Ingeniería en Sistemas Computacionales

Bases de Datos No Relacionales

Neo4j Lab 2 – Neo4j con Python

Marco Ricardo Cordero Hernández

Tlaquepaque, Jal., 19 de abril de 2023

Para la sorpresa de los miembros de algún conjunto vacío de personas, ya que se ha visto cómo interactuar con Neo4j, ya sea a través del navegador o aplicación nativa, es momento de manipular grafos de forma programática, lo cual, es un gran alivio.

Actualmente, existen métodos de importar archivos de valores separados por coma desde el cliente gráfico de escritorio de Neo4j, contando con dichos ficheros como objetos maleables a los cuales se puede acceder directamente para una rápida creación de nodos. Sin embargo, la opción descrita no deja de ser un punto de acceso manual y gráfico. En este laboratorio se busca la manera de mitigar esta interacción y automatizarla a través de Python, el lenguaje por excelencia a lo largo del curso.

Construcción de archivo de valores

En laboratorios anteriores, la forma de manejar los datos de entrada era de forma directa, cruda, con los datos deseados embebidos inmediatamente en los scripts de procesamiento. Para el propósito del presente desarrollo, se han tomado los datos del laboratorio anterior y se han modificado para extraerlos de las sentencias de inserción y relación, hacía un archivo externo, el cual contiene los mismos datos y sus relaciones de forma implícita; así, el manejo y creación de nuevos nodos y relaciones se realiza directamente desde Python.

El archivo generado es el siguiente.

```
- CONFERENCIAS -
name
AFC
NFC

- POSICIONES -
id|name|type
QB|Mariscal de campo|Backs
TE|Ala cerrada|Backs
WR|Receptor abierto|Backs
RB|Corredor|Backs
LB|Apoyador|Línea defensiva
CB|Esquinero|Línea defensiva
S|Profundo|Línea defensiva
K|Pateador|Especial
P|Despeje|Especial
C|Centro|Línea ofensiva
OG|Guardia ofensiva|Línea ofensiva

- EQUIPOS -
id|name|chmp|conf_id|div
BUF|Bills|0|AFC|Este
IND|Colts|2|AFC|Sur
MIA|Dolphins|2|AFC|Este
TB|Buccaneers|2|NFC|Sur
PIT|Steelers|6|AFC|Norte
KC|Chiefs|3|AFC|Oeste
PHI|Eagles|1|NFC|Este
BLT|Ravens|2|AFC|Norte
CIN|Bengals|0|AFC|Norte
```

```
CHI|Bears|1|AFC|Norte
GB|Packers|4|NFC|Norte
MIN|Vikings|0|NFC|Norte
LAC|Chargers|0|AFC|Oeste
SF|49ers|0|NFC|Oeste
CLV|Browns|0|AFC|Norte
LV|Raiders|3|AFC|Oeste
NE|Patriots|6|AFC|Este
DEN|Broncos|3|AFC|Oeste
SEA|Seahawks|1|NFC|Oeste

- JUGADORES -
name|bd|height|weight|number|pos_id|team_id|in_yr
Josh Allen|1996-05-21|1.96|108|17|QB|BUF|2018
Steffon Diggs|1993-11-29|1.83|88|14|WR|BUF|2020
John Unitas|1933-05-07|1.85|88|-1|QB|LAC|1973
Raheem Mostert|1992-04-09|1.78|89|31|RB|MIA|2022
Antonio Brown|1988-07-10|1.78|84|-1|WR|TB|2020
Patrick Mahomes|1995-09-17|1.88|102|15|QB|KC|2017
Travis Kelce|1989-10-05|1.96|113|87|TE|KC|2013
Jason Kelce|1987-11-05|1.91|128|62|C|PHI|2011
Justin Tucker|1989-11-21|1.85|82|9|K|BLT|2012
AJ Dillon|1998-05-02|1.83|112|28|RB|GB|2020

- EQUIPOS_PASADOS -
name|team_id|in_yr|out_yr
Steffon Diggs|MIN|2015|2019
John Unitas|PIT|1955|1955
John Unitas|IND|1956|1972
Raheem Mostert|SF|2016|2021
Raheem Mostert|CHI|2016|2016
Raheem Mostert|CLV|2015|2015
Raheem Mostert|BLT|2015|2015
Antonio Brown|PIT|2010|2018
Antonio Brown|LV|2019|2019
Antonio Brown|NE|2019|2019

- PARTIDOS -
winner|loser|tmp_year|spbwl
KC|PHI|2022|1
BUF|LAC|2022|0
CHI|SF|2022|0
PIT|CIN|2022|0
MIN|GB|2022|0
BLT|BUF|2018|0
KC|SF|2019|1
LAC|CIN|2021|1
SEA|DEN|2013|1
PHI|NE|2017|1

- ESTADOS -
id|name
NY|Nueva York
IN|Indianápolis
FL|Florida
PA|Pensilvania
KS|Kansas
MD|Maryland
OH|Ohio
IL|Illinois
MN|Minesota
MO|Misuri
CA|California
```

```
NV|Nevada
CO|Colorado
WA|Washington
WI|Wisconsin
MA|Massachusetts

- CASAS -
state_id|team_id|munc
NY|BUF|Búfalo
IN|IND|Indianápolis
FL|MIA|Miami
FL|TB|Tampa Bay
PA|PIT|Pittsburgh
MO|KC|Kansas City
NY|PHI|Filadelfia
MD|BLT|Baltimore
OH|CIN|Cincinnati
IL|CHI|Chicago
WI|GB|Green Bay
MN|MIN|Minesota
CA|LAC|Los Ángeles
CA|SF|San Francisco
OH|CLV|Cleveland
NV|LV|Las Vegas
MA|NE|Foxborough
CO|DEN|Denver
WA|SEA|Seattle
```

