Proyecto de Fin de Curso
BDNR 6

Anaclara Di Doménico - CI: 4775686-4
anaclara.di.domenico@fing.edu.uy
Fernando Rabago - CI: 4867039-8
fernando.rabago@fing.edu.uy
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay

I. Resumen

En este documento se podrá observar los pasos realizados para la construcción de una base de datos de grafos con el fin principal de poder modelar recetas junto con sus ingredientes, nutrientes y autores. El objetivo de la misma es poder mostrar recetas dependiendo de lo que el usuario quiera, ya sea cocinar con determinados ingredientes, conseguir recetas que respeten ciertos valores nutricionales, o recetas creadas por un autor específico.

II. Introducción

Este proyecto surgió de la idea de poder en un solo lugar conseguir recetas de comida que cuenten con determinados ingredientes o que respeten determinadas limitaciones alimenticias (no excedan tanta cantidad de carbohidratos, por ejemplo).

En un mundo donde cada vez es más difícil encontrar tiempo para cocinar, la idea de esta base es facilitar esto, ayudando a brindar recetas que se amolden a lo que uno tiene en su casa y a su vez poder controlar que nutrientes uno ingiere. A partir de la misma, se podría crear una aplicación para que cualquier persona pueda acceder a la información que la misma contiene, y hasta podría convertirse en una red social en donde compartir recetas y comentarlas entre los usuarios de la misma.

Por esto mismo, con el objetivo de aprender más sobre bases de datos de grafos y queriendo construir una, optamos por llevar a cabo este proyecto. Su funcionalidad principal es poder brindar una forma simplificada y rápida de buscar recetas, pudiendo filtrarlas por autor, ingredientes que uno quiere que contengan, cantidad de nutrientes que aporta, entre otros.

Para construir esta base utilizamos Neo4J Aura [1], donde primero construimos una base con unos pocos datos, y luego reproducimos lo mismo para una gran cantidad de datos. De esta forma logramos comprender como funciona Neo4J en el primer paso y construir una base de datos más robusta y con menos errores en el segundo paso.

III. Manipulación de datos

Lo primero que hicimos fue obtener los datos [2] de las recetas desde la plataforma web Kaggle[1], el cual es un sitio que reúne a la comunidad de ciencia de datos más grande y es respaldada por Walmart y Facebook. Para la manipulación de los mismos, realizamos varios scripts en Python 3.9 [3], primero para tener un espacio muestral más reducido, en donde decidimos reducir la cantidad de recetas en 99 para poder cargar una primera instancia de la base de datos de manera rápida, y así poder generarla de vuelta en el caso de obtener algún error. En el codigo se puede observar que obtenemos los datos de 'recipes_total.csv' que es el archivo que contiene todas las recetas obtenido desde el sitio de Kaggle anteriormente mencionado.

Listado 1 Reduccion de la cantidad de datos

```
import csv

with open("recipes_total.csv", "r") as input_file:
   reader = csv.reader(input_file)
   count = 0
   with open("data_part_3_10000.csv", "w") as output_file:
      writer = csv.writer(output_file)
      for row in reader:
         writer.writerow(row)
         count += 1

         if count == 10000:
            break
```

Luego de esto procedimos a obtener los ingredientes de cada receta por separado desde el archivo generado en el primer script llamado 'data_part_3_10000.csv', para luego poder crear la relación de manera correcta en Neo4j Aura.

Listado 2 Obtención de ingredientes

```
import csv

import pandas as pd

df = pd.read_csv("data_part_3_10000.csv")

new_df = pd.DataFrame(columns=["RecipeId", "Ingredient"])

for index, row in df.iterrows():
   recipe_id = row["RecipeId"]
   ingredients = row["RecipeIngredientParts"].strip("c()").replace('"', "").split(", ")

   recipe_df = pd.DataFrame(
      {"RecipeId": [recipe_id] * len(ingredients), "Ingredient": ingredients}
   )

   new_df = pd.concat([new_df, recipe_df], ignore_index=True)

new_df.to_csv("extracted_ingredients_10000.csv", index=False)
```

[1]https://www.kaggle.com/

Y por último también obtuvimos el nombre de cada nutriente y su valor, separando a cada uno en columnas en formato csv.

