Ciencia de la Computación

Base de Datos II

Docente Yessenia Deysi Yari Ramos

Proyecto final

Entregado el 11/07/2023

Ana Cecilia Concha Castro
Angel Gabriel Villagomez Iquira
Fernando Jair Peralta Bustamante

Semestre V

2023-1

"Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

_____ _____ _____

PROYECTO FINAL BASE DE DATOS II

Unir mongoDB y Neo4J

INTRODUCCIÓN:

El proyecto tiene como objetivo principal la integración de dos potentes sistemas de bases de datos, MongoDB y Neo4j, para lograr que las consultas realizadas en MongoDB se puedan visualizar en forma de nodos. La combinación de estas dos bases de datos ofrecen un enfoque innovador para la gestión y visualización de datos, permitiendo una representación gráfica y relacionada de la información almacenada.

MongoDB, una base de datos NoSQL altamente escalable y flexible, se utiliza para almacenar grandes volúmenes de datos de manera eficiente. Por otro lado, Neo4j, una base de datos orientada a grafos, se especializa en modelar y consultar relaciones complejas entre los datos, representando los nodos y las conexiones entre ellos.

Al unir MongoDB y Neo4j, se obtiene una poderosa solución que combina la capacidad de almacenamiento y consulta de MongoDB con la capacidad de representación y navegación de datos en forma de nodos de Neo4j. Esto permite realizar consultas enriquecidas, explorar relaciones entre entidades y visualizar los datos de manera intuitiva y comprensible

En esta documentación oficial, se proporcionará una guía detallada de cómo se realizó la unión entre las base de datos MongoDB y Neo4j. Se explicará paso a paso el

proceso de creación de documentos en MongoDB utilizando la librería **pymongo**, así como la creación de nodos y relaciones en Neo4j utilizando la librería **py2neo**.

DATASET

El dataset escogido es Anime Dataset: A comprehensive collection of anime information, más específicamente el archivo anime-filtered.csv el cual cuenta con 14,952 filas y 25 diferentes columnas de las cuales hemos usado 21 debido a motivos que explicaremos más adelante, las columnas originales son:

1. **anime_id**: Identifica de forma única cada programa de anime en el conjunto de datos.

2. **Name**: Proporciona el título del programa de anime, tanto en inglés como en japonés.

3. **Score**: Indica la puntuación promedio del programa de anime.

4. **English name**: Muestra el nombre del programa de anime en inglés, si está disponible.

5. **Japanese name**: Muestra el nombre del programa de anime en japonés.

6. **Synopsis**: Ofrece un resumen breve de la trama o argumento de cada anime.

7. **Type**: Categoriza el anime como una serie de televisión, película u otro formato.

8. **Episodes**: Especifica el número total de episodios para cada programa de anime.

9. **Aired**: Muestra el rango de fechas en que se emitió el anime.

10. **Premiered**: Indica la temporada específica y el año de estreno del anime.

11. **Producers**: Enumera las compañías involucradas en la producción del anime.

12. **Licensors**: Muestra las compañías que tienen los derechos de distribución del anime.

13. **Studios**: Muestra los estudios de animación responsables de crear el anime.

14. **Source**: Indica el medio del cual se adaptó el anime, como manga u original.

15. **Duration**: Especifica la duración de cada episodio.

16. **Rating**: Proporciona la clasificación objetivo de la audiencia para el anime.

17. **Ranked**: Clasifica el anime según su popularidad o las calificaciones de los usuarios.

18. **Popularity**: Representa el rango de popularidad del anime.

19. **Members**: Indica el número de miembros que tienen el anime en su lista.

20. **Favorites**: Muestra el número de usuarios que han marcado el anime como favorito.

21. **Watching**: Representa el número de usuarios que están viendo actualmente el anime.

22. **Completed**: Indica el número de usuarios que han completado el anime.

23. **On-Hold**: Muestra el número de usuarios que han puesto el anime en espera.

24. **Dropped**: Representa el número de usuarios que han abandonado el anime.

