# Bases de Datos No Estructuradas

UNAM— Semestre 2022-2

Integrantes:
Andrés Urbano Guillermo Gerardo
Avitúa Varela Fernando
Santa Rita Vizuet Fernando

## § Bases de Datos orientadas a grafos §

## Introducción

*Neo4j* es un software libre de Base de datos orientada a grafos, implementado en Java. Los desarrolladores describen a *Neo4j* como un motor de persistencia embebido, basado en disco, implementado en Java, completamente transaccional, que almacena datos estructurados en grafos en lugar de en tablas ("embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs rather than in tables"). La versión 1.0 de Neo4j fue lanzada en febrero de 2010.

En *Neo4j*, todo se almacena en forma de arista, nodo o atributo. Cada nodo y arista puede tener cualquier número de atributos y tanto los nodos como aristas se pueden etiquetar. Las etiquetas se pueden utilizar para limitar las búsquedas. A partir de la versión 2.0, se agregó la indexación a Cypher con la introducción de esquemas. Anteriormente, los índices se admitían por separado de Cypher.

## Diseño e implementación de la base de datos

Una de las virtudes de las bases de datos orientadas a grafos es que pueden ser diseñadas vía dibujo, donde podemos ver las entidades y las relaciones que mantienen estas. El siguiente esquema muestra, de forma general, la idea detrás del diseño de la base de datos.

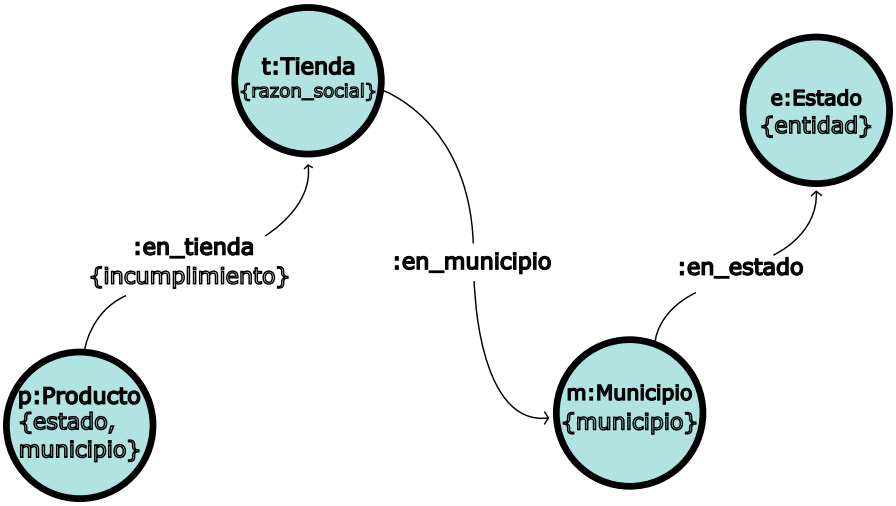


**Figure 1:** Diseño general de la base de datos.

El diseño de la gráfica de la figura 1 nos dice de forma natural la pertenencia entre cada una de las entidades. Existen atributos en el nodo de *Producto* relativos a la pertenencia del producto en cada uno de los estados y municipios del país; el agregar estos atributos ayuda a realizar consultas más sencillas, ayudando a satisfacer de mejor manera las necesidades del usuario final.

Dado el conjunto de datos a trabajar (PROFECO), cargamos este al un DataFrame de la librería Pandas y luego es exportado a la base de datos de *Neo4j*. Para ello, primero establecemos conexión con *Neo4j*:

```
# conexión con la base de datos
data_base_connection = GraphDatabase.driver(uri="bolt://localhost:7687",
                                            auth=("neo4j", "1234"))
session = data_base_connection.session()
```

