplimiento del producto especificado.

6.- **Registro de compras de un usuario** - Se muestra un registro de las compras en la tienda que ha hecho el usuario.

7.- **Recomendaciónes de establecimientos para encontrar todos los productos que ha comprado un usuario** - Se le recomienda al usuario una tienda en donde pueda encontrar en un solo lugar, lo que compra en diferentes tiendas.

8.- **Nivel de incumplimiento relativo de cada estado** - Se muestra los estados con mayor y menor incumplimiento relativo conforme al número de tiendas que tiene.

9.- **Recomendaciones de productos** - Se le recomienda al usuario productos, según lo que otros usuarios han comprado. Se obtienen todos los productos de esa misma categoría que han comprado otros usuarios que también han comprado ese mismo producto.

10.- **Frecuencia de tipos de productos comprados por los usuarios** - De los registros de órdenes que se tienen registrados, se hace una tabla y se muestra la frecuencia de los diferentes tipos de productos comprados por los usuarios.

11.- **Disponibilidad de productos de un estado** - Dado un estado por el usuario, se muestra una tabla de frecuencia los productos disponibles en ese estado.

12.- **Disponibilidad de tiendas de un estados** - Dado un estado por el usuario, se muestra una tabla con las tiendas que existen en dicho estado (útil para identificar grandes cadenas comerciales como Oxxo, Bodega Aurrera, etc)

13.- **Número de productos ofrecidos por cada tienda de un estado** - Dado un estado por el usuario, se mustra una tabla con la frecuencia de productos que ofrece cada tienda de ese estado.

14.- **Disponibilidad de cada producto que ofrece una tienda** - Dada una razón social especificada por el usuario, se le muestran los productos que esta ofrece y la cantidad de cada producto en dicha razón social.

15.- **Salir** - Fin de la Aplicación.

In [0]:

```
# Comando para guardar los cambios permantenes en la base de datos
tx = graph.begin()
```