Listado 3 Obtencion de nutrientes

```
import csv

input_filename = "data_part_3_10000.csv"
output_filename = "extraceted_nutrients_10000.csv"
columns = [
    "Calories",
    "SaturatedFatContent",
    "SodiumContent",
    "CarbohydrateContent",
    "FiberContent",
    "SugarContent",
    "ProteinContent",
    "RecipeId",
]

with open(input_filename, "r") as input_file, open(
    output_filename, "w", newline=""
) as output_file:
    reader = csv.DictReader(input_file)
    rows = list(reader)

    writer = csv.DictWriter(output_file, fieldnames=columns)
    writer.writeheader()

    for row in rows[:-1]:
        output_row = {column: row[column] for column in columns}
        writer.writerow(output_row)

print("CSV file generated")
```

En cada uno de estos scripts se terminan generando archivos csv que fueron utilizados para poder cargarlos en AuraDB, y crear a partir de ellos los nodos y relaciones correspondientes.

IV. Implementación de la base de datos

En primer lugar, creamos una base de datos vacía para así poder agregarle nuestros nodos y crear las relaciones desde cero. Para ello, usamos la UI que Neo4J AuraDB provee para popular una base de datos vacía.

Para la creación de los nodos, cargamos los CSV generados en la etapa anterior, y creamos nodos de tipo *Receta*, *Author*, *Category*, *Nutrient* e *Ingredient*. Los nodos de tipo *Receta*, *Author* y *Category* los pudimos crear a partir del CSV original, pero para crear los nodos de tipo *Nutrient* e *Ingredient* tuvimos que generar nuevos CSV.

De igual forma, independiente de cuál CSV usamos para crear los distintos nodos, para todos ellos optamos por mapear directamente la información que tenía el CSV, populando los atributos de cada nodo con la información que este contenía, y manteniendo el nombre de cada atributo.

Una vez creados los nodos, pasamos a la parte de crear las relaciones entre ellos. Para ellos optamos por crearlas con consultas en Cypher, directamente desde la consola. A continuación mostramos las consultas usadas:

Listado 4 Creación de relación RECIPE - AUTHOR

```
MATCH (author:Author), (recipe:Recipe {AuthorId: author.AuthorId})
CREATE (recipe)-[:CREATED_BY]->(author)
```

Listado 5 Creación de relación RECIPE - INGREDIENTE

```
LOAD CSV WITH HEADERS FROM "https://drive.google.com/u/0/uc?id=1uS_QxgP6jAs3IeCnoWIG7mYyfMB98hbQ&export=download"
   AS row
MATCH (recipe:Recipe {RecipeId: toInteger(row.RecipeId)})
MERGE (i:Ingredient {Ingredient: row.Ingredient})
MERGE (recipe)-[:INCLUDES]->(i)
```

Listado 6 Creación de relación CATEGORY - RECIPE

```
MATCH (c:Category), (r:Recipe {RecipeCategory: c.RecipeCategory})
CREATE (c)-[:HAS]->(r)
```

Listado 7 Creación de relación RECIPE - NUTRIENT

```
LOAD CSV WITH HEADERS FROM "https://drive.google.com/u/0/uc?id=1vtIjaVBPsrepXYYYLSlYxugr6XzxdIBE&export=download"
   AS row
MATCH (n:Nutrient {Nutrient: "FiberContent"}),(recipe:Recipe {RecipeId: toInteger(row.RecipeId)})
MERGE (recipe)-[:CONTAINS {Amount: row.FiberContent}]->(n)
```

Esta última consulta tuvimos que hacerla para cada distinto tipo de nutriente (*Calories*, *SaturatedFatContent*, *SodiumContent*, *CarbohydrateContent*, *FiberContent*, *SugarContent*, *ProteinContent*) dado que, por como decidimos modelarlo, tenemos un nodo del tipo Nutrient que está relacionado a todas las recetas, indicando en la relación que proporción de ese nutriente tiene esa receta en particular.

Cabe destacar que en las consultas 2 y 4 tuvimos que utilizar CSVs para poder generar las relaciones, dado que la información de qué receta tenía cuál ingrediente, y en que proporción tiene cada nutriente cada receta, no estaba en los nodos, ya que optamos por dejarla afuera de

los CSVs usados en la primera parte. De esta forma, tuvimos que poner estos CSVs en un lugar accesible y público para Neo4J, y luego usarlos en estas consultas para generar las relaciones antes mencionadas.

Terminados estos pasos, obtuvimos la base de datos buscada. Los realizamos primero para un conjunto de datos de 99 recetas, y luego para un conjunto de 9.999 recetas.