De el total de columnas listado anteriormente, hemos tomado la decisión de descartar las columnas

- anime_id
- English name
- Japanese name
- Sinopsis

El motivo del descarte de anime_id es debido a que MongoDB ya agrega un identificador único a cada uno de los documentos al momento de su creación, mientras que tanto “English name” y “Japanese name” fueron descartadas a causa de redundancia en la información pues ya se cuenta con una columna name, por otro lado “Sinopsis” fue descartado debido a su baja utilidad para este proyecto en específico, sin embargo, consideramos que en otro proyecto o como base de datos, es un atributo a tener en cuenta.

CREACIÓN DE LOS DOCUMENTOS EN MONGODB

La creación de la base de datos, colección y documentos fue realizada desde python haciendo uso de la librería pymongo.

Una vez nuestro dataset ha sido “limpiado” y ya conocemos la información que deseamos almacenar en cada uno de nuestros documentos, podemos proseguir con la creación de los mismos.

La forma de explicar la creación de los documentos será a través de pasos guia de como se estructuró el código

1. **Importación de las librerías a utilizar**

```
import csv

from pymongo import MongoClient

from datetime import datetime, timedelta
```

csv: Se da para poder tener acceso al archivo de tipo csv que es nuestro dataset y poder leer la información almacenada en él.

pymongo: librería principal para el funcionamiento de todo el código de donde importamos MongoClient que nos permitirá hacer la conexión con la base de datos y crear tanto la colección como los documentos de manera sencilla desde el código.

datetime: librería utilizada para la creación de un atributo en específico, “Aired” el cual explicaremos más adelante

2. **Conexión con la base de datos de Mongodb**

```
from config import USERNAME, PASSWORD

client = MongoClient(f"mongodb+srv://{USERNAME}:{PASSWORD}@cluster0.mjbi0oc.mongodb.net/Health_2023?retryWrites=true&w=majority")

db = client["PROYECTO"]

collection = db["uwu"]
```

Procedemos a conectarnos a la base de datos de Mongodb, sobre todo al cluster que está cargado en la nube de manera que todo aquel que tenga acceso a ese cluster pueda acceder a la información y trabajar con ella.

Lo siguiente que hacemos es declarar tanto la base de datos como la colección a utilizar, la ventaja de hacerlo de esta manera es que en caso no tengamos ninguna base de datos creada con ese nombre ni esa colección, automáticamente se creará y podremos trabajar en ella desde el cluster configurado.

Cabe destacar que la línea de importación de config de los datos de USERNAME y PASSWORD son básicamente las credenciales de acceso a Mongodb pero por motivos de seguridad se trabajarán de manera incógnita, en caso se desee implementar este código, se debe cambiar ambas por las credenciales particulares del usuario.

3. **Accedemos a nuestro archivo csv**

```
csv_file = r"anime-filtered.csv"
```

De manera sencilla, lo único que hacemos es acceder a la dirección de nuestro csv que contiene toda la información