Con una conexión exitosa, es posible ejecutar las intrucciones para cargar el conjunto de datos a Neo4j:

```
# valores del DataFrame como listas de listas
profeco_lista = profeco_df.values.tolist()

# construcción de lista de intrucciones para Neo4j
instrucciones_neo4j = []

for transaccion in profeco_lista[:]:
    instruccion=f'''
MERGE (e:Estado {{nombre_estado:"{transaccion[10]}"}})
MERGE (tp:TipoProcuto {{tipo: "{transaccion[5]}"}})
MERGE (t:Tienda {{razon_social:"{transaccion[7]}"}})
MERGE (m:Municipio {{nombre_municipio:"{transaccion[9]}"}})
MERGE (p:Producto {{nombre_producto:"{transaccion[6]}", estado:e.nombre_estado,
       municipio:m.nombre_municipio}})
MERGE (p)-[et:en_tienda {{incumplimiento:"{transaccion[11]}"}}]->(t)
MERGE (t)-[:en_municipio]->(m)
MERGE (m)-[:en_estado]->(e)
MERGE (p)-[:es_de_tipo]->(tp)
'''.replace('\n',' ')

    instrucciones_neo4j.append(instruccion)

# ejecutar instrucciones de instrucciones_neo4j
for i in notebook.tqdm(instrucciones_neo4j):
    session.run(i)
```

## Consultas

**Consulta 1:** Dado un estado y un producto, buscar lugares donde pueda encontrarlo.

Para realizar esta consulta basta encontrar todas las tiendas que están asociadas a un producto, cuidando que el estado y municipio estén conectados

```
def estado_producto(estado,producto):
    '''
    Regresa la tienda y municipio que tiene ese producto
    '''
    query=f'''
MATCH(m:Municipio)
```

```
MATCH(e:Estado {{nombre_estado:"{estado}"}})
MATCH(t:Tienda)
MATCH(p:Producto {{nombre_producto:"{producto}"
,estado:"{estado}",municipio:m.nombre_municipio}})
MATCH (p)-[:en_tienda]->(t)
MATCH (m)-[:en_estado]-> (e)
MATCH (t)-[:en_municipio]->(m)
RETURN t,m'''
    q_res=session.run(query)
    return q_res.data()
```

Por ejemplo, haciendo el llamado de la función con los siguientes valores:

```
estado='baja california'
producto='productos con informacion comercial'
```

obtenemos

```
Producto: productos con informacion comercial
--------------------------------------
Tienda:mercado y carniceria la fiesta
Municipio:tijuana
--------------------
Tienda:auto servicio ojeda
Municipio:tijuana
--------------------
Tienda:fruteria y dulceria huicho
Municipio:tijuana
--------------------
Tienda:dulceria arcoiris
Municipio:tijuana
```

**Figure 2:** Resultado de la consulta 1 para un producto

También lo podemos ver directamente en la interfaz gráfica de Neo4j.