Código para generación del grafo

A manera introductoria y en forma de tutorial autodidacta, se proporcionó un código base para visualizar una forma de acercarse al driver de Neo4j a través de Python, esto, por supuesto, realizado a través de un ambiente virtual. El siguiente código es una propuesta funcional para la generación de grafos en formato sumamente general y flexible, el cual parte del repositorio inicial y ha sido puesto a prueba con el set de datos anterior.

```python
#!/usr/bin/env python3
# Modified main.py base by IS727272 - Cordero Hernández, Marco Ricardo
import os, time

from neo4j import GraphDatabase
from neo4j.exceptions import ClientError, ConstraintError

class NFLStats(object):

    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password))
        self.transactions = {
            'DELETE_ALL': 'MATCH (n) DETACH DELETE n',
            'CONFERENCIAS': 'CREATE (c:Conferencia {nombre: $name})',
            'POSICIONES': 'CREATE (p:Posicion {id: $id, nombre: $name, tipo: $type})',
            'EQUIPOS': ['CREATE (e:Equipo {id: $id, nombre: $name, campeonatos: toInteger($chmp)})',
                        'MATCH (e:Equipo {id: $id}), (c:Conferencia {nombre: $conf_id}) MERGE (e)-[r:PARTE_DE {division: $div}]-
>(c)'],
            'JUGADORES': ['CREATE (j:Jugador {nombre: $name, nacimiento: date($bd), estatura: toFloat($height), peso:
toFloat($weight), numero: toInteger($number)})',
                          'MATCH (j:Jugador {nombre: $name}), (p:Posicion {id: $pos_id}) MERGE (j)-[r:JUEGA_DE]->(p)',
                          'MATCH (j:Jugador {nombre: $name}), (e:Equipo {id: $team_id}) MERGE (j)-[r:PERTENECE_A {ingreso:
toInteger($in_yr)}]->(e)'],
            'EQUIPOS_PASADOS': 'MATCH (j:Jugador {nombre: $name}), (e:Equipo {id: $team_id}) MERGE (j)-[r:PERTENECIO_A {ingreso:
toInteger($in_yr), retiro: toInteger($out_yr)}]->(e)',
            'PARTIDOS': 'MATCH (w:Equipo {id: $winner}), (l:Equipo {id: $loser}) MERGE (w)-[r:GANO_A {temporada:
toInteger($tmp_year), superbowl: toBoolean(toInteger($spbwl))}]->(l)',
            'ESTADOS': 'CREATE (s:Estado {id: $id, nombre: $name})',
            'CASAS': 'MATCH (s:Estado {id: $state_id}), (e:Equipo {id: $team_id}) MERGE (s)-[r:CASA_DE {municipalidad: $munc}]->(e)'
        }

        with self.driver.session() as session: # Constraint creation as transactions
            session.execute_write(self._create_constraints)
```

```
    def close(self):
        self.driver.close()

    # Modified for own constraints and transaction oriented operation
    def _create_constraints(self, tx):
        # Ensure constraint deletion (ommited as it throws an exception)
        '''tx.run('DROP CONSTRAINT nombreJugador IF EXISTS')
        tx.run('DROP CONSTRAINT posID IF EXISTS')
        tx.run('DROP CONSTRAINT equipoID IF EXISTS')
        tx.run('DROP CONSTRAINT nombreConferencia IF EXISTS')
        tx.run('DROP CONSTRAINT estadoINIT IF EXISTS')'''

        # Create constraints
        tx.run('CREATE CONSTRAINT nombreJugador IF NOT EXISTS FOR (j:Jugador) REQUIRE j.nombre IS UNIQUE')
        tx.run('CREATE CONSTRAINT posID IF NOT EXISTS FOR (p:Posicion) REQUIRE p.id IS UNIQUE')