4. **Leemos el archivo csv para crear los documentos**

```
# Leemos el archivo para crear los documentos

with open(csv_file, "r", encoding="utf-8") as file:

    reader = csv.DictReader(file)

    for row in reader:

            #Todos los atributos que deseamos almacenar en forma de
array les hacemos un split y los separamos al momento de encontrar
un ", "

        genres = row["Genres"].split(", ")
```

```
        producers = row["Producers"].split(", ")

        licensors = row["Licensors"].split(", ")



        #En el caso de aired, lo almacenamos en su formato de string
y creamos dos variables para almacenar el start y end

        aired_str = row["Aired"]

        aired_start = None

        aired_end = None



        if aired_str.endswith(" to ?"):

            formato_fecha = "%b %d, %Y"  # Formato de la cadena de
fecha

            try: #Intentamos crear los dos atributos donde extraemos
la cantidad del string necesario en el formato y lo combinamos

                aired_start = datetime.strptime(aired_str[:-6],
formato_fecha).date()

                    aired_start = datetime.combine(aired_start,
datetime.min.time())

                aired_end = None

            except ValueError:

                aired_start = None

                aired_end = None
```

```
        elif "to" in aired_str: #En caso si tengamos fecha de inicio
y final

            formato_fecha = "%b %d, %Y"  # Formato de la cadena de
fecha

            aired_dates = aired_str.split(" to ") #Los separamos

            try: #Creamos ambos valores

                    aired_start = datetime.strptime(aired_dates[0],
formato_fecha).date()

                        aired_start = datetime.combine(aired_start,
datetime.min.time())

                    aired_end = datetime.strptime(aired_dates[1],
formato_fecha).date()

                            aired_end = datetime.combine(aired_end,
datetime.min.time())

            except ValueError:

                aired_start = None

                aired_end = None

        elif aired_str != "?":

            try:

                aired_start = datetime.strptime(aired_str, "%b %d,
%Y").date()

                        aired_start = datetime.combine(aired_start,
datetime.min.time())
```

```
            except ValueError:

                aired_start = None

        else:

            aired_start = None

            aired_end = None

#En caso de ranked almacenamos solo si es un numero entero, caso contrario lo marcamos como None

         ranked = int(row["Ranked"]) if row["Ranked"].isdigit() else None



        #Creamos nuestro documento con todos sus atributos

        document = {

            "Name": row["Name"],

            "Score": float(row["Score"]),

            "Genres": genres,

            "Type": row["Type"],

            "Episodes": row["Episodes"],

                "Aired": {"start": aired_start, "end": aired_end} if aired_end else aired_start,

            "Premiered": row["Premiered"],

            "Producers": producers,
```

```
            "Licensors": licensors,
            "Studios": row["Studios"],
            "Source": row["Source"],
            "Duration": row["Duration"],
            "Rating": row["Rating"],
            "Ranked": ranked,
            "Popularity": int(row["Popularity"]),
            "Members": int(row["Members"]),
            "Favorites": int(row["Favorites"]),
            "Watching": int(row["Watching"]),
            "Completed": int(row["Completed"]),
            "On-Hold": int(row["On-Hold"]),
            "Dropped": int(row["Dropped"]),
        }
        #Insertamos los documentos a la coleccion
        collection.insert_one(document)

print(f"Se llenaron todos los {collection.count_documents({})} documentos")
```

Una vez finalizado con este código, ya tendríamos creado tanto nuestra base de datos como la colección a utilizar y además de eso los documentos, en este caso un total de 14952 documentos, cada uno con 21 atributos diferentes.

CONSULTAS Y CREACIÓN DE LOS NODOS EN NEO4J

Una vez terminado con la creación de los documentos en Mongodb procedemos a realizar las consultas también en python para de esta forma con el resultado obtenido poder proseguir con la creación de los nodos en Neo4j de manera óptima. Tengamos en cuenta las siguientes observaciones.

- Al momento de realizar una consulta nueva en Mongodb lo mejor que podemos hacer es limpiar la totalidad de nodos creados hasta ese momento en Neo4j para solo observar el resultado de nuestra consulta
- La creación de los nodos se da de manera manual debido a la necesidad de tener los resultados en la consulta de Mongodb.
- En caso se desee tener la base de datos Neo4j cargada con toda la información y solo transcribir la consulta de formato Mongodb a Neo4j lo mejor sería hacerlo a través de una IA para facilitar el trabajo pues son formatos totalmente diferentes y se tendría que hacer doble trabajo creando los nodos y sabiendo cual de ellos es el que se invoca a consulta.
- Algunos de los pasos son repetidos por lo que no se entrará mucho en detalle respecto a lo que se está realizando en esa porción de código.

El código que se realizó para llevar a cabo las consultas en Mongodb fue escrito en python haciendo uso de la librería pymongo y la conexión con Neo4j fue realizada a través de la librería py2neo.

La forma de explicar como está escrito el codigo sera en pasos guia para una fácil comprensión del trabajo realizado

1. Importamos las librerías a utilizar

```
from pymongo import MongoClient

from config import USERNAME, PASSWORD

from py2neo import Graph, Node, Relationship

import itertools
```

py2neo: De esta librería importamos Graph, Node y Relationship pues son las 3 cosas que vamos a utilizar para la creación de nuestro grafo, las detallaremos mejor más adelante.

itertools: Nos servirá para que nuestro bucle solo haga una cantidad límite de iteraciones al momento de ejecutar la creación de los nodos y relaciones, esto es necesario para bajar de manera drástica el tiempo de ejecución de nuestro programa, crear mas de 14000 nodos es mucha carga de trabajo para una vista que solo permite 300 nodos en pantalla a la vez.

