

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ  
УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

### **Лабораторна робота № 3**

з дисципліни “Бази даних 2. БД на основі XML”

тема “Практика використання графової бази даних Neo4J”

Виконав

студент III курсу

групи КП-83

Коваль Андрій Олександрович  
*(прізвище, ім'я, по батькові)*

варіант № 8

Зарахована

“ \_\_\_\_ ” “ \_\_\_\_ ” 20 \_\_\_\_ р.

викладачем

Петрашенко Андрієм Васильовичем  
*(прізвище, ім'я, по батькові)*

Київ 2021

## **Вступ**

Метою роботи є здобуття практичних навичок створення програм, орієнтованих на використання графової бази даних Neo4J за допомогою мови Python.

## **Завдання**

Реалізувати можливості формування графової бази даних в онлайн-режимі на основі модифікованої програми лабораторної роботи №2. На основі побудованої графової бази даних виконати аналіз сформованих даних.

### *Окремі програмні компоненти*

1. Інфраструктура лабораторної роботи №2:
  - 1.1. Redis server.
  - 1.2. Програма емуляції активності користувачі (вхід/вихід, відправка/отримання повідомлення).
  - 1.3. Виконувач задач (Worker).
2. Сервер Neo4J.
3. Інтерфейс користувача Neo4J.

### *Порядок виконання роботи*

1. В ЛР№2 залишити єдиний режим роботи - емуляція активності.
2. Внести доповнення у програму ЛР№2 шляхом додавання у повідомлення тегу або тегів з переліку, заданого у вигляді констант, обраних студентом.
3. Встановити сервер [Neo4J Community Edition](#).
4. Розробити схему бази даних Neo4J для збереження інформації про активності користувачів (вхід/вихід, відправлення/отримання

повідомлень) та Worker (перевірка на спам). Визначити вузли та зв'язки між ними на графі.

5. Розширити функціональність ЛР№2 шляхом збереження будь-якої активності (див. п. 4) у базу даних Neo4J у момент збереження даних у Redis.
6. У програмі “Інтерфейс користувача Neo4J” виконати і вивести результат наступних запитів до сервера Neo4J:

6.1. Задано список тегів (*tags*). Знайти всіх користувачів, що відправили або отримали повідомлення з набором тегів *tags*.

6.2. Задано довжину зв'язку  $N$  - кількість спільних повідомлень між користувачами. Знайти усі пари користувачів, що мають зв'язок довжиною  $N$  через відправлені або отримані повідомлення. Наприклад, якщо користувач А відправив повідомлення користувачу В, а В відправив повідомлення С, то довжина зв'язку між А і С є  $N=2$ .

6.3. Задано два користувача. Знайти на графі найкоротший шлях між ними через відправлені або отримані повідомлення.

6.4. Знайти авторів повідомлень, які пов'язані між собою лише повідомленнями, позначеними як “спам”.

6.5. Задано список тегів (*tags*). Знайти всіх користувачів, що відправили або отримали повідомлення з набором тегів *tags*, але ці користувачі не пов'язані між собою.

