Universidad Católica
San Pablo

Ciencia de la Computación

Base de Datos II

Proyecto Final: Mongo + Neo4j

Leon Zarate, Ariana
Carpio Peña, Josue
Rodriguez Pinto, Anthony
Zeballos Cartagena, Diego

Semestre V

2023-1

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

ÍNDICE

INTRODUCCIÓN:

En el presente trabajo, realizaremos el anàlisis y la representaciòn de una Base de Datos, la cual inicialmente estará en formato .csv, para posteriormente subirla a MongoDB donde podremos administrar y realizar consultas correspondientes; para finalmente estas consultas, por medio de un conector, verlas reflejadas con forma de grafo, es decir, nodos y relaciones en Neo4J.

Para este trabajo se trató de buscar una base de datos con aproximado de 10,000 datos; los cuales deben ser coherentes para poder realizar relaciones entre datos; las cuales se verán reflejadas al momento de crear nodos y graficarlos en Neo4J.

Adicionalmente, se buscaron varias opciones para poder realizar la conexión entre dichas aplicaciones, obteniéndose como método más conveniente, realizarla por medio de Python, a través de sus librerías pymongo y py2neo.

BASE DE DATOS:

Inicialmente, debemos encontrar una Base de Datos adecuada para poder realizar todo el trabajo, tanto en MongoDB, como en Neo4J, para lo cual dicha Base de Datos debe cumplir con los siguiente requisitos:

- Tener como mínimo 10,000 datos.
- Formato .csv o .json.
- Dichos datos deben ser de diferentes tipos, los cuales posean la capacidad de ser relacionados entre sí.
- Cada colección debe poseer por lo menos algún dato clave, el cual no se repita a lo largo de la data restante.
- No tener ningún problema de formato, ya que esto nos daría más problemas al momento de subir el archivo a mongo.

Finalmente, luego de mucho tiempo de busqueda, se logró encontrar una Base de Datos en el sitio web Kaggle, que cumplía la mayoría de Datos; la cual se puede visualizar en el siguiente link: https://www.kaggle.com/datasets/manishkumar7432698/airline-passangers-booking-data?resource=download.

Dicha Base de Datos consta de la información de reservas de vuelos para los feriados a lo largo del año.

Sin embargo, esta Base de datos no cumplía con nuestros requisitos 3 y 4, es por ello que recurrimos a la idea de agregar Data a nuestro archivo csv, para así tener una base de datos completa. Para realizar esta tarea, se utilizó el siguiente código de generación de Datos.

```python
from faker import Faker
import random

fake = Faker()

data = []

for _ in range(10000):
    booking_id = fake.random_int(min=1, max=999999)
    passenger_id = fake.random_int(min=1, max=999999)
    num_passengers = random.randint(1, 5)
    sales_channel = random.choice(["Online", "Offline"])
    trip_type = random.choice(["One-way", "Round-trip"])
    purchase_lead = random.randint(1, 60)
    length_of_stay = random.randint(1, 10)
    flight_hour = fake.time(pattern="%H:%M")
    flight_day = fake.date_between(start_date='-30d', end_date='+30d')
    route = fake.random_element(["New York - London", "Paris - Rome", "Tokyo - Sydney"])
    booking_origin = fake.random_element(["JFK", "CDG", "NRT"])
    wants_extra_baggage = random.choice(["Yes", "No"])
    wants_preferred_seat = random.choice(["Yes", "No"])
    wants_in_flight_meals = random.choice(["Yes", "No"])
    flight_duration = random.uniform(1, 12)
    booking_complete = random.randint(0, 1)
    name = fake.name()
    age = random.randint(18, 80)
    gender = random.choice(["Male", "Female"])
    nationality = fake.country()
    email = fake.email()
    country_id = fake.random_int(min=1, max=10)

    data.append([booking_id, passenger_id, num_passengers, sales_channel, trip_type, purchase_lead,
                 length_of_stay, flight_hour, flight_day, route, booking_origin, wants_extra_baggage,
                 wants_preferred_seat, wants_in_flight_meals, flight_duration, booking_complete, name,
                 age, gender, nationality, email, country_id])
```