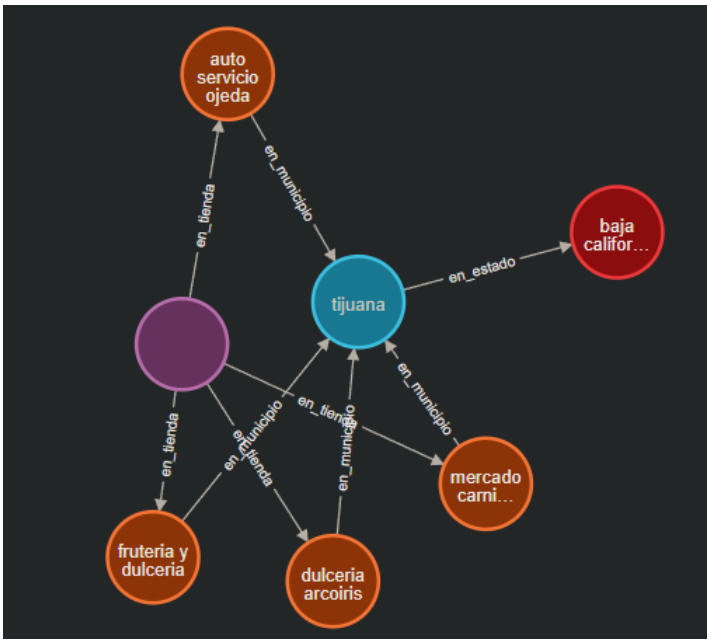


**Figure 3:** Resultado de la consulta 1.

**Consulta 2:** Dado un estado y una tienda, verificar si tiene algún incumplimiento con un producto.

El programa que resuelve esta consulta recibe como parámetros la tienda y estado. Si la salida es vacía, significa que en la tienda de dicho estado no hay ninguna clase de incumplimiento; de otro modo, serán listados los artículos que presentan dicho incumplimiento.

```
def falta_producto(tienda, estado):

    query = f"""
    MATCH (m:Municipio)
    MATCH (p:Producto {{estado:"{estado}", municipio:m.nombre_municipio}})
    MATCH (e:Estado {{nombre_estado:"{estado}"}})
    MATCH (t:Tienda {{razon_social:"{tienda}"}})
    MATCH (p) -[et:en_tienda]-> (t)
    WHERE NOT et.incumplimiento = "no se detecto incumplimiento"
    MATCH (t) -[:en_municipio]-> (m)
    MATCH (m) -[:en_estado]-> (e)
    RETURN p
    """.replace('\n',' ')

    return session.run(query).data()
```

Obsérvese que se aprovecha la existencia de los atributos en los nodos, lo cuál permite realizar consultas sencillas. Como ejemplo, veremos los incumplimiento que se encuentran en el *Hotel del Fresno de Fresnillo S.A. de C.V.* del estado de *Zacatecas*:

```
falta_producto('hotel del fresno de fresnillo s.a. de c.v.', 'zacatecas')
```

Resultado:

```
[{'p': {'estado': 'zacatecas',
   'municipio': 'fresnillo',
   'nombre_producto': 'botella de mezcal 1 lt.'}},
 {'p': {'estado': 'zacatecas',
   'municipio': 'fresnillo',
   'nombre_producto': 'botellas con mezcal 1lt.'}},
 {'p': {'estado': 'zacatecas',
   'municipio': 'fresnillo',
   'nombre_producto': 'botella de tequila reposado 100% de agave 950ml.'}},
 {'p': {'estado': 'zacatecas',
   'municipio': 'fresnillo',
   'nombre_producto': 'botella de tequila 950ml.'}}]
```

**Figure 4:** Resultado de la consulta 2.

**Consulta 3:** Dado un estado y un producto, buscar alternativas sin incumplimiento de ese producto o categoría.

Para la realización de esta consulta filtraremos a traves de la sentencia WHERE para obtener los productos alternativos sin inclumpliento.

```
def buscar_alternativas(estado, producto):
    query = f'''
    MATCH (p:Producto)-[et:en_tienda]->(t:Tienda)
    MATCH (e:Estado)
```

```
    MATCH (m:Municipio)
    MATCH (t)-->(m)-->(e)
    WHERE et.incumplimiento <> "no se detecto incumplimiento"
        AND e.nombre_estado = "{estado}" AND p.nombre_producto = "{producto}"
    RETURN p,t,m,e
    '''
    alternativa = session.run(query)
    return alternativa.data()
```

```
nombre_producto = 'cajacon reproductor de disco...'
alternativas = buscar_alternativas('mexico', nombre_producto)
print('Alternativas de lugares...')
for i, alternativa in enumerate(alternativas):
    print(f"Alternativa {i+1}: {alternativa['m']['nombre_municipio']}, \
                {alternativa['t']['razon_social']}")
```

Salida:

```
--Alternativas de lugares donde se puede encontrar sin incumplimiento el producto--
Alternativa 1: valle de chalco solidaridad, tiendas soriana s.a. de c.v.
Alternativa 2: cuautitlan izcalli, tiendas soriana s.a. de c.v.
Alternativa 3: naucalpan de juarez, tiendas soriana s.a. de c.v.
```