        tx.run('CREATE CONSTRAINT equipoID IF NOT EXISTS FOR (e:Equipo) REQUIRE e.id IS UNIQUE')
        tx.run('CREATE CONSTRAINT nombreConferencia IF NOT EXISTS FOR (c:Conferencia) REQUIRE c.nombre IS UNIQUE')
        tx.run('CREATE CONSTRAINT estadoINIT IF NOT EXISTS FOR (s:Estado) REQUIRE s.id IS UNIQUE')

    def _generic_write_tx(self, exec_str, **kwargs):
        def inner_tx(tx):
            tx.run(exec_str, **kwargs)

        with self.driver.session() as session:
            try:
                session.execute_write(inner_tx)
            except (ClientError, ConstraintError) as e:
                print(f'Error in transaction: {exec_str}\nParameters -> {kwargs}\nFull trace -> {e}')
                exit()

    def init(self, source):
        start = time.time() # Starting time of operations
        self._generic_write_tx(self.transactions['DELETE_ALL']) # Uncomment for initial data deletion
        with open(source, newline='') as f:
            contents = f.readlines()
            segment_type = ''
            curr_headers = []

            for line in contents:
                line = line.rstrip('\n')

                if (line.startswith('-')): # Start of new data segment
                    segment_type = line.strip('- ')
                    continue
                elif (not line): # End of data segment
                    segment_type = ''
                    curr_headers = []

                if (segment_type): # Data segment parsing
                    vals = line.split('|')

                    if (not curr_headers): # Get data segment headers
                        curr_headers = vals
                        continue

                    vals = {k:v for k,v in zip(curr_headers,vals)}

                    if (segment_type in self.transactions.keys()):
                        tx = self.transactions[segment_type]
                        if (type(tx) is list):
                            for ind_tx in tx:
                                self._generic_write_tx(ind_tx, **vals)
                        else:
                            self._generic_write_tx(self.transactions[segment_type], **vals)
                    else:
                        raise ValueError('Invalid node type. Incomplete operations made.')
            else:
                print(f'All operations completed successfully in {time.time() - start:.3f} seconds.')

if __name__ == "__main__":
    # Read connection env variables
    neo4j_uri = os.getenv('NEO4J_URI', 'bolt://localhost:7687')
    neo4j_user = os.getenv('NEO4J_USER', 'neo4j')
    neo4j_password = os.getenv('NEO4J_PASSWORD', 'neo4j_nosql')

    nfl = NFLStats(neo4j_uri, neo4j_user, neo4j_password)
    nfl.init("data/nfl.txt")

    nfl.close()
```

Quizás la estructura de la propuesta resulte intrigante a primera vista, pero, si se examina detalladamente y con un conocimiento superficial del funcionamiento de Python (a nivel de

lenguaje), se podrá percatar que la parte con más carga programática es la inicialización de modelo, que es donde precisamente se realiza el post procesado de los datos de entrada.