Donde agregamos parámetros, los cuales puedan completar la información que requerimos, tales como información puntual de cada pasajero y reserva: nombres, correo, gènero, etc.

```python
import csv

header = ["booking_id", "passenger_id", "num_passengers", "sales_channel", "trip_type", "purchase_lead",
          "length_of_stay", "flight_hour", "flight_day", "route", "booking_origin", "wants_extra_baggage",
          "wants_preferred_seat", "wants_in_flight_meals", "flight_duration", "booking_complete", "name",
          "age", "gender", "nationality", "email", "country_id"]

with open('Passanger_booking_data.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(header)
    writer.writerows(data)
```

Finalmente, la data escogida para nuestro proyecto consta de un total de 22 atributos, los cuales fueron compilados en el archivo “Passanger_booking_data .csv”; teniendo este la siguiente informaciòn

1. **booking_id**: Es el identificador único asociado a la reserva o reserva de vuelo. Se utiliza para identificar de manera exclusiva cada reserva en el sistema.

2. **passenger_id:** Es el identificador único asociado a un pasajero en particular. Puede haber varios pasajeros en una reserva, por lo que este ID se utiliza para distinguir a cada pasajero.

3. **num_passengers:** Es el número total de pasajeros incluidos en la reserva. Indica la cantidad de personas que viajarán juntas en el vuelo.

4. **sales_channel:** Es el canal o medio a través del cual se realizó la venta de la reserva. Puede ser una agencia de viajes, un sitio web de reservas, un mostrador de aeropuerto, entre otros.

5. **trip_type:** Indica el tipo de viaje asociado a la reserva. Puede ser "ida" para un solo trayecto, "ida y vuelta" para un viaje de ida y regreso, o "multidestino" para viajes con múltiples paradas.

6. **purchase_lead:** Es el tiempo, generalmente en días, que pasó desde la búsqueda o consulta de vuelo hasta la compra o reserva real.

7. **length_of_stay:** Es la duración total de la estadía o estancia del pasajero en el destino, medida en días.

8. **flight_hour:** Es la hora del día en que se realiza el vuelo. Puede ser una representación numérica o una etiqueta de tiempo para indicar la hora de salida o llegada.

9. **flight_day:** Es el día del vuelo, generalmente representado como una fecha.

10. **route:** Es la ruta o trayecto del vuelo, es decir, los aeropuertos de origen y destino.

11. **booking_origin:** Indica el origen de la reserva o la ubicación desde donde se realizó la reserva. Puede ser una ciudad, un país o un código de ubicación.

12. **wants_extra_baggage:** Es un valor booleano que indica si el pasajero desea añadir equipaje adicional a su reserva.

13. **wants_preferred_seat:** Es un valor booleano que indica si el pasajero desea seleccionar un asiento preferido en el avión.

14. **wants_in_flight_meals:** Es un valor booleano que indica si el pasajero desea solicitar comidas durante el vuelo.

15. **flight_duration:** Es la duración total del vuelo, generalmente medida en horas y minutos.

16. **booking_complete:** Es un valor booleano que indica si la reserva se ha completado o finalizado.

17. **name:** Es el nombre del pasajero.

18. **age:** Es la edad del pasajero.

19. **gender**: Es el género del pasajero.

20. **Nationality**: Es la nacionalidad del pasajero.

21. **email**: Es la dirección de correo electrónico asociada al pasajero.

22. **country_id**: Es el identificador único del país del pasajero.

En el siguiente punto explicaremos cómo se utilizó la data y cómo se almacenó en MongoDB.

INICIALIZACIÓN DE DATA EN MONGODB:

Para la inicialización de los datos en mongoDB se utilizó la librería pymongo para crear la base de datos, colección y los datos que incluyen esta.