In [ ]:

```
import os
import json

os.system('cls')
clear = lambda:os.system('cls')

from py2neo import NodeMatcher
nodes_matcher = NodeMatcher(graph)
nodes_matcher.match()

def menu_aplication():

    print('+'*10,"PLATAFORMA DE COMPRAS", '+'*10, '\n')
    print('+'*20," MENÚ ", '+'*20, '\n')
    print("1.- Ingresar a un nuevo usuario ")
    print("2.- Ingresar una nueva compra de un usuario ")
    print("3.- Buscar productos por estados ")
    print("4.- Revisar incumplimientos por estados y tiendas ")
    print("5.- Localizar alternativas de productos ")
    print("6.- Registro de compras de un usuario ")
    print("7.- Recomendaciónes de establecimientos para encontrar
          'todos los productos que ha comprado un usuario  ")
    print("8.- Nivel de incumplimiento relativo de cada estado ")
    print("9.- Recomendaciones de productos ")
    print("10.- Frecuencia de tipos de productos comprados por los usuarios ")
    print("11.- Disponibilidad de productos de un estado ")
    print("12.- Disponibilidad de tiendas de un estado ")
    print("13.- Número de productos ofrecidos por cada tienda de un estado ")
    print("14.- Disponibilidad de cada producto que ofrece una tienda ")
    print("15.- Salir ")
    print("")


# Iniciamos la interfaz de la aplicación
while True:
    menu_aplication()
    opMenu = input("Ingrese una opción >> ")
    clear()

    if opMenu == '1':
        print('-'*5," Ingresar a un nuevo usuario ", '-'*5, '\n')
        #Ingresamos a un nuevo usuario dentro de la tabla de usuarios.

        # Primero contamos cuántos usuarios hay en la base de datos
        id_usuario = graph.run('MATCH (u:Usuario) RETURN count(u) as count')
          .to_data_frame()['count'][0]

        # Como la enumeración de los ID's de los usuarios empiezan
        # desde cero, a los nuevos usuarios le colocaremos la cantidad de
        # la consulta anterior, para asignar un ID de manera ordenada
        # (ascendente) y así evitar usuarios distintos con mismos ID's

        # Ahora, en esta parte pedimos que brinde la información siguiente
        # del usuario:
        nombre_usuario = str(input("Ingrese el nombre del usuario: "))
        apellido_usuario = str(input("Ingrese el apellido del usuario: "))
        edad_usuario = int(input("Ingrese la edad del usuario: "))

        # Guardamos la información
```

```
        graph.run('CREATE (u:Usuario{ID_USUARIO:\"'+str(id_usuario)+'\",NOMBRE:\"'
                  +nombre_usuario+'\", APELLIDO:\"'+apellido_usuario+
                  '\",EDAD:\"'+str(edad_usuario)+'\"})')

    elif opMenu == '2':
        print('-'*5," Ingresar una nueva compra de un usuario ", '-'*5, '\n')

        # Primero contamos cuántos ordenes hay en la base de datos

        id_orden = graph.run('MATCH (o:Orden) RETURN count(o) as count')
          .to_data_frame()['count'][0]

        # Como la enumeración de los ID's de las ordenes empiezan
        # desde cero, a los nuevas ordenes les colocaremos la cantidad
        # de la consulta anterior, para asignar un ID
        # de manera ordenada (ascendente), es decir, valga la redundancia,
        # llevar un orden con las ordenes de los usuarios

        # Ahora, en esta parte pedimos que brinde la siguiente
        # información sobre la orden:
        id_usuario = int(input("Ingrese el ID del usuario: "))
        id_producto = str(input("Ingrese el ID del producto que compró: "))
        id_tienda = str(input("Ingrese el ID de la tienda en la que
                              'compró ese producto: "))
        cantidad_producto = int(input("Ingrese la cantidad que compró
                                      'de ese producto: "))


        graph.run('MATCH (u:Usuario{ID_USUARIO:\"'+str(id_usuario)
                  +'\"})  CREATE (o:Orden {ID_ORDEN:\"'+str(id_orden)+
                  '\", ID_USUARIO:\"'+str(id_usuario)+'\", \
                    CANTIDAD:\"'+str(cantidad_producto)+'\",ID_PRODUCTO:\"'+
                  str(id_producto)+'\",ID_RAZON_SOCIAL_VISITADA:\"'+str(id_tienda)
                  +'\"}) CREATE (u)-[:Hace]->(o)')

    elif opMenu == '3':

        print('-'*5," Buscar productos por estados ", '-'*5, '\n')
        #Buscar productos por estados.
        # Solicitamos al usuairo que ingrese el producto y la entidad
        producto = json.dumps(input("Ingrese producto : "))
        entidad = json.dumps(input("Ingrese entidad: "))

        # Modificamos la cadena de texto para que pueda ser leido.
        mi_entidad = "{" + "ENTIDAD:" + entidad + "}"
        mi_producto = "{" + "TIPO_DE_PRODUCTO:" + producto + "}"

        # Corremos la consulta

        consult3 = graph.run(f'MATCH (p:Producto {mi_producto})
                -[:Se_encuentra_en]-> (t:Tienda {mi_entidad}) RETURN
                DISTINCT p.RAZON_SOCIAL_VISITADA,t.MUNICIPIO').to_data_frame()
        display(consult3)
    elif opMenu == '4':

        print('-'*5," Revisar incumplimientos por estados y tiendas ",'-'*5,'\n')
        #Solicitamos la razon social visitada y el estado
        razonSocial = json.dumps(input("Ingrese la tienda que desea consultar:"))
        entidad = json.dumps(input("Ingrese entidad: "))

        #Modificamos la cadena de texto ajustando al formato
```

```
        mi_entidad = "{" + "ENTIDAD:" + entidad + "}"
        mi_razonS = "{" + "RAZON_SOCIAL_VISITADA:" + razonSocial + "}"