#### *Вимоги до засобів емуляції даних*

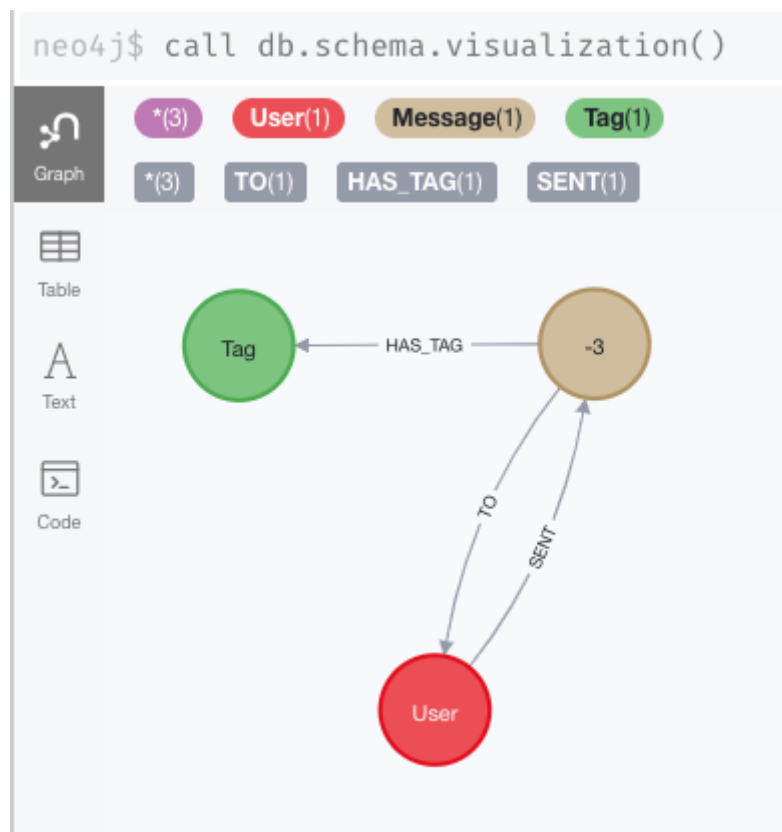
Забезпечити генерацію даних відносно невеликого обсягу, що підтверджують коректність виконання завдання пунктів 6.1 - 6.5.

*Вимоги до інтерфейсу користувача*

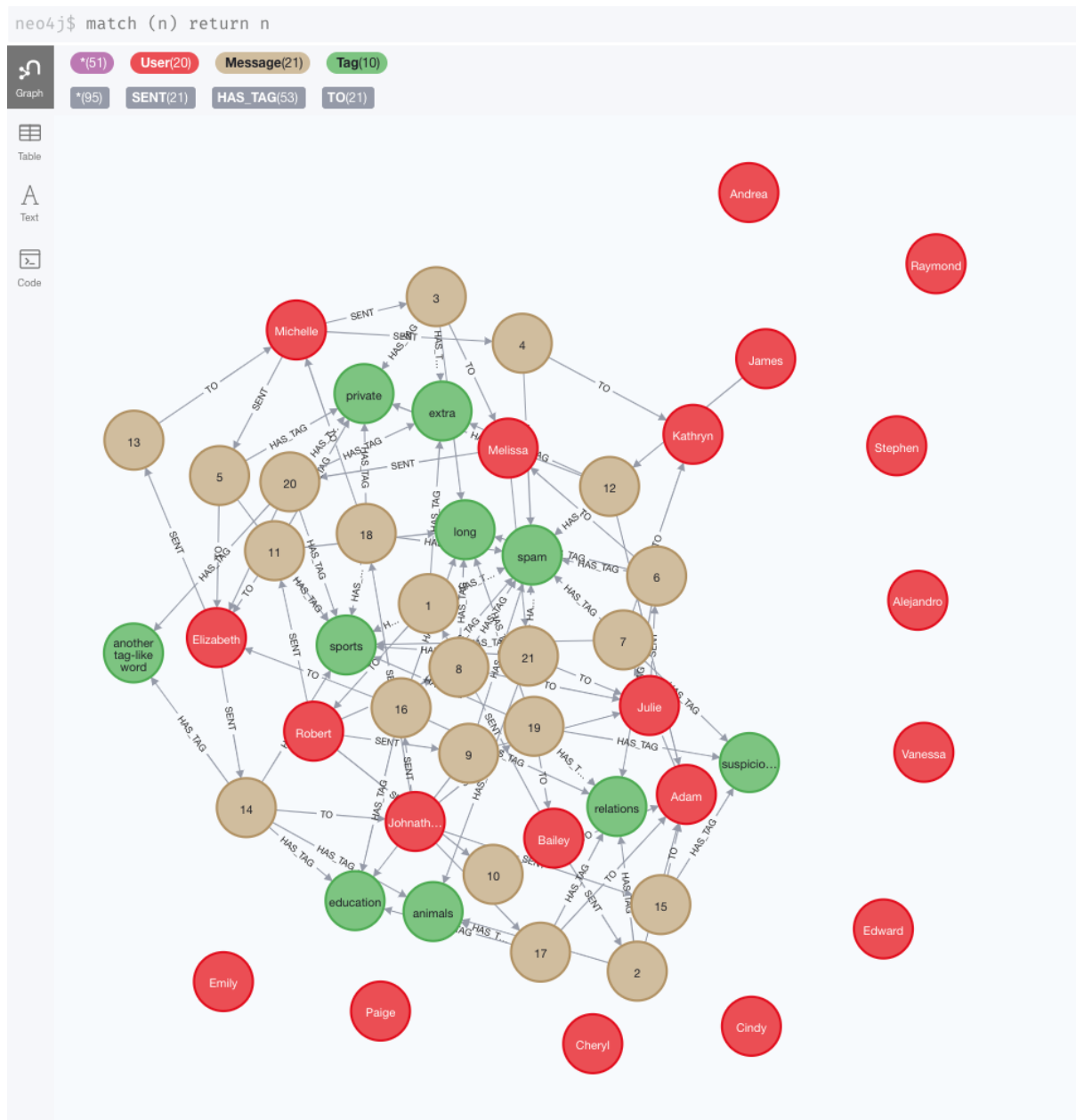
Використовувати консольний (текстовий) інтерфейс користувача.