La creación de documentos se dio de la siguiente manera:

1. Importamos las librerías que utilizaremos para este punto. En este caso usamos csv para que python pueda tener acceso al archivo de este tipo, ya que nuestra data está almacenada en un archivo csv. También hacemos uso de la librería pymongo para el funcionamiento del código con lo que vendría a ser el servidor de MongoDB a través de MongoClient y así de esta manera sencilla conectarla con nuestro código.

2. Para terminar la conexión con la base de datos en MongoDB utilizamos el comando de MongoClient el cual nos pide nuestra dirección al localhost:27017, luego le pasamos el nombre de la base de datos y la colección que crearemos

```
import csv
from pymongo import MongoClient

MONGO_URL = "mongodb://localhost:27017"
DB_NAME = "proyecto"
COLLECTION_NAME = "proyecto"
COUNTRIES_COLLECTION_NAME = "countries"
PASSENGERS_COLLECTION_NAME = "passangers"
CSV_FILE = "/Users/josuecarpio/Downloads/bd2 2/Passanger_booking_data .csv"
```

```
client = MongoClient(MONGO_URL)
db = client[DB_NAME]
collection = db[COLLECTION_NAME]
countries_collection = db[COUNTRIES_COLLECTION_NAME]
passangers = db[PASSENGERS_COLLECTION_NAME]
```

   En nuestro caso se utilizan tres colecciones para disminuir la redundancia de datos, además que de esta manera nos permitió realizar otras consultas más adelante.

3. Utilizamos ciertas funciones para crear las colecciones con la data correspondiente:

```python
def insert_passenger_documents(passengers, collection):
    with open(CSV_FILE, "r") as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            passenger_id = int(row['passenger_id'])
            booking_ids = passengers.get(passenger_id, [])
            # Obtener los datos adicionales del pasajero del CSV
            passenger_data = {
                "name": row['name'],
                "age": int(row['age']),
                "gender": row['gender'],
                "nationality": row['nationality'],
                "email": row['email'],
                "country_id": row['country_id']
            }
            document = {
                "passenger_id": passenger_id,
                "booking_ids": booking_ids,
                **passenger_data
            }
            collection.insert_one(document)
```

- En la función insert_passenger_documents, parecida a insert documents que explicaremos más adelante, lo que realizamos es pasar la data a la colección passengers. El extra que realizamos en este caso es que expandimos la data en passenger_data con "**".

```python
def create_passenger_list(file):
    passengers = {}
    with open(file, "r") as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            passenger_id = int(row['passenger_id'])
            booking_id = int(row['booking_id'])
            if passenger_id in passengers:
                passengers[passenger_id].append(booking_id)
            else:
                passengers[passenger_id] = [booking_id]
    return passengers
```

- En la función create_passeger_list lo que realizamos es crear la lista de los pasajeros, la cual se extrae el passenger_id y booking_id. Se verifica si el passenger_id se encuentra en la colección de passengers, en caso exista lo que realizamos es agregar el passenger_id a la lista existente de booking_id. En caso contrario, creamos una entrada al diccionario utilizando el booling_id.

```python
def insert_documents(passengers, collection):
    with open(CSV_FILE, "r") as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            booking_id = int(row['booking_id'])
            passenger_id = int(row['passenger_id'])
            document = {
                "booking_id": booking_id,
                "passenger_id": passenger_id,
                "num_passengers": int(row['num_passengers']),
                "sales_channel": row['sales_channel'],
                "trip_type": row['trip_type'],
                "purchase_lead": row['purchase_lead'],
                "length_of_stay": int(row['length_of_stay']),
                "flight_hour": row['flight_hour'],
                "flight_day": row['flight_day'],
                "route": row['route'],
                "booking_origin": row['booking_origin'],
                "wants_extra_baggage": bool(row['wants_extra_baggage']),
                "want_preferred_seat": row['wants_preferred_seat'],
                "wants_in_flight_meals": bool(row['wants_in_flight_meals']),
                "flight_duration": float(row['flight_duration']),
                "booking_complete": int(row['booking_complete'])
            }
            collection.insert_one(document)
```