2. Conectamos con Mongodb

```
client = MongoClient(f"mongodb+srv://{USERNAME}:{PASSWORD}@cluster0.mjbi0oc.mongodb.net/Health_2023?retryWrites=true&w=majority")

db=client["PROYECTO"]
```

```
collection=db["uwu"]
```

3. Revisamos el código para consultas .find en Mongodb

```
query = {

    "Genres":{"$eq":"Action"},

    "Score": {"$gt": 8},

    "Source": {"$ne": "Original"}

}

results = collection.find(query).sort("Popularity", 1).limit(5)
```

query: Una variable sencilla que nos permite almacenar el texto de como se haría la consulta .find() en mongo para de esta manera ordenar lo que estamos filtrando

results: Variable donde almacenaremos los resultados de toda la consulta para poder trabajar luego con ella en forma de iterador

4. Limitamos la cantidad de iteraciones que va a hacer nuestro bucle for

```
limited_results=itertools.islice(results,30)
```

limited_results: Variable para almacenar únicamente la cantidad de documentos que decidamos, esto es útil sobre todo cuando no contamos con un .limit en nuestras consultas pues también es una forma de truncar la cantidad de repeticiones que haremos en el bucle

5. Conexión con Neo4j

```
graph = Graph("bolt://localhost:7687", auth=("neo4j", "12345678"))

Delete_query="MATCH (n) DETACH DELETE n"

graph.run(Delete_query)

print("Se han borrado todos los nodos")
```

Graph: Variable que almacena todo el texto necesario para trabajar con la base de datos de Neo4j sin necesidad de repetirlo continuamente, compuesto de dos partes, en la primera indica la dirección y puerto en la que está corriendo nuestra base de datos mientras que en la segunda se requieren las credenciales para acceder a la base de datos de neo4j y poder manipularla, “neo4j” es el usuario por default de la base de datos, sin embargo, “12345678” es la contraseña que se le coloca a la base de datos y debería ser de carácter privado de preferencia.

Delete_query: Variable que nos permite almacenar la línea de comandos a correr en la terminal de Neo4j para realizar el borrado de todos los nodos cargados en ese momento en la base de datos facilitando creación de nuevos nodos permitiendo la observación de únicamente los nodos resultados de nuestra consulta

6. Con un bucle for creamos todos los nodos y relaciones, por motivos didácticos, dividiremos este paso en muchos más pequeños para facilitar la comprensión del código.

   6.1. Creamos el nodo anime

```
for result in limited_results:

    aired=result["Aired"]

    anime_node = Node("Anime",

                      Name=result["Name"],
```

```
                        Score=result["Score"],
                        Episodes=result["Episodes"],
                            Aired_start=aired["start"] if aired
and "start" in aired else None,
                            Aired_end=aired["end"] if aired and
"end" in aired else None,
                        Premiered=result["Premiered"],
                        Source=result["Source"],
                        Duration=result["Duration"],
                        Rating=result["Rating"],
                        Ranked=result["Ranked"],
                        Popularity=result["Popularity"],
                        Members=result["Members"],
                        Favorites=result["Favorites"],
                        Watching=result["Watching"],
                        Completed=result["Completed"],
                        On_Hold=result["On-Hold"],
                        Dropped=result["Dropped"],
    )
    graph.create(anime_node)
```

aired: Variable sencilla que nos permite acceder al array almacenado en “Aired” y de esta forma acceder más rápido a cada uno de sus componentes

anime_node: Declaramos un objeto de tipo Node que va a almacenar toda la estructura que va a contener nuestros nodos de anime, todos los atributos son extraídos de los resultados utilizando la forma:

result[“atributo _a_ extraer”]

En este nodo tenemos dos atributos que destacan por su particularidad

- Aired_start
- Aired_end

Para estos dos atributos extraemos los datos encapsulados en la variable aired y declaramos un if para cada uno donde en caso no tengan ningún dato almacenado se le coloca el valor None a ese atributo dado que la librería no reconoce el none generado al momento de la creación de nuestros documentos.