        #Realizamos la consulta
        consult4 = graph.run(f'MATCH (p:Producto {mi_razonS})
                -[:Se_encuentra_en]-> (t:Tienda {mi_entidad}) RETURN
                p.INCUMPLIMIENTO, p.TIPO_DE_PRODUCTO').to_data_frame()
        display(consult4)

    elif opMenu == '5':
        print('-'*5," Localizar alternativas de productos ", '-'*5, '\n')
        # Buscamos alternativas sin incumplimiento dato un producto
        # y un estado Solicitamos al usuairo que ingrese el producot y la entidad
        producto = json.dumps(input("Ingrese producto : "))
        entidad = json.dumps(input("Ingrese entidad: "))

        # Modificamos la cadena de texto para que pueda ser leido.
        mi_entidad = "{" + "ENTIDAD:" + entidad + "}"
        mi_producto = "{" + "TIPO_DE_PRODUCTO:" + producto + "}"

        consult5 = graph.run(f'MATCH (p:Producto{mi_producto})
            -[:Se_encuentra_en]-> (t:Tienda {mi_entidad}) WHERE
            p.INCUMPLIMIENTO = "NO SE DETECTO INCUMPLIMIENTO" RETURN DISTINCT
            p.RAZON_SOCIAL_VISITADA').to_data_frame()
        display(consult5)

    elif opMenu == '6':

        print('-'*5," Registro de compras de un usuario ", '-'*5, '\n')
        id_usuario = json.dumps(input("Ingrese el identificador
                                'del usuario que desea consultar: "))
        mi_usuario = "{" + "ID_USUARIO:" + id_usuario + "}"
        consult6 = graph.run(f'MATCH (u:Usuario{mi_usuario})
        MATCH (o:Orden) MATCH (p:Producto)  WHERE
        u.ID_USUARIO = o.ID_USUARIO AND o.ID_PRODUCTO = p.ID_PRODUCTO
        RETURN DISTINCT u.ID_USUARIO, u.NOMBRE, o.ID_ORDEN, o.CANTIDAD,
        p.TIPO_DE_PRODUCTO, p.RAZON_SOCIAL_VISITADA').to_data_frame()
        display(consult6)

    elif opMenu == '7':
        print('-'*5," Recomendaciones de establecimientos para encontrar todos
              'los productos que ha comprado un usuario ", '-'*5, '\n')
        # Recomendar a un usuario una tienda donde se pueda encontrar
        # en un sólo lugar,
              #lo que compra en diferentes tiendas
        # Pedimos introducir el tipo de usuario
        usuario = json.dumps(input("Ingrese el identificador del
                                'usuario que desea consultar: "))
        mi_usuario = "{" + "ID_USUARIO:" + usuario + "}"

        # Consultamos los productos de un usuario
        consult7 = graph.run(f'MATCH (o:Orden{mi_usuario}) -[:Tiene]-> (p:Producto)
            RETURN DISTINCT  o.ID_USUARIO, p.ID_PRODUCTO, p.TIPO_DE_PRODUCTO,
            p.DESCRIPCION_DEL_PRODUCTO,  p.RAZON_SOCIAL_VISITADA').to_data_frame()
        # Esta lista es para recorrer los indices y buscar coincidencias
        mis_productos_id = list(consult7['p.ID_PRODUCTO'])
        # La segunda es para llamar a todas las posibles coincidencias
        mis_productos = list(consult7['p.TIPO_DE_PRODUCTO'])
        mis_productos = list(dict.fromkeys(mis_productos))

        # si solo compra productos de un mismo tipo, le devolvemos todas las
```

```
        # tiendas donde puede conseguir esos artículos
        if len(mis_productos) == 1:
            comunes = graph.run(f'MATCH (p:Producto) WHERE
            p.TIPO_DE_PRODUCTO ="{mis_productos[0]}"  RETURN DISTINCT
            p.RAZON_SOCIAL_VISITADA,p.ID_PRODUCTO').to_data_frame()

        # De lo contrario hacemos toda una maquinaria para calcular las
        # tiendas donde son ofertados los productos que compra habitualmente
        else:
            lista_productos=[]
            for i in range(len(mis_productos)):
                comun_n = graph.run(f'MATCH (p:Producto) WHERE
                p.TIPO_DE_PRODUCTO ="{mis_productos[i]}"  RETURN DISTINCT
                p.RAZON_SOCIAL_VISITADA, p.ID_PRODUCTO').to_data_frame()
                lista_productos.append(comun_n)

            #comunes = pd.merge(lista_productos[0],lista_productos[1],
            # on='p.RAZON_SOCIAL_VISITADA')

            lista_comunes=[]
            for i in range(len(lista_productos)-1):
                comunes = pd.merge(lista_productos[i],lista_productos[i+1],
                        on='p.RAZON_SOCIAL_VISITADA')
                comunes= comunes.merge(lista_productos[i+1])



            # Lugares donde puede encontrar ambos
            #comunes = pd.merge(comun_2,comun_1, on='p.RAZON_SOCIAL_VISITADA')
            comunes = comunes['p.RAZON_SOCIAL_VISITADA']
            comunes = comunes.drop_duplicates()
            # En principio aquí ya tenermos los lugares
            # donde puede encontrar sus articulos, pero podemos expandir
            # la consulta  dando posibles articulos
        display(comunes)

    elif opMenu == '8':
        print('-'*5," Nivel de incumplimiento relativo de cada estado ",'-'*5
              , '\n')
        # Primero seleccionamos la frecuencia de cada entidad
        # (cuantas tiendas aparecen en cada municipio)

        a = graph.run('MATCH (p:Producto) -[:Se_encuentra_en]-> (t:Tienda)
            RETURN t.ENTIDAD, count(p.RAZON_SOCIAL_VISITADA) AS frecuencia
            ORDER BY t.ENTIDAD').to_data_frame()

        # Despues seleccionamos la frecuencia de cada entidad
        # (cuantas tiendas aparecen en cada municipio) pero que tenga
        # incumplimientos

        b = graph.run('MATCH (p:Producto) -[:Se_encuentra_en]-> (t:Tienda)
            WHERE p.INCUMPLIMIENTO <> "NO SE DETECTO INCUMPLIMIENTO" MERGE