- En la función insert_documents lo que realizamos es incluir todos los documentos que tenemos, respetando el tipo de dato que se añadirá a cada columna de la colección.

```python
# Insertar los documentos de países en la colección de países
with open(CSV_FILE, "r") as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        document = {
            "country_id": row['country_id'],
            "country": row['nationality']
        }
        if not countries_collection.find_one(document):
            countries_collection.insert_one(document)
```

- En esta función lo que realizamos es insertar los documentos de países a la colección del mismo nombre.

CONEXIÓN, CONSULTAS Y CREACIÓN DE NODOS EN NEO4J

Una vez tengamos todos los datos cargados en MongoDB es momento de realizar las consultas y reflejarlas gráficamente en Neo4J

Para realizar esto utilizaremos el còdgio explicado acontinuaciòn:

1. Primeramente debemos importar ciertas herramientas y librerías que nos ayudarán a realizar esto, las cuales son “itertools”, “pymongo” y py2neo.

```
import itertools
from pymongo import MongoClient
from py2neo import Graph, Node, Relationship
```

2. Seguidamente estableceremos la conexión con MongoDB y Neo4J, para lo cual debemos inicializar nuestras credenciales y declarar cuales son las colecciones o bases de datos a utilizar.

```
# Establecer la conexion con MongoDB
client = MongoClient('mongodb://localhost:27017')
db = client["proyecto"]
collection = db["proyecto"]
passengers = db["passangers"]
countries = db["countries"]

# Establecer la conexion con Neo4j
neo4j_graph = Graph("bolt://localhost:7687", auth=("neo4j", "12345678"))
```

3. El siguiente paso consta de realizar las consultas necesarias para filtrar la data, en este caso solo utilizaremos la consulta .find(), la cual nos devolverá todos los datos de las 3 colecciones mencionadas anteriormente.

```
# Obtener todos los elementos de la coleccion de reservas en MongoDB
results = collection.find()

# Obtener todos los elementos de la coleccion de pasajeros en MongoDB
passenger_results = passengers.find()

# Obtener todos los elementos de la coleccion de paises en MongoDB
countries_results = countries.find()
```

4. Adicionalmente, agregò las siguientes líneas de código, que nos permitirán saber con cuantos datos contamos en cada colecciòn, los cuales seràn impresos al finalizar esta operaciòn.

```
# Obtener el numero de documentos en la coleccion de reservas
booking_document_count = collection.count_documents({})

# Obtener el numero de documentos en la coleccion de pasajeros
passenger_document_count = passengers.count_documents({})

# Obtener el numero de paises en la coleccion de paises
countries_document_count = countries.count_documents({})

# Imprimir el numero de documentos de reservas, pasajeros y paises
print(f"La coleccion de reservas tiene {booking_document_count} documentos.")
print(f"La coleccion de pasajeros tiene {passenger_document_count} documentos.")
print(f"La coleccion de paises tiene {countries_document_count} documentos.")
```

5. Luego, tenemos una parte importante de nuestro código, ya que al tener 10000 datos, esto puede generar problemas al momento de correr los archivos, por ellos se añadieron unos limitadores de datos, los cuales nos señalan la cantidad máxima de datos que se procesaran.

```
a_results=itertools.islice(passenger_results,50)
b_results=itertools.islice(results,50)
c_results=itertools.islice(countries_results,50)
```

6. Ahora pasaremos a crear nodos para la colecciòn Country, la cual contiene diferentes paìses presentes en la Base de Datos, los cuales contendrán un id.

```
# Crear nodos de paises en Neo4j
for country in c_results:
    country_node = Node("Countries",
                        country_id=country["country_id"],
                        country=country["country"])
    neo4j_graph.create(country_node)
```