**Consulta 4:** Llevar un registro de las compras que ha hecho cada usuario y el lugar donde las ha hecho.

Para esta consulta se crea una función que permite agregar usuarios que compraron un producto. Se almacena la hora de la compra para distinguirla de haber una segunda compra del producto.

```
def registrar_compra(user,curp,tienda,producto,municipio,año,mes,dia):
  q_compra_usuario=f'''
  MERGE (p: Producto {{nombre_producto:"{producto}",municipio:"{municipio}"
  ,estado:"{estado}"}})
  MERGE (t: Tienda {{razon_social:"{tienda}"}})
  MERGE (m: Municipio {{nombre_municipio:"{municipio}"}})
  MERGE (p)-[:en_tienda]->(t)
  MERGE (t)-[:en_municipio]->(m)
  MERGE (u: Usuario {{CURP:"{curp}",nombre:"{user}"}})
  MERGE (u)-[cp:compro_producto
  {{hora:date({{year: {año}, month: {mes}, day: {dia}}})}}]->(p)

  RETURN u,p,t,m,cp.hora
  '''

  return session.run(q_compra_usuario).data()
```

Después usamos la siguiente función para regresar el historial de compra de un usuario

```
def historial_usuario(user,curp):

  q_historial=f'''
  MATCH (u: Usuario {{CURP:"{curp}",nombre:"{user}"}})
  MATCH (u)-[cp:compro_producto]->(p:Producto)
  RETURN u,p,cp.hora
  '''
  l=session.run(q_historial).data()

  municipio=[]
```

```
    producto=[]
    estado=[]
    hora=[]
    for li in l:
      municipio.append(li['p']['municipio'])
      estado.append(li['p']['estado'])
      producto.append(li['p']['nombre_producto'])
      hora.append(li['cp.hora'])

    return pd.DataFrame({'user':user,'estado':estado,'municipio':municipio,
                         'hora':hora,'producto': producto})
```

Después de hacer algunas inserciones usando la función *registrar_ compra* obtenemos la siguiente salida para el usuario *K. Reeves*

| | user | estado | municipio | hora | producto |
|---|---|---|---|---|---|
| 0 | K. Reeves | tamaulipas | reynosa | 2021-11-21 | atun aleta amarilla en aceite marca tuny cont.... |
| 1 | K. Reeves | chihuahua | chihuahua | 2019-07-23 | 25 lapices delineadores |
| 2 | K. Reeves | hidalgo | pachuca de soto | 2021-03-25 | palomitas para microondas bolsa con 85 gramos ... |
| 3 | K. Reeves | aguascalientes | aguascalientes | 2019-06-20 | manos libres 1 pieza color blanco marca flet m... |
| 4 | K. Reeves | mexico | tecamac | 2020-02-28 | tarro grande comercializadora bofian |

**Figure 5:** Historial de compras de K.Reeves

de esta manera podemos hacer la consulta, obtener el historial de compras y después buscar información como el patrón de compra de la persona.

Podemos ver el efecto que esto tuvo en la base de datos en la siguiente imagen:

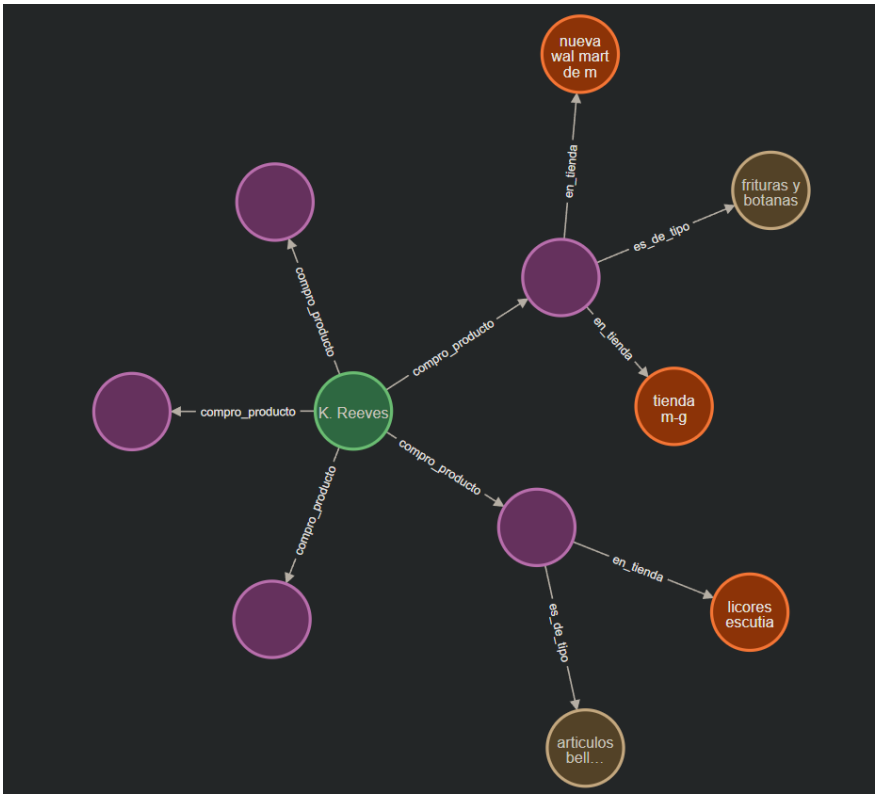


**Figure 6:** Muestra de la inserción de artículos comprados por *K. Reeves*. En la interfaz gráfica se desplegaron la tienda y el tipo de artículo de los dos artículos de la derecha para mostrar que las compras se unen al resto de la base.

**Consulta 5:** Encontrar los estados con mayor y menor incumplimiento relativo al número de tiendas que tiene.

La estrategia para resolver este problema será mediante la creación de dos consultas a la base de datos de Neo4j para obtener:

- Los productos y tiendas que presentan incumplimiento dado un estado.
- La cuenta del número total de tiendas en un estado.

El resultado de estas consultas es procesado para obtener la razón entre el número de tiendas que presentan incumplimiento y el número total de tiendas.

```
def incumplimiento_por_tienda(estado):

    query = f"""
    MATCH (e:Estado {{nombre_estado:"{estado}"}})
    MATCH (m:Municipio)
    MATCH (p:Producto {{estado:"{estado}", municipio:m.nombre_municipio}})
    MATCH (t:Tienda)
    MATCH (t) -[:en_municipio]-> (m)
    MATCH (m) -[:en_estado]-> (e)
    MATCH (p) -[et:en_tienda]-> (t)
    WHERE NOT et.incumplimiento = "no se detecto incumplimiento"
    RETURN p, t
    """

    return session.run(query).data()

def tiendas_por_estado(estado):

    query = f"""
    MATCH (e:Estado {{nombre_estado:"{estado}"}})
    MATCH (m:Municipio)
    MATCH (p:Producto {{estado:"{estado}", municipio:m.nombre_municipio}})
    MATCH (t:Tienda)
    MATCH (t) -[:en_municipio]-> (m)
    MATCH (m) -[:en_estado]-> (e)
    MATCH (p) -[et:en_tienda]-> (t)
    RETURN t
    """

    return session.run(query).data()


faltantes = {}
for estado in profeco_df.ENTIDAD.unique():

    tiendas = set()
    for dict in incumplimiento_por_tienda(f'{estado}'):
        tiendas.add(dict['t']['razon_social'])

    tiendas_totales = set()
    for dict in tiendas_por_estado(f'{estado}'):
        tiendas_totales.add(dict['t']['razon_social'])

    faltantes[f'{estado}'] = len(tiendas) / len(tiendas_totales)

# creamos un DataFrame con los resultados obtenidos
faltas_por_estado = pd.Series(faltantes).sort_values(ascending=False)
```