            (p) -[:Se_encuentra_en]-> (t) RETURN t.ENTIDAD,
            count(p.RAZON_SOCIAL_VISITADA)
            AS frecuencia ORDER BY t.ENTIDAD').to_data_frame()

        # Crearemos un nuevo DataFrame, el cual nos servirá para mostrar
        # el resultado de las consultas anteriores, y obviamente
        # nos basaremos en dichas consultas anteriores para
        # mostrar de manera descendente  los estados con mayor y menor
        # incumplimiento relativo al número de tiendas que tiene.
```

```
        c = pd.DataFrame(a['t.ENTIDAD'])

        # como en el diseño de nuestra base de datos hace que todos
        # los estados tengan
        # incumplimientos, podemos obtener el incumplimiento relativo
        # directamente por medio de una división: la frecuencia de
        # cada estado en la
        # consulta b entre la frecuencia de cada estado en la consulta a
        # (Es decir, el tamaño de a y de b es de 32, y ambos
        #están acomodados  en orden alfabético)

        c['INCUMPLIMIENTO RELATIVO'] = b['frecuencia']/a['frecuencia']

        # Finalmente, el resultado de la división anterior los ponemos
        # descendentemente según el incumpliento relativo
        consulta_8 = c.sort_values('INCUMPLIMIENTO RELATIVO',ascending=False)

        # Mostramos la consulta pedida:
        display(consulta_8)

    elif opMenu == '9':
        print('-'*5," Recomendaciones de productos ", '-'*5, '\n')
        # primero mandamos a llamar toda la tabla, donde conocemos exactamente
        # que producto ha comprado cada usuario
        lista =graph.run('MATCH (o:Orden) -[:Tiene]-> (p:Producto)
        RETURN DISTINCT
        o.ID_USUARIO,o.ID_PRODUCTO, p.TIPO_DE_PRODUCTO,p.DESCRIPCION_DEL_PRODUCTO,
        p.RAZON_SOCIAL_VISITADA').to_data_frame()
        # Agrupamos desde python porque no esta dicha funcion en Neo4j
        lista.groupby('p.TIPO_DE_PRODUCTO')

        # En un data frame aparte filtramos  las compras de un tipo
        # de usuario en específico
        usuario = input("Introduzca el ID del usuario para buscar recomendaciones:
        lista_usuario=lista[lista['o.ID_USUARIO']==usuario]

        print("Los productos que ha comprado este usuario son:\n",
              lista_usuario['p.TIPO_DE_PRODUCTO'])


        tipo_producto = input("Introduzca el tipo de producto sobre
                        'el cual quieres buscar recomendaciones ")
        lista_recomendacion = lista[lista['p.TIPO_DE_PRODUCTO']==tipo_producto]
        display(lista_recomendacion)

    elif opMenu == '10':
        print('-'*5," Frecuencia de tipos de productos comprados por los usuarios"
              ,'-'*5,'\n')
        productos = graph.run("MATCH (u:Usuario) MATCH (o:Orden) MATCH (p:Producto)
        WHERE u.ID_USUARIO = o.ID_USUARIO AND o.ID_PRODUCTO = p.ID_PRODUCTO
        RETURN p.TIPO_DE_PRODUCTO, count(o.ID_PRODUCTO) AS FRECUENCIA ORDER BY
        FRECUENCIA DESC").to_data_frame()
        display(productos)

    elif opMenu == '11':

        print('-'*5," Disponibilidad de productos de un estado ", '-'*5, '\n')
        entidad = json.dumps(input("Ingrese la entidad donde desea buscar: "))
        mi_entidad = "{" + "ENTIDAD:" + entidad + "}"
        cantidad_producto = graph.run(f'MATCH (p:Producto) -[:Se_encuentra_en]->
```

```
                                    (t:Tienda{mi_entidad})
        RETURN p.TIPO_DE_PRODUCTO, count(p.RAZON_SOCIAL_VISITADA) AS FRECUENCIA
        ORDER BY FRECUENCIA DESC').to_data_frame()
        display(cantidad_producto)

    elif opMenu == '12':
        print('-'*5," Disponibilidad de tienda de un estado ", '-'*5, '\n')
        entidad = json.dumps(input("Ingrese el estado en donde desea buscar: "))
        mi_entidad = "{" + "ENTIDAD:" + entidad + "}"
        disp_tiendas = graph.run(f'MATCH (p:Producto) -[:Se_encuentra_en]->
                                 (t:Tienda{mi_entidad})
        RETURN DISTINCT p.RAZON_SOCIAL_VISITADA').to_data_frame()
        display(disp_tiendas)


    elif opMenu == '13':
        print('-'*5," Número de productos ofrecidos por cada tienda de un estado",
              '-'*5, '\n')
        entidad = json.dumps(input("Ingrese la entidad donde desea buscar: "))
        mi_entidad = "{" + "ENTIDAD:" + entidad + "}"
        cantidad_producto = graph.run(f'MATCH (p:Producto) -[:Se_encuentra_en]->
                                    (t:Tienda{mi_entidad})
        RETURN t.RAZON_SOCIAL_VISITADA, count(p.TIPO_DE_PRODUCTO) AS FRECUENCIA
        ORDER BY FRECUENCIA DESC').to_data_frame()
        display(cantidad_producto)

    elif opMenu == '14': # Devuelve el numero de sucursales que ofren
        print('-'*5," Disponibilidad de cada producto que ofrece una tienda",
              '-'*5, '\n')
        razon_social = json.dumps(input("Ingrese la tienda donde desea buscar:"))
        mi_razon_social = "{" + "RAZON_SOCIAL_VISITADA:" + razon_social + "}"
        cantidad_producto = graph.run(f'MATCH (p:Producto{mi_razon_social})
                -[:Se_encuentra_en]->
        (t:Tienda) RETURN p.TIPO_DE_PRODUCTO, count(p.TIPO_DE_PRODUCTO)
        AS FRECUENCIA ORDER BY FRECUENCIA DESC').to_data_frame()
        display(cantidad_producto)


    elif opMenu == '15':
        break;

    else:
        print("Opción invalida. Por favor, vuelva a intentarlo.")
    input("\n\n Presione ENTER para continuar.")
    clear()
```