Por último con el comando de graph.create(anime_node) nos permite crear todos los nodos generados hasta el momento con esa estructura

6.2 Creación de nodos de género, la complejidad de este nodo recae al momento de la creación de un nodo partiendo de un array donde podemos tener diferentes valores en un mismo anime, sin embargo, varios animes pueden pertenecer a un o varios mismos géneros.

```
genres_array=result["Genres"]
```

```
for genre in genres_array:
```

```
        Genre_Node = Node("Genre", Name=genre)

        graph.merge(Genre_Node, "Genre", "Name")

        relationship = Relationship(anime_node, "genero:", Genre_Node)

        graph.create(relationship)
```

genres_array: variable donde almacenamos todos los géneros posibles dentro de un documento

Con un bucle for recorremos todos los objetos del array y se crea un nodo donde el nombre del nodo pasa a ser el genero que esta almacenado, hacemos .merge() en lugar de un .create() debido a que el .merge() no permite repetición, lo cual es útil al momento de tener varios animes con los mismos nodos

6.3 Repetimos el mismo proceso para el nodo productores y licensors.

```
    #Creacion del nodo productores
```

```
        producers_array=result["Producers"]

        for producer in producers_array:

            Producer_Node = Node("Producers", Name=producer)

            graph.merge(Producer_Node, "Producers", "Name")


relacion3=Relationship(anime_node,"Productor:",Producer_Node)

            graph.create(relacion3)

        #Creacion del nodo Licensors
```

```
        Licensors_array=result["Licensors"]

        for licensor in Licensors_array:

            Licensor_node = Node("Licensors", Name=licensor)

            graph.merge(Licensor_node, "Licensors", "Name")

            relacion4 = Relationship(anime_node, "Licencia:",
Licensor_node)

            graph.create(relacion4)
```

6.4 Creamos el nodo tipo y nodo estudio

```
    #Creacion del nodo Tipo

    Type_node = Node("Type", Name=result["Type"])

    graph.merge(Type_node, "Type", "Name")

    #Creacion del nodo Studios

    Studios_node = Node("Studios", Name=result["Studios"])

    graph.merge(Studios_node, "Studios", "Name")
```

6.5 Creamos las relaciones entre cada uno de estos nodos con el nodo anime

```
    #Relacion entre anime y tipo

    relacion2 = Relationship(anime_node,"Tipo:",Type_node)

    graph.create(relacion2)
```

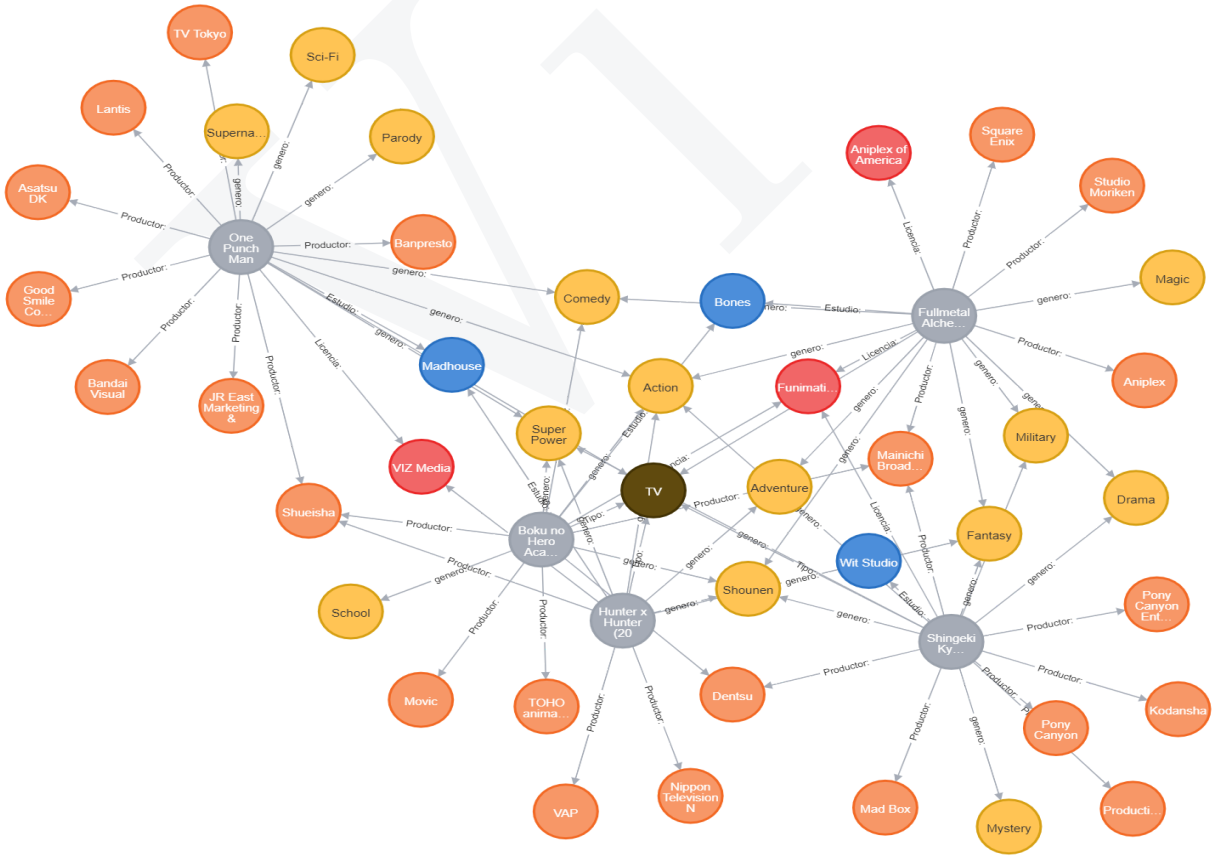
```
#relacion entre anime y estudios

relacion5=Relationship(anime_node,"Estudio:",Studios_node)

graph.create(relacion5)
```