7. También crearemos los nodos de la colecciòn pasajeros, los cuales contendrán toda la informaciòn detallada de todos los pasajeros.

```
# Crear nodos de pasajeros en Neo4j y establecer relaciones con los paises
for passenger in a_results:
    passenger_node = Node("Passenger",
                          passenger_id=passenger["passenger_id"],
                          name=passenger["name"],
                          age=passenger["age"],
                          gender=passenger["gender"],
                          nationality=passenger["nationality"],
                          email=passenger["email"],
                          country_id=passenger["country_id"])
    neo4j_graph.create(passenger_node)
```

8. Luego haremos la conexión con la colección Countries, tomando como parámetro de comparación, el country_id, y creando así la relación “COUNTRY”.

```
# Obtener el nodo de pais correspondiente en Neo4j
country_id = passenger["country_id"]
country_node = neo4j_graph.nodes.match("Countries", country_id=country_id).first()

# Establecer la relacion entre el nodo de pasajero y el nodo de pais
relationship = Relationship(passenger_node, "COUNTRY", country_node)
neo4j_graph.create(relationship)
```

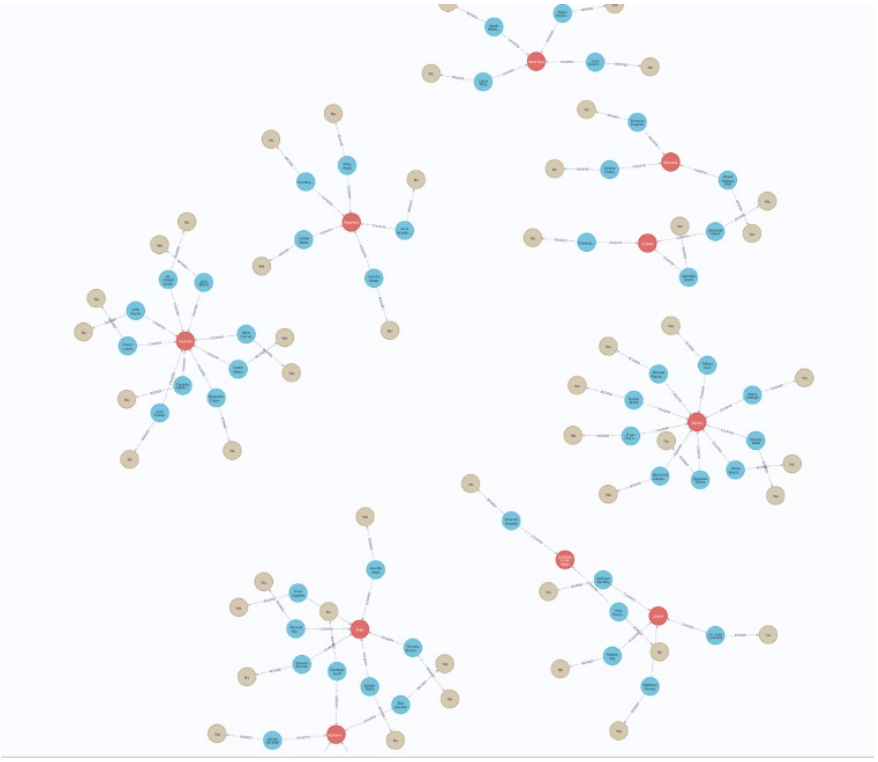
9. Seguidamente crearemos los nodos sobre las Reservas de los vuelos, los cuales contienen toda la información relacionada a este punto.

```
# Crear nodos de reservas en Neo4j y establecer relaciones con los pasajeros
for result in b_results:
    # Crear un nodo de reserva en Neo4j
    booking_node = Node("Booking",
                        booking_id=result["booking_id"],
                        num_passengers=result["num_passengers"],
                        sales_channel=result["sales_channel"],
                        trip_type=result["trip_type"],
                        purchase_lead=result["purchase_lead"],
                        length_of_stay=result["length_of_stay"],
                        flight_hour=result["flight_hour"],
                        flight_day=result["flight_day"],
                        route=result["route"],
                        booking_origin=result["booking_origin"],
                        wants_extra_baggage=result["wants_extra_baggage"],
                        want_preferred_seat=result["want_preferred_seat"],
                        wants_in_flight_meals=result["wants_in_flight_meals"],
                        flight_duration=result["flight_duration"],
                        booking_complete=result["booking_complete"])
    neo4j_graph.create(booking_node)
```