Funcionamiento de la aplicación.

A continuación presentamos capturas del funcionamiento de cada una de las consultas de nuestra aplicación.

Iniciamos con el menú principal de la aplicación el cual tiene 15 opciones

```
++++++++++ PLATAFORMA DE COMPRAS ++++++++++

++++++++++++++++++++  MENÚ  ++++++++++++++++++++

1.- Ingresar a un nuevo usuario
2.- Ingresar una nueva compra de un usuario
3.- Buscar productos por estados
4.- Revisar incumplimientos por estados y tiendas
5.- Localizar alternativas de productos
6.- Registro de compras de un usuario
7.- Recomendaciónes de establecimientos para encontrar todos los productos que ha comprado un usuario
8.- Nivel de incumplimiento relativo de cada estado
9.- Recomendaciones de productos
10.- Frecuencia de tipos de productos comprados por los usuarios
11.- Disponibilidad de productos de un estado
12.- Disponibilidad de tiendas de un estado
13.- Número de productos ofrecidos por cada tienda de un estado
14.- Disponibilidad de cada producto que ofrece una tienda
15.- Salir
```

En la primera opción introducimos los datos de un nuevo usuario

```
Ingrese una opción >> 1
-----  Ingresar a un nuevo usuario  -----

Ingrese el nombre del usuario: ALEJANDRO
Ingrese el apellido del usuario: PIMENTEL
Ingrese la edad del usuario: 25


 Presione ENTER para continuar.
```

Introducimos el registro de nuestro usuario introducido en la opción anterior

```
Ingrese una opción >> 2
-----  Ingresar una nueva compra de un usuario  -----

Ingrese el ID del usuario: 11
Ingrese el ID del producto que compró: 8150
Ingrese el ID de la tienda en la que compró ese producto: 1281
Ingrese la cantidad que compró de ese producto: 3
```

Buscamos la disponibilidad de un producto en un estado (se muestra solamente una parte de la consulta porqu no cupo completa en una imagen)

```
Ingrese una opción >> 3
-----  Buscar productos por estados  -----

Ingrese producto : BRANDY
Ingrese entidad: AGUASCALIENTES
```

	p.RAZON_SOCIAL_VISITADA	t.MUNICIPIO
0	CADENA COMERCIAL OXXO SA DE CV	AGUASCALIENTES
1	LA BOMBILLA VINOS SA DE CV	AGUASCALIENTES
2	LA CANTINA DE ANTAÃO - JOSE LUIS MAGAÃA GONZ...	AGUASCALIENTES
3	GRUPO PERA DEL CENTRO S.A. DE C.V.	AGUASCALIENTES
4	CADENA COMERCIAL OXXO SA DE CV (MORELOS)	AGUASCALIENTES
5	ANTIGUA HACIENDA LA NORIA- HECTOR FLORES AGUILAR	AGUASCALIENTES
6	LOS RIELES	AGUASCALIENTES
7	RUIDOSO - PEYC CONSTRUCCIONES S.A. DE C.V.	AGUASCALIENTES
8	MERENDERO SAN PANCHO - EXQUISIMAR S.A. DE C.V.	AGUASCALIENTES
9	SINATRA LIVE - MARIOLLA S.A. DE C.V.	AGUASCALIENTES
10	EL ALMACEN DEL BIFE- SI NO HAY PARA TODOS NO H...	AGUASCALIENTES
11	MOTEL PEDRA- PEDRA S.A. DE C.V.	AGUASCALIENTES
12	MOTEL LUA- COMPAÃIA HOTELERA PESCADORES S.A....	AGUASCALIENTES
13	CADENA COMERCIAL OXXO S. A. DE C. V.	AGUASCALIENTES
14	TIENDAS SORIANA S.A. DE C.V.	AGUASCALIENTES
15	CADENA COMERCIAL OXXO SA DE CV (COLON)	AGUASCALIENTES
16	CARNITAS SAN PANCHO - SIGELFREDO DURAN ESPARZA	AGUASCALIENTES