Con esto hemos terminado con la creación tanto de nuestros nodos como de nuestros documentos.

Si todo lo hemos creado bien entonces solo debemos ejecutar el siguiente comando en Neo4j.

MATCH (n)-[r]->(m) RETURN n,r,m

Y se debería de observar el grafo con todas las relaciones creadas anteriormente



CONCLUSIÓN

Al concluir el proyecto de integración entre MongoDB y Neo4j para la gestión y visualización de datos de anime, se pueden obtener las siguientes conclusiones:

1. La combinación de MongoDB y Neo4j proporciona una solución poderosa y versátil para el manejo de datos relacionales y consultas enriquecidas. MongoDB ofrece un almacenamiento eficiente y flexible para grandes volúmenes de datos, mientras que Neo4j permite modelar y consultar relaciones complejas entre los datos en forma de nodos y grafos.
2. La integración de MongoDB y Neo4j permite aprovechar las fortalezas de ambas bases de datos. MongoDB se utiliza para almacenar y consultar datos detallados de los animes, como su puntaje, géneros, estudios de producción, entre otros. Por otro lado, Neo4j se utiliza para representar y explorar las relaciones entre los animes, así como las conexiones entre géneros, productores y licenciatarios.

3. La visualización de datos en forma de nodos y relaciones proporcionada por Neo4j permite una comprensión más intuitiva de la estructura y conexiones entre los animes. Esto facilita la exploración y análisis de los datos, permitiendo descubrir patrones y tendencias interesantes en la industria del anime.

4. La automatización del proceso de creación de documentos en MongoDB y nodos en Neo4j utilizando las librerías pymongo y py2neo, respectivamente, agiliza el flujo de trabajo y permite una gestión eficiente de los datos. Esto facilita la actualización y expansión del conjunto de datos, así como la realización de consultas y análisis continuos.

5. La correcta configuración de los atributos y relaciones en los nodos y grafos es fundamental para garantizar una representación precisa de los datos. Es importante establecer los atributos relevantes y establecer relaciones adecuadas entre los nodos para obtener resultados significativos en las consultas y visualizaciones.

En resumen, el proyecto de integración entre MongoDB y Neo4j ha demostrado ser una solución efectiva para la gestión y visualización de datos de anime. La combinación de estas bases de datos ofrece un enfoque innovador para el análisis y exploración de datos, permitiendo descubrir insights valiosos en la industria del anime.