A las sentencias de inserción de nuevos nodos con datos correspondientes y sus relaciones entre ellos no se les da un método individual por clase, sino que se almacenan las cadenas de ejecución en un diccionario de posibles transacciones. ¿Por qué es que se realiza de esta forma? Por la parte epistemológica del todo, quizás la respuesta se encuentre en el poco ortodoxo e inherente deseo innato de la superación personal en forma de retos auto delegados; por la parte técnica, se hizo para poner a prueba las capacidades tanto de Neo4j como de Python, todo al mismo tiempo.
El método *_generic_write_tx* es una aberración que funciona de maravilla, haciendo uso de algún concepto de programación funcional y un poco de las características que el propio lenguaje ofrece. Así, se logró crear un método que recibe una cadena base que se ejecutará directamente en una comunicación con la base de grafos, en conjunto de palabras claves para insertar dinámicamente en la cadena mencionada. Fuera de esto, los únicos datos predefinidos son las restricciones propuestas desde el laboratorio pasado (ejecutadas en la inicialización del modelo) y las ya mencionadas transacciones que aluden a la creación del grafo.

El orden en que los datos han sido predispuestos en su archivo contenedor tiene relevancia, puesto que este modelo funciona de forma secuencial, esto con el fin de ahorrar la dinamicidad en el funcionamiento del constructor del grafo, algo que usualmente propicia errores en tiempo de ejecución. A lo que esto refiere es al orden de los bloques de datos, colocando primero a aquellos nodos que no dependen de otros o que sus relaciones apunten a ellos y no desde ellos, así, logrando una sincronización previa al momento de replicar el grafo.

Este último punto queda abierto a discusión y da pie a un área de mejora que finalmente otorgaría el máximo nivel de flexibilidad y abstracción necesaria para poder replicar cualquier tipo de grafo sin estar atado a sus reglas individuales. Esto, aunque parezca novedoso, presenta ciertos problemas de concurrencia, por lo cual quizás se debería optar por el enfoque inicial con una atadura a modelos rígidos y fuertemente definidos. Ahondar en estos tópicos excede el alcance deseado para este documento, por ende, por ahora se dejarán de lado junto con sus implicaciones.