Detectamos el cumplimiento o incumplimiento por producto

```
Ingrese una opción >> 4
-----  Revisar incumplimientos por estados y tiendas  -----

Ingrese la tienda que desea consultar: ASFEIXAS  S.A. DE C.V.
Ingrese entidad: AGUASCALIENTES
```

	p.INCUMPLIMIENTO	p.TIPO_DE_PRODUCTO
0	NO SE DETECTO INCUMPLIMIENTO	BRANDY

Damos la posibilidad de buscar alternativas de un producto sin incumplimiento

```
Ingrese una opción >> 5
-----  Localizar alternativas de productos  -----

Ingrese producto : WHISKY
Ingrese entidad: AGUASCALIENTES
```

	p.RAZON_SOCIAL_VISITADA
0	CADENA COMERCIAL OXXO SA DE CV
1	LA BOMBILLA VINOS SA DE CV
2	SONORA'S MEAT BOUTIQUE DE CORTES FINOS
3	THAI AGUASCALIENTES S. DE R.L. DE C.V.
4	TIENDAS EXTRA S.A DE C.V

Mostramos el registro de compras de un usuario en la base de datos

```
Ingrese una opción >> 6
-----  Registro de compras de un usuario  -----

Ingrese el identificador del usuario que desea consultar: 11
```

	u.ID_USUARIO	u.NOMBRE	o.ID_ORDEN	o.CANTIDAD	p.TIPO_DE_PRODUCTO	p.RAZON_SOCIAL_VISITADA
0	11	ALEJANDRO	11	3	PRODUCTOS ALIMENTICIOS	MERCADO V Y M

Buscamos recomendaciones de un producto con respecto a los demás usuarios

```
Ingrese una opción >> 7
-----  Recomendaciónes de establecimientos para encontrar todos los productos que ha comprado un usuario  -----

Ingrese el identificador del usuario que desea consultar: 7
0                      CADENA COMERCIAL OXXO S.A. DE C.V.
140          NUEVA WAL MART DE MEXICO  S. DE R.L. DE C.V.
184                          7 ELEVEN MEXICO S.A. DE C.V.
220                            TIENDAS EXTRA S.A. DE C.V.
228     TIENDAS COMERCIAL MEXICANA  S.A. DE C.V.   ( M...
240                   CADENA COMERCIAL OXXO  S.A. DE C.V.
2090    VINOS Y LICORES LA PRINCIPAL Y/O SALVADOR FERN...
2091                         TIENDAS SORIANA  S.A. DE C.V.
2191                           TIENDAS EXTRA  S.A. DE C.V.
2206    NUEVA WAL MART DE MEXICO  S. DE R.L. DE C.V. Y...
2207              TIENDAS COMERCIAL MEXICANA  S.A. DE C.V.
2208          NUEVA WAL MART DE MEXICO S. DE R.L. DE C.V.
2209                         TIENDAS TRES B  S.A. DE C.V.
2236            LA VIKINA Y/O ROSA MARIA ALVAREZ ILLESCAS
2237               SERVICIO COMERCIAL GARIS  S.A. DE C.V.
2239       CADENA COMERCIAL OXXO  S.A. DE C.V.   ( OXXO )
2249                       7 ELEVEN MEXICO  S.A. DE C.V.
2265      ABARROTES 'MAGALY'   Y/O ROSA MA. PALACIOS DIAZ
2266    NUEVA WAL MART DE MEXICO  S. DE R.L. DE C.V.  ...
2274    NUEVA WAL MART DE MEXICO S. DE R.L. DE C.V.   ...
2275                             SUPERANDA  S.A. DE C.V.
2277                ABARROTES VINOS Y LICORES CASA GUZMAN
2279        ABARROTES CASA MIKY Y/O PEDRO RIVEROS RAMIREZ
2280             ABARROTES VINOS Y LICORES LA REALIZADORA
2281                ABARROTES  VINOS Y LICORES LA ILUSION
2282    NUEVA WAL MART DE MEXICO  S. DE R. L. DE C. V....
2286                      ABARROTES VINOS Y LICORES DANIEL
Name: p.RAZON_SOCIAL_VISITADA, dtype: object
```