V. Experimentación

Realizamos diferentes casos de prueba para probar la base de datos realizada. Primero que nada realizamos consultas simples como por ejemplo:

Listado 8 Ejemplo de consulta

```
MATCH (r:Recipe) RETURN r LIMIT 1000;
```

De esta forma logramos visualizar los nodos que tiene cada categoría de nodos, y probar que los datos se hayan cargado correctamente.

Dado que el objetivo de esta base es que uno pueda encontrar fácilmente recetas que cumplan con determinadas características, una consulta muy recurrente, dado nuestros casos de uso, es la de buscar recetas que contengan determinado ingrediente.

Listado 9 Ejemplo de consulta para un ingrediente

```
MATCH (r:Recipe)-[:INCLUDES]->(i:Ingredient{Ingredient: "butter"})
RETURN r
```

Listado 10 Ejemplo de consulta para varios ingredientes

```
MATCH (r:Recipe)-[:INCLUDES]->(:Ingredient {Ingredient: "butter"}),
 (r)-[:INCLUDES]->(:Ingredient {Ingredient: "sugar"}),
 (r)-[:INCLUDES]->(:Ingredient {Ingredient: "egg"})
RETURN r
```

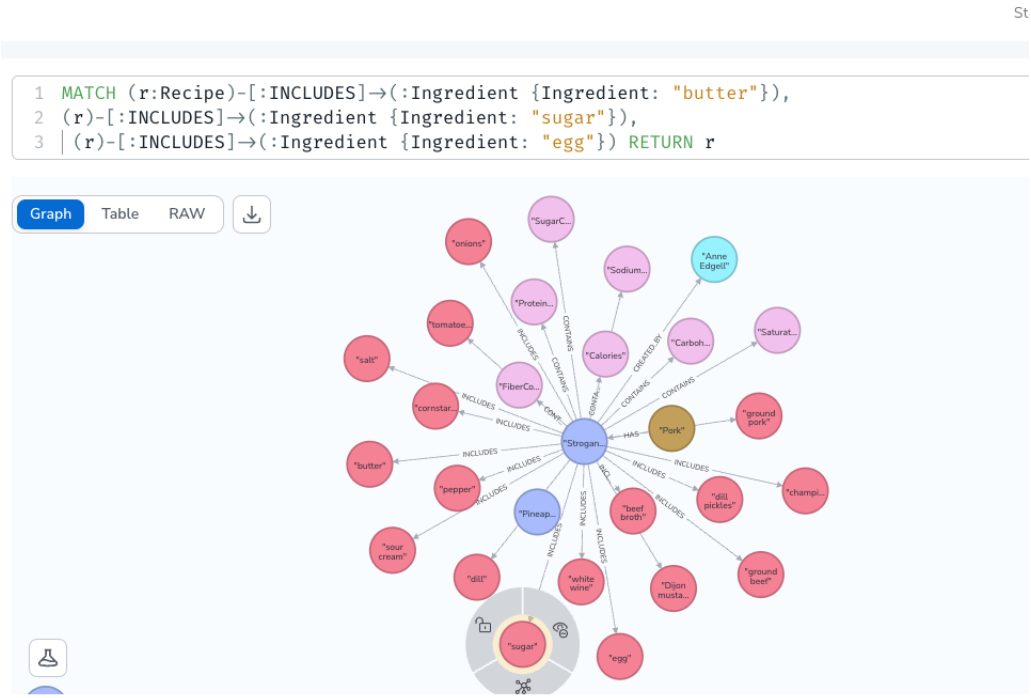

Figura 1: Parte del resultado de la consulta por varios ingredientes(sugar y egg)

También experimentamos buscando recetas por autor y filtrando recetas en función de su contenido nutricional.

Listado 11 Ejemplo de consulta por autor

```
MATCH (r:Recipe)-[:CREATED_BY]->(a:Author {AuthorName: "Doreen Randal"} )
RETURN r
```