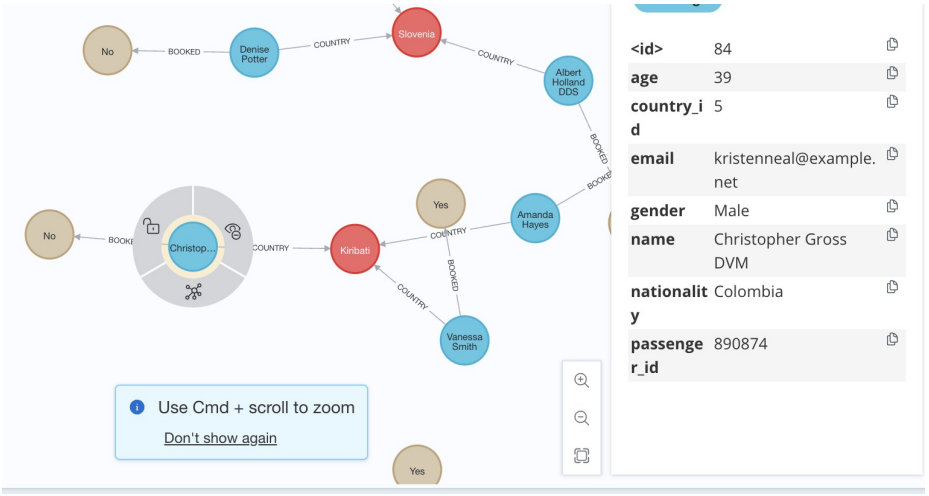
10. De la misma manera como realizamos en el paso 4, conectaremos la informaciòn de reservas con la colecciòn Pasajeros, de manera que el vínculo se de gracias al atributo passenger_id; creando así la relaciòn BOOKED

```
    # Obtener el nodo de pasajero correspondiente en Neo4j
    passenger_id = result["passenger_id"]
    passenger_node = neo4j_graph.nodes.match("Passenger", passenger_id=passenger_id).first()

    # Establecer la relacion entre el nodo de pasajero y el nodo de reserva
    relationship = Relationship(passenger_node, "BOOKED", booking_node)
    neo4j_graph.create(relationship)

# Cerrar la conexion con Neo4j (automáticamente cierra la conexion con MongoDB)
client.close()
```

Luego de realizar todo este procedimiento, obtendremos el siguiente resultado en Neo4J:



Donde podemos apreciar todos los nodos y relaciones creadas, apartir del c`dogo proporcionado anteriormente.



Cabe resaltar que esto se hizo sin consultas específicas, ya que lo que se quería inicialmente era reflejar la data completa en forma de grafos. Si deseamos realizar una consulta, solo debemos cambiar los parámetros del punto número 3, mencionado anteriormente.

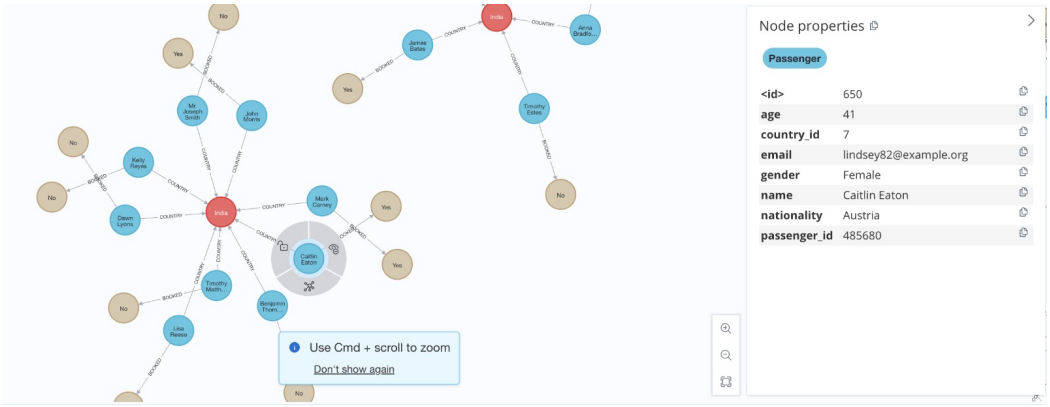
Aquì tres ejemplos de la consultas:

1. Queremos obtener la informacin completa de las personas que realizaron la reserva mayores a 30 años, organizados por paìs:

```
# Obtener todos los elementos de la coleccion de pasajeros en MongoDB
query = {"age":{"$gt":30}}
passenger_results = passengers.find(query)
```

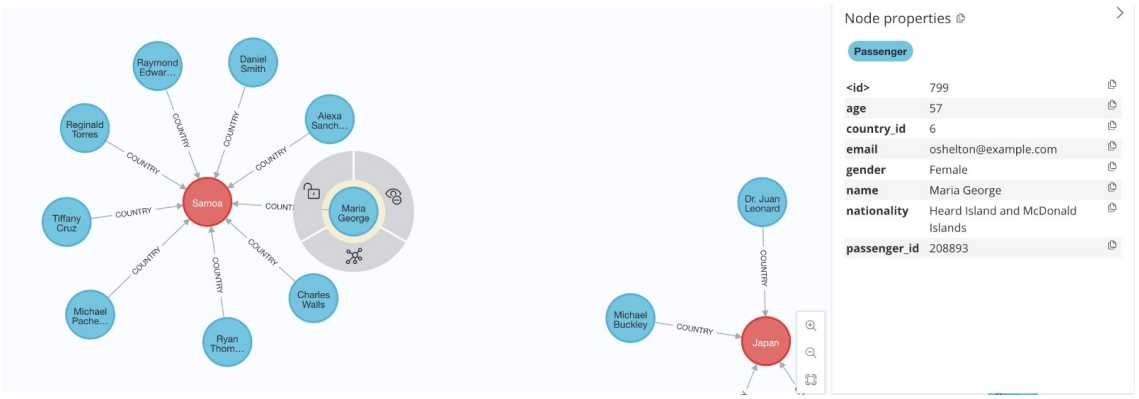
2. Queremos obtener la información completa de las personas que realizaron sus viajes desde la India:

```
# Obtener todos los elementos de la coleccion de paises en MongoDB
query = {"country" : "India"}
countries_results = countries.find(query)
```



3. Queremos obtener la información completa de las personas cuyo vuelo dura menos de 7 horas y 30 min.

```
# Obtener todos los elementos de la coleccion de reservas en MongoDB
query = {"flight_duration" : {"lt":7.5}}
results = collection.find(query)
```



Como podemos ver, es fácil manipular las Bases de Datos Documentales apoyados en MongoDB, ya que podemos administrarlas de manera eficiente. A su vez tambièn, podemos reflejar estas bases y sus consultas de manera gráfica, gracias a Neo4J.

CONCLUSIONES

- La forma más óptima que se pudo encontrar para realizar la conexión entre MongoDB y Neo4J es utilizar python, con las librerías pymongo y py2neo.

- Las Base de Datos Documentales son fáciles de manipular a la hora de reflejar dicha información en grafos, para una mejor comprensión de la data.

RECOMENDACIONES

- Se debe tener una buena Base de Datos, la cual nos facilita el trabajo, tanto a la hora de subir la información, como de manipularla.

- La Base de Datos a utilizar nos tiene que proporcionar la facilidad de definir relaciones entre sus datos, ya que sin esto será imposible crear los grafos y nodos correspondientes.