Mostramos el nivel de incumplimiento por estado (se muestra solamente una parte de la consulta porqu no cupo completa en una imagen)

```
Ingrese una opción >> 8
-----  Nivel de incumplimiento relativo de cada estado  -----
```

	t.ENTIDAD	INCUMPLIMIENTO RELATIVO
13	JALISCO	0.604925
26	TABASCO	0.171812
17	NAYARIT	0.153191
19	OAXACA	0.136126
22	QUINTANA ROO	0.123711
2	BAJA CALIFORNIA SUR	0.105031
30	YUCATAN	0.104213
18	NUEVO LEON	0.103937
0	AGUASCALIENTES	0.096330
1	BAJA CALIFORNIA	0.079858
10	GUANAJUATO	0.072209
21	QUERETARO	0.071293
5	CHIHUAHUA	0.061764
20	PUEBLA	0.060606
8	DISTRITO FEDERAL	0.057898
4	CHIAPAS	0.057818
27	TAMAULIPAS	0.056836
9	DURANGO	0.056109

Buscamos recomendaciones de productos para un usuario

```
Ingrese una opción >> 9
-----  Recomendaciones de productos  -----

Introduzca el ID del usuario para buscar recomendaciones: 3
Los productos que ha comprado este usuario son:
 0    ROPA
Name: p.TIPO_DE_PRODUCTO, dtype: object
Introduzca el tipo de producto sobre el cual quieres buscar recomendaciones APARATOS ELECTRICOS
```

	o.ID_USUARIO	o.ID_PRODUCTO	p.TIPO_DE_PRODUCTO	p.DESCRIPCION_DEL_PRODUCTO	p.RAZON_SOCIAL_VISITADA
2	8	1013	APARATOS ELECTRICOS	4 REGULADORES ELECTRICOS	PRO STAR
3	1	1014	APARATOS ELECTRICOS	4 TELEVISIONES	SANBORN HERMANOS S.A.

Mostramso los artículos más consumidos

```
Ingrese una opción >> 10
-----  Frecuencia de tipos de productos comprados por los usuarios  -----
```

	p.TIPO_DE_PRODUCTO	FRECUENCIA
0	PRODUCTOS ALIMENTICIOS	5
1	APARATOS ELECTRICOS	2
2	ROPA	1
3	TEQUILA REPOSADO	1
4	JOYERIA	1
5	FRIJOLES	1
6	MEDICAMENTOS	1

Por estado, mostramos la gama de artículos disponibles

```
Ingrese una opción >> 11
-----  Disponibilidad de productos de un estado  -----

Ingrese la entidad donde desea buscar: ZACATECAS
```

	p.TIPO_DE_PRODUCTO	FRECUENCIA
0	ATUN	488
1	TEQUILA REPOSADO	346
2	ZAPATOS	203
3	TEQUILA	198
4	PRODUCTOS ALIMENTICIOS	160
...	...	...
148	JUEGOS DE MESA	1
149	ARTICULOS DE PELUCHE	1
150	COMIDA	1
151	PINTURAS (MATERIALES PARA CONST)	1
152	VERDURAS FRESCAS	1

153 rows × 2 columns

Por estado, mostramos todas las tiendas disponibles

```
Ingrese una opción >> 12
-----  Disponibilidad de tienda de un estado  -----

Ingrese el estado en donde desea buscar: HIDALGO
```

	p.RAZON_SOCIAL_VISITADA
0	27 MICRAS INTERNATIONAL S.A. DE C.V.
1	ABARROTES SEMILLAS Y CHILES SECOS SIN NOMBRE
2	ABARROTES CARNES FRIAS DON QUIJOTE
3	ABARROTES LA FE
4	ABARROTES PAPELERIA Y REGALOS MARIFER
...	...
228	ZAPATERIA MEMO
229	ZAPATERIA PEPIN Y REGALOS ALKA
230	ZAPATERIA QUETZALLI
231	ZAPATERIA Z
232	ZAPATOS FINOS DE TULANCINGO