## Реалізація

Схема створеної бази даних



## Візуалізація усіх нод згенерованої бази



1. Задано список тегів (*tags*). Знайти всіх користувачів, що відправили або отримали повідомлення з набором тегів *tags*.

```

match (tag:Tag) where tag.name in ['education']
match (msg:Message)→(tag)
match (sender:User)-[:SENT]→(msg)-[:TO]→(receiver:User)
return sender.username as username
union
match (tag:Tag) where tag.name in ['education']
match (msg:Message)→(tag)
match (sender:User)-[:SENT]→(msg)-[:TO]→(receiver:User)
return receiver.username as username

```

username

1 "Robert"

2 "Johnathan"

3 "Elizabeth"

4 "Julie"

5 "Adam"

2. Задано довжину зв'язку  $N$  - кількість спільних повідомлень між користувачами. Знайти усі пари користувачів, що мають зв'язок довжиною  $N$  через відправлені або отримані повідомлення. Наприклад, якщо користувач А відправив повідомлення користувачу В, а В відправив повідомлення С, то довжина зв'язку між А і С  $\in N=2$ .

```

match (user1:User)
match (user2:User)
match (user1)-[:SENT|TO*2]-(user2) where user1.username <>
user2.username // 2 is N
return user1.username as User1, user2.username as User2

```

# Приклади результатів

```
1 match (user1:User)
2 match (user2:User)
3 match (user1)-[:SENT|T0*4]-(user2) where user1.username <> user2.username
4 return user1.username as User1, user2.username as User2
```

	User1	User2
1	"Cynthia"	"Aaron"
2	"Cynthia"	"Jasmine"
3	"Cynthia"	"Robert"
4	"Cynthia"	"Kimberly"
5	"Cynthia"	"Elizabeth"
6	"Cynthia"	"Aaron"
7	"Cynthia"	"Jasmine"

Started streaming 546 records after 1 ms and completed after 2 ms.


```
1 match (user1:User)
2 match (user2:User)
3 match (user1)-[:SENT|T0*2]-(user2) where user1.username <> user2.username
4 return user1.username as User1, user2.username as User2
```

	User1	User2
1	"Cynthia"	"Kimberly"
2	"Cynthia"	"Robert"
3	"Cynthia"	"Jasmine"
4	"Cynthia"	"Kimberly"
5	"Elizabeth"	"Robert"
6	"Elizabeth"	"Jasmine"
7	"Elizabeth"	"Robert"


Started streaming 116 records after 1 ms and completed after 2 ms.

neo4j\$

```
1 match (user1:User)
2 match (user2:User)
3 match (user1)-[:SENT|TO*3]-(user2) where user1.username <> user2.username
4 return user1.username as User1, user2.username as User2
```

Table

(no changes, no records)

Code

Completed after 1 ms.