Graficando los resultados guardados en el DataFrame, se tiene la siguiente gráfica:

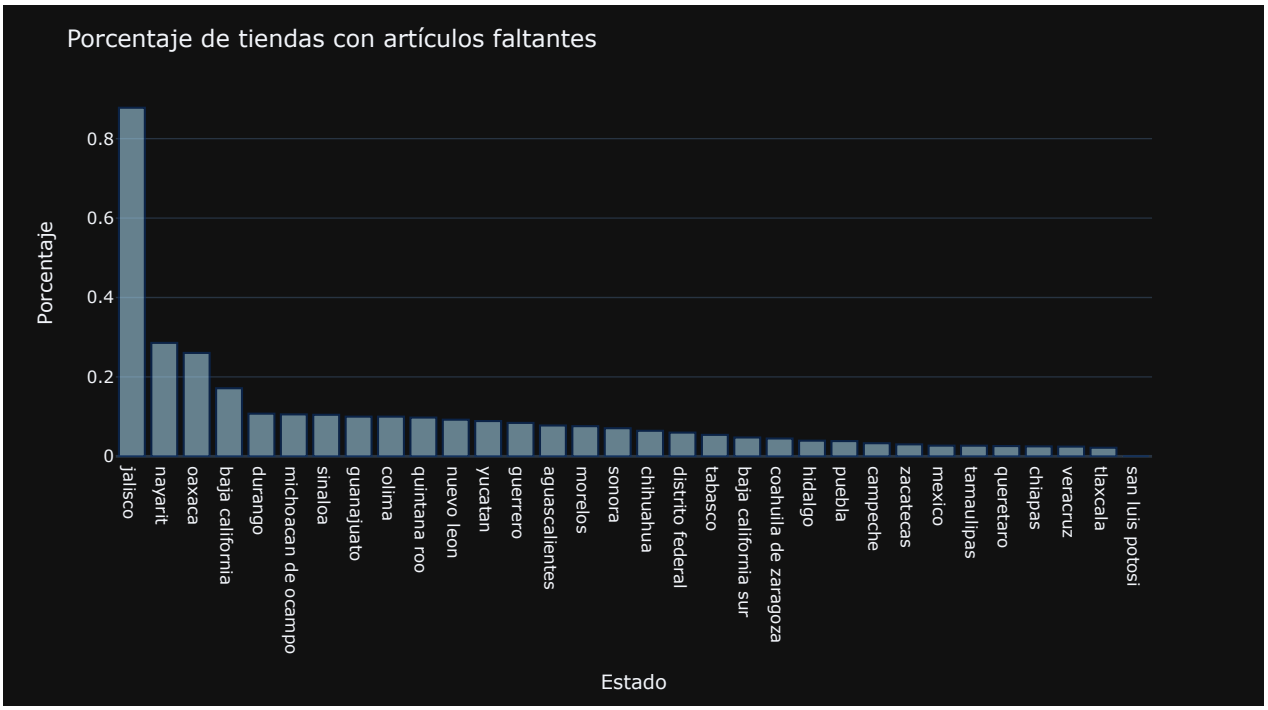


**Figure 7:** Resultado de la consulta 5.

Jalisco es el estado que la mayor de las irregularidades.

## Conclusiones

Las bases de datos orientadas a grafos ofrecen una forma visual y sencilla de visualización de las relaciones de diferentes entidades, con lo cuál es más sencillo el diseñar e implementar las bases de datos.

Estas bases de datos son particularmente buenas con las relaciones que existen entre diferentes objetos, lo cuál la convierte en una base de datos ideal para el manejo de bases de datos con redes colaborativas. La sintaxis y acciones que se pueden ejecutar son bastante similares a las ya vistas en otras bases de datos, por lo que migrar a esta es un paso natural para el entendimiento de estructuras de datos más versátiles.