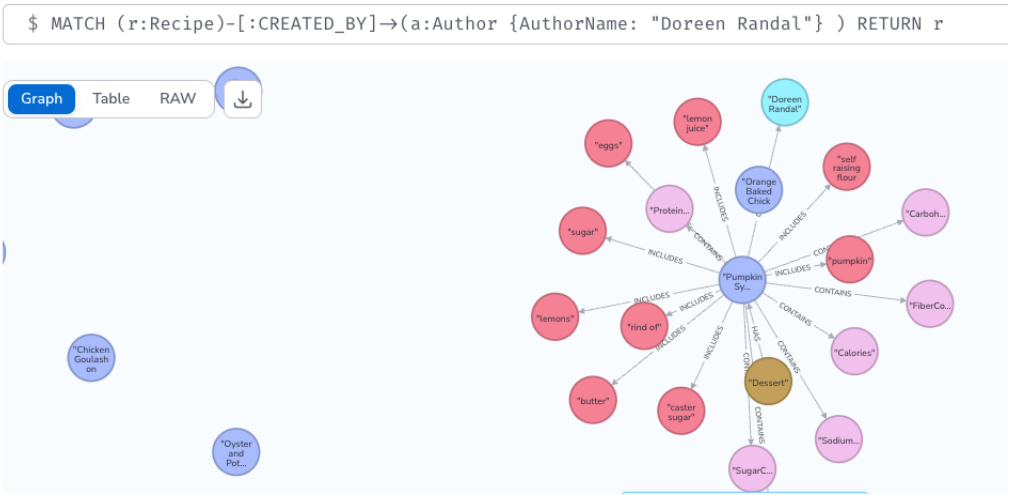

Figura 2: Parte del resultado de la consulta por autor

Listado 12 Ejemplo de consulta por valor nutricional

```
MATCH (r:Recipe)-[c:CONTAINS]->(:Nutrient {Nutrients:"SugarContent"})
WHERE (toFloat(c.Amount)<3)
RETURN r
```

Como se puede notar al correr estas consultas en la base, se logra llegar al resultado de forma rápida para cualquiera de ellas, ya sea para una base con 99 recetas o una con 9999.

VI. Conclusiones y trabajo futuro

Pudimos conseguir la base de datos que nos propusimos en un principio, y logramos construirla tanto para un conjunto reducido de datos como para uno bastante más extenso. Al haber realizado la misma base de datos, pero con distinta cantidad de datos, pudimos notar que con un juego de datos reducido, tanto las importaciones como las consultas, son más rápidas que cuando incrementamos el tamaño de los datos. Dada las limitaciones que tiene el plan gratuito de Neo4J AuraDB, la máxima cantidad de recetas con la que pudimos construir esta base fueron 9999 recetas, por lo que una posible continuación a futuro de este trabajo podría ser construir esta base con un juego de datos mayor, tratando de encontrar otra plataforma donde esto si se podría hacer, o haciéndola localmente. Pensando en otro trabajo a futuro, se podría realizar una aplicación móvil o web que utilice esta base de datos, y así poder mostrar de forma más amigable al usuario las diferentes recetas, y sus nutrientes e ingredientes. También se podría realizar una red social donde cada usuario publique sus recetas y de esa manera poder formar una comunidad. Por último, podemos notar que no se presentaron grandes dificultades a lo largo de la realización de este proyecto, siendo la más grande la manipulación de los datos, para poder separarlos y unirlos luego en la base de datos de grafos.

VII. CREDENCIALES

Dejamos expuestas las credenciales para poder conectarse a la base de datos creada en AuraDB[2].

Listado 13 Credenciales para AuraDB

```
NEO4J=URI=neo4j+s://6b6476cd.databases.neo4j.io
NEO4J=USERNAME=neo4j
NEO4J=PASSWORD=x588OFWrH9CYKG5DVrnd_CWxGxTeePYfIPGisNj64Lc
AURA=INSTANCEID=6b6476cd
AURA=INSTANCENAME=Instance01
```

También dejamos las credenciales de la primera base de prueba que hicimos, la cual construimos con 99 recetas como prueba de concepto.

Listado 14 Credenciales para AuraDB de base con solo 99 recetas

```
NEO4J_URI=neo4j+s://9933e984.databases.neo4j.io
NEO4J_USERNAME=neo4j
NEO4J_PASSWORD=pgtT7XDDk8EWJpeg9GyN61MIw3tB3zUmtkE-hphyPDg
AURA_INSTANCEID=9933e984
AURA_INSTANCENAME=Instance01
```

Al tener varios problemas con gitlab decidimos crear el repositorio en github. Dejamos el enlace del repositorio para que se puedan observar los diferentes scripts en Python utilizados para la manipulación de datos que igualmente mencionamos en el documento.

Listado 15 Enlace del repositorio en GiHub

```
https://github.com/AnaclaraDiDo/BDNR6-food
```

[2]https://neo4j.com/cloud/platform/aura-graph-database/

REFERENCIAS

[1] *Documentacion de AuraDB*. https://neo4j.com/docs/aura/.
[2] *Datos utilizados*. https://www.kaggle.com/datasets/irkaal/foodcom-recipes-and-reviews.
[3] *Documentacion de Python 3.9*. https://docs.python.org/es/3.9/index.html.