3. Задано два користувача. Знайти на графі найкоротший шлях між ними через відправлені або отримані повідомлення.

```
match (user1:User {username: 'Shawn'})
match (user2:User {username: 'Madison'})
with shortestPath((user1)-[:SENT|TO*]-(user2)) as path
return path
```



```

1 match (user1:User {username: 'Shawn'})
2 match (user2:User {username: 'Madison'})
3 with shortestPath((user1)-[:SENT|TO*]-(user2)) as path
4 return path

```

Graph

\*(3) User(2) Message(1)

\*(2) SENT(1) TO(1)

Table

Text

Warn

Code

Приклад користувачів без зв'язку

```

1 match (user1:User {username: 'Jasmine'})
2 match (user2:User {username: 'Taylor'})
3 with shortestPath((user1)-[:SENT|TO*]-(user2)) as path
4 return path

```

Table

	path
1	null

Text

Warn

Code

4. Знайти авторів повідомлень, які пов'язані між собою лише повідомленнями, позначеними як “спам”.

1	match (tag:Tag {name: 'spam'})	
2	match (msg:Message)→(tag)	
3	match (sender:User)-[:SENT]→(msg)-[:TO]→(receiver:User)	
4	match (msg)-[connections:HAS_TAG]→(allTags:Tag)	
5	with sender, receiver, count(connections) as messageTagsCount	
6	where messageTagsCount = 1	
7	return sender.username as Sender, receiver.username as Receiver	
8		

Table	Sender	Receiver
1	"Michelle"	"Kathryn"

Без перевірки на кількість тегів, яке має повідомлення, ми отримуємо більше результатів

1	match (tag:Tag {name: 'spam'})	
2	match (msg:Message)→(tag)	
3	match (sender:User)-[:SENT]→(msg)-[:TO]→(receiver:User)	
4	// match (msg)-[connections:HAS_TAG]→(allTags:Tag)	
5	// with sender, receiver, count(connections) as messageTagsCount	
6	// where messageTagsCount = 1	
7	return sender.username as Sender, receiver.username as Receiver	
8		

Table	Sender	Receiver
1	"Melissa"	"Julie"
2	"Johnathan"	"Michelle"
3	"Johnathan"	"Elizabeth"
4	"James"	"Adam"
5	"Robert"	"Julie"
6	"Robert"	"Julie"