233 rows × 1 columns

Mostramos la cantidad de productos disponibles por cada razón social en cada estado

```
Ingrese una opción >> 13
-----  Número de productos ofrecidos por cada tienda de un estado  -----

Ingrese la entidad donde desea buscar: SAN LUIS POTOSI
```

	t.RAZON_SOCIAL_VISITADA	FRECUENCIA
0	CADENA COMERCIAL OXXO S.A. DE C.V.	950
1	TIENDAS CHEDRAUI S.A. DE C.V.	270
2	TIENDAS SORIANA S.A. DE C.V.	247
3	TIENDAS COMERCIAL MEXICANA S.A. DE C.V.	123
4	NUEVA WALMART DE MEXICO S. DE R.L. DE C.V.	102
...	...	...
134	STYLO	1
135	ZAPATERIAS FLEXI	1
136	ZAPATERIA JUVENA	1
137	ZAPATERIA LUCA	1
138	ZAPATERIA CASSANO	1

139 rows × 2 columns

Mostramos la cantidad de productos ofrecidos por razón social

```
Ingrese una opción >> 14
-----  Disponibilidad de cada producto que ofrece una tienda  -----

Ingrese la tienda donde desea buscar: BODEGA AURRERA
```

	p.TIPO_DE_PRODUCTO	FRECUENCIA
0	JUGUETES	14
1	ATUN	14
2	HUEVO	14
3	ARTICULOS DESECHABLES	7
4	ARTICULOS DE LIMPIEZA PERSONAL	7
5	ARTICULOS VACACIONALES	7
6	LECHE	7
7	PRODUCTOS LACTEOS	7

CONCLUSIONES

Todo lo que podemos hacer en Neo4j lo *podríamos* hacer en una Base de Datos Relacional. Sin embargo, una base de datos gráfica puede facilitar la expresión de cierto tipo de consultas. Además, con optimizaciones específicas, ciertas consultas pueden funcionar mejor. Es por eso que Neo4j está diseñada principalmente para consultas en las que queramos saber relaciones de relaciones de relaciones...; en las que este tipo de consultas podrían ser muy complejas en las Bases de Datos Relaciones.
Entonces, es recomendable usar Neo4j (Las Bases de Datos de tipo Grafo) en caso de que nuestra aplicación tenga relaciones complejas de muchos a muchos: a medida que la aplicación evoluciona, se agreguen nuevas relaciones; como lo comprobamos en esta práctica, pues los nodos usuario, orden, producto y tiendas se conectaban mediante conexiones; y las que en resumidas cuentas la podímos representar de la siguiente forma: Los **usuarios** *hacen* **ordenes** de **productos**, los cuales *se encuentran en* **tiendas**.

Consideramos que el punto más sobresaliente de esta práctica, y que de cierta forma es el que presenta la mayor ventaja de usar Neo4j, fue el de hacer recomendaciones de productos y establecimientos (tiendas), pues se buscaba analizar datos y relaciones interconectados, y en una Base de Datos Relacional requeriría hacer varios JOINS.

Por consiguiente, es importante mencionar algunas ventajas y desventajas de Neo4j:

VENTAJAS:

- No requiere JOINS complejos para recuperar datos (nodos) relacionados, ya que es muy fácil recuperar su nodo adyacente o detalles de la relación sin JOINS o índices.
- La sintaxis de las consultas es muy legible y similar a las BDR.
- El modelo de datos es más natural.Es compatible con pizarrones, pues puede usar el lenguaje de nodo, relaciones y propiedades para describir el dominio de la aplicación en lugar de usar modelos complejos.
- Cumple con las propiedades ACID y brinda soporte total para transacciones.

DESVENTAJAS:

- Tiene una limitación del número de nodos, relaciones y propiedades.
- Neo4j admite clústeres del tipo maestro-esclavo que pueden escalar linealmente las lecturas donde los esclavos pueden compartir la carga de lectura. En cuanto a la carga de escritura, solo la instancia maestra en el clúster puede manejarla. Otras instancias esclavas aún pueden recibir las solicitudes de escritura de

los clientes, pero estas solicitudes se enviarán al nodo maestro. Por lo tanto, no escala las escrituras muy bien y, en caso de cargas de escritura excepcionalmente altas, solo es posible el escalamiento vertical de la instancia maestra.

- No es compatible con Sharding.

Podemos concluir que Neo4j es adecuado para aplicaciones donde tenemos datos conectados y en los que deseamos ejecutar consultas complejas de relaciones de relaciones, y así obtener un alto rendimiento. Además, Neo4j es adecuado si tiene datos que se representarán de manera más natural utilizando el modelo gráfico, como datos de redes, semiestructurados o altamente conectados. Sin embargo, si tiene un modelo de datos simple que no requiere mucha unión o agregación, es posible que no obtenga ninguna mejora de rendimiento si usa Neo4j o la ventaja de rendimiento será muy pequeña, si no peor. Además, Neo4j tiene debilidades de escalabilidad relacionadas con escalar escrituras, por lo tanto, si se espera que su aplicación tenga rendimientos de escritura muy grandes, entonces Neo4j no es la solución para la aplicación.