Evidencias

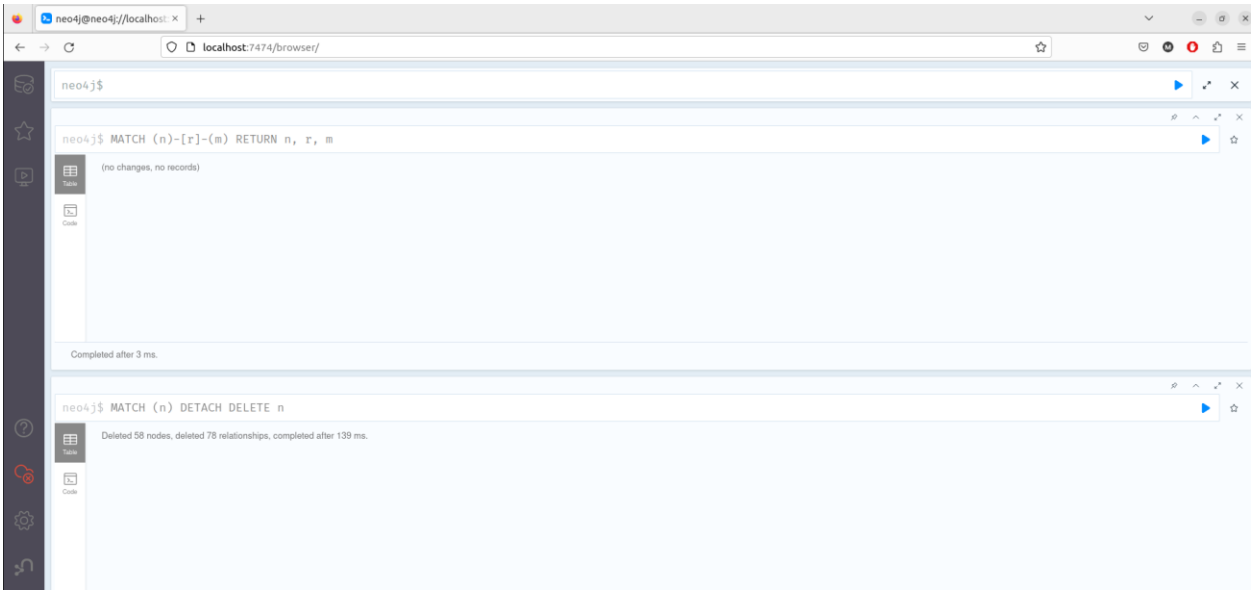

Ilustración 1: Limpieza inicial de la base y comprobación

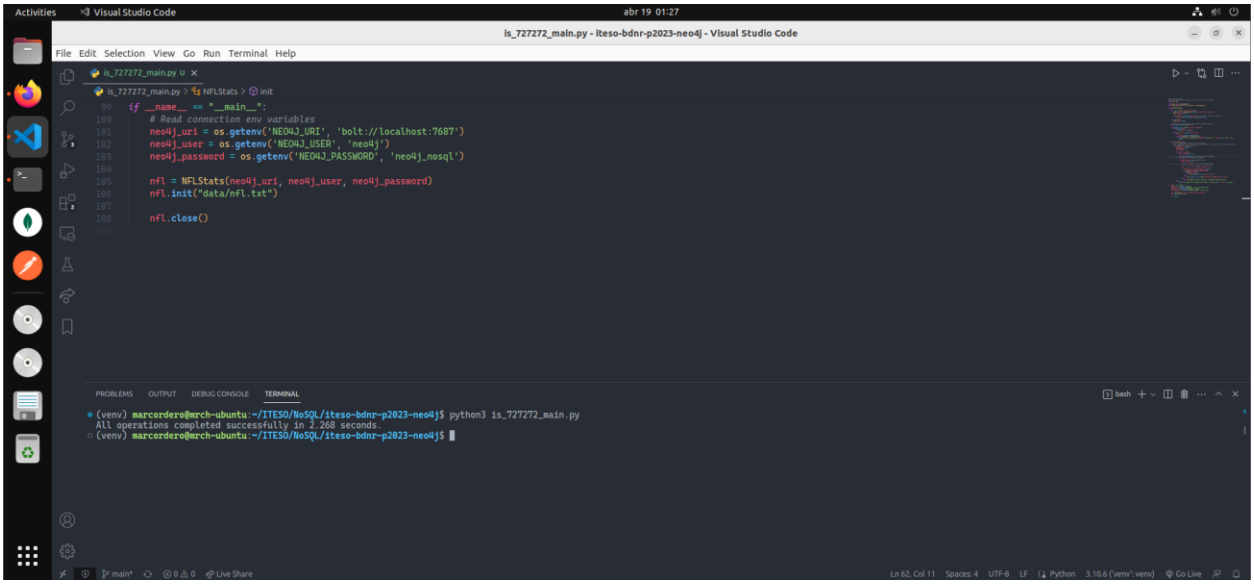

Ilustración 2: Ejecución de modelo

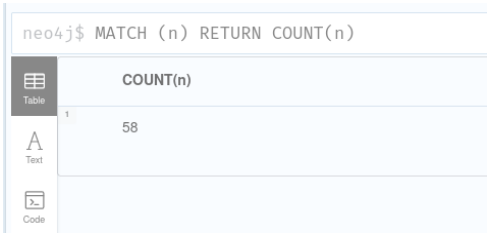

Ilustración 3: Conteo inicial de nodos resultantes

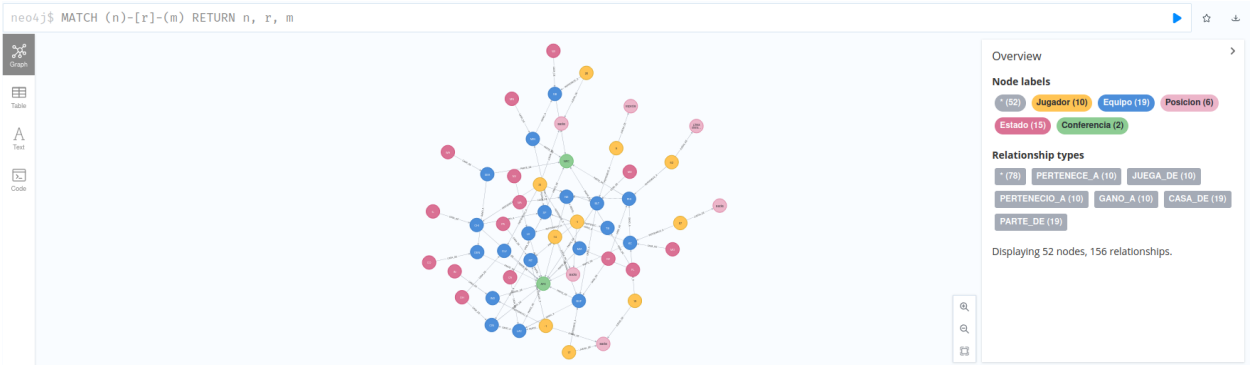

Ilustración 4: Visualización de nodos y relaciones resultantes

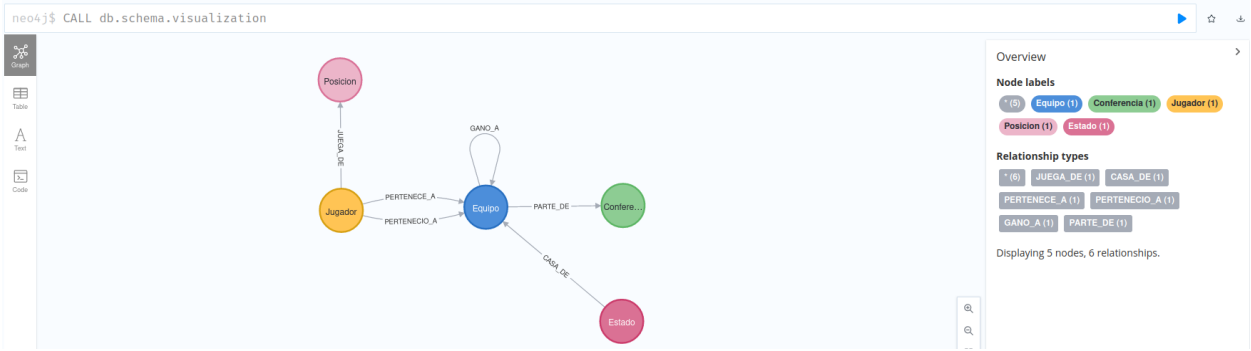

Ilustración 5: Visualización de grafo resultante

Conclusiones

El driver manejado en esta ocasión resulta sumamente interesante, no por la forma tan fluida de integrarse con Python ni por la amplia gama de funciones que ofrece, sino por la capacidad de resiliencia y atomicidad ante fallos en operaciones.

Puede parecer tedioso traducir el grafo original hacía un archivo separado, sin embargo, con una estructura bien definida, en el futuro la inserción de datos sería sumamente sencilla, a través de un proceso de una sola etapa, la cual es eficaz y eficiente.

Sin duda alguna, la utilidad que se le puede dar a este driver para dar soporte a almacenamiento de datos para aplicaciones web o nativas es evidente. No está de más recalcar también que la simpleza y efectividad que proporcionan las instrucciones, será de gran utilidad para el desarrollo posterior del proyecto que cada vez se avecina cada vez más.