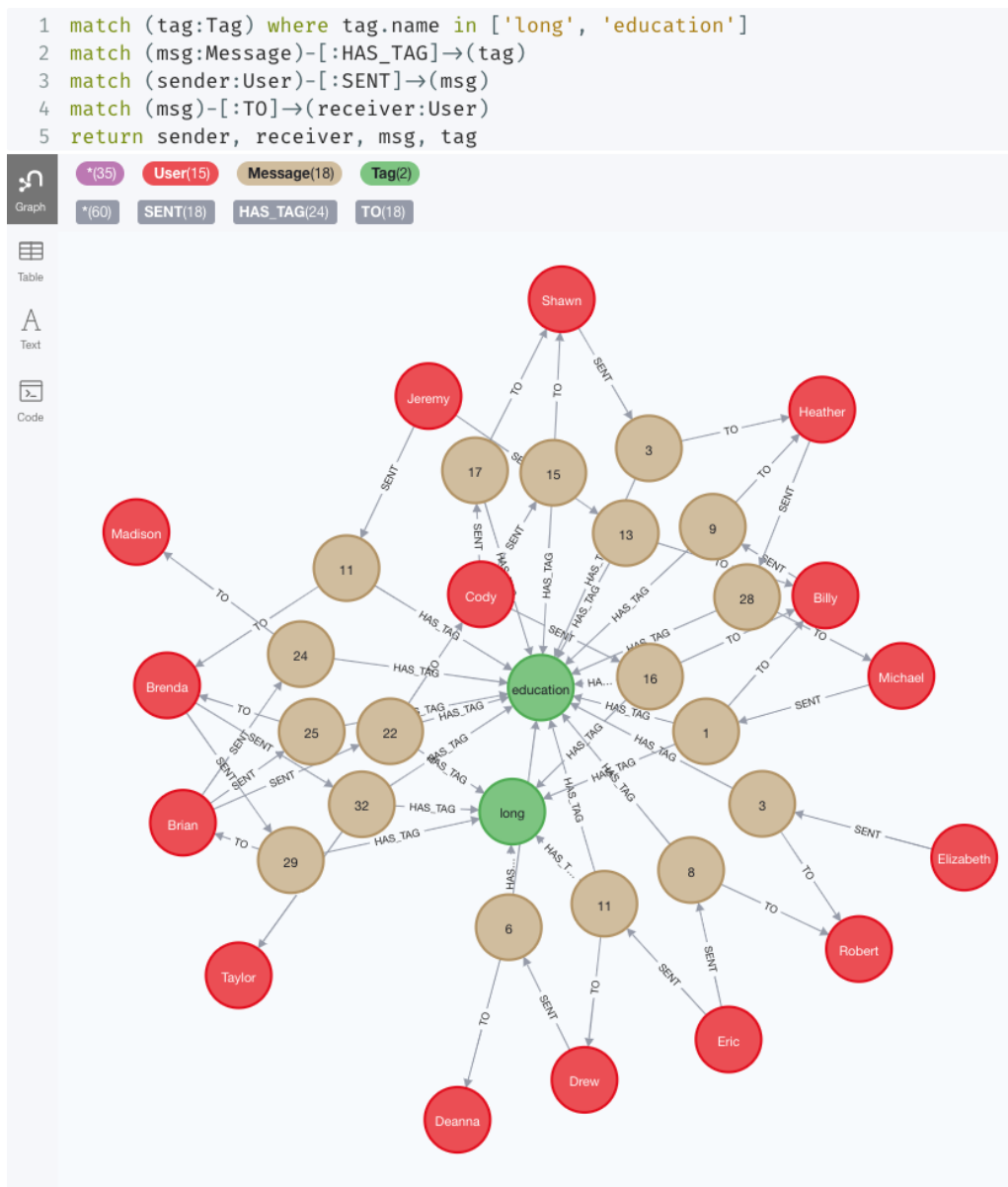
5. Задано список тегів (*tags*). Знайти всіх користувачів, що відправили або отримали повідомлення з набором тегів *tags*, але ці користувачі не пов'язані між собою.

Була спроба написати такий запит

```
match (tag:Tag) where tag.name in ['long', 'education']
match (msg:Message)-[:HAS_TAG]->(tag)
match (sender:User)-[:SENT]->(msg)
match (msg)-[:TO]->(receiver:User)
match (sender)-[sendProps:SENT]->(msg)-[receiveProps:TO]->(receiver)
where sendProps.to <> receiver.name and receiveProps.from <> sender.name
return sendProps.to, receiver.name, receiveProps.from, sender.name
```

Він на жаль не спрацював. Очевидно, що десь є помилка, але за браком досвіду роботи з Cypher я не зміг досягнути робочого результату.

У загальному вигляді вдалося отримати список усіх користувачів, які відправляли повідомлення з деякими тегами. Наочним є зображення наступного графу



Також наступні запити (спочатку один, потім другий закоментований рядок) не дали результату, хоча згідно документації мали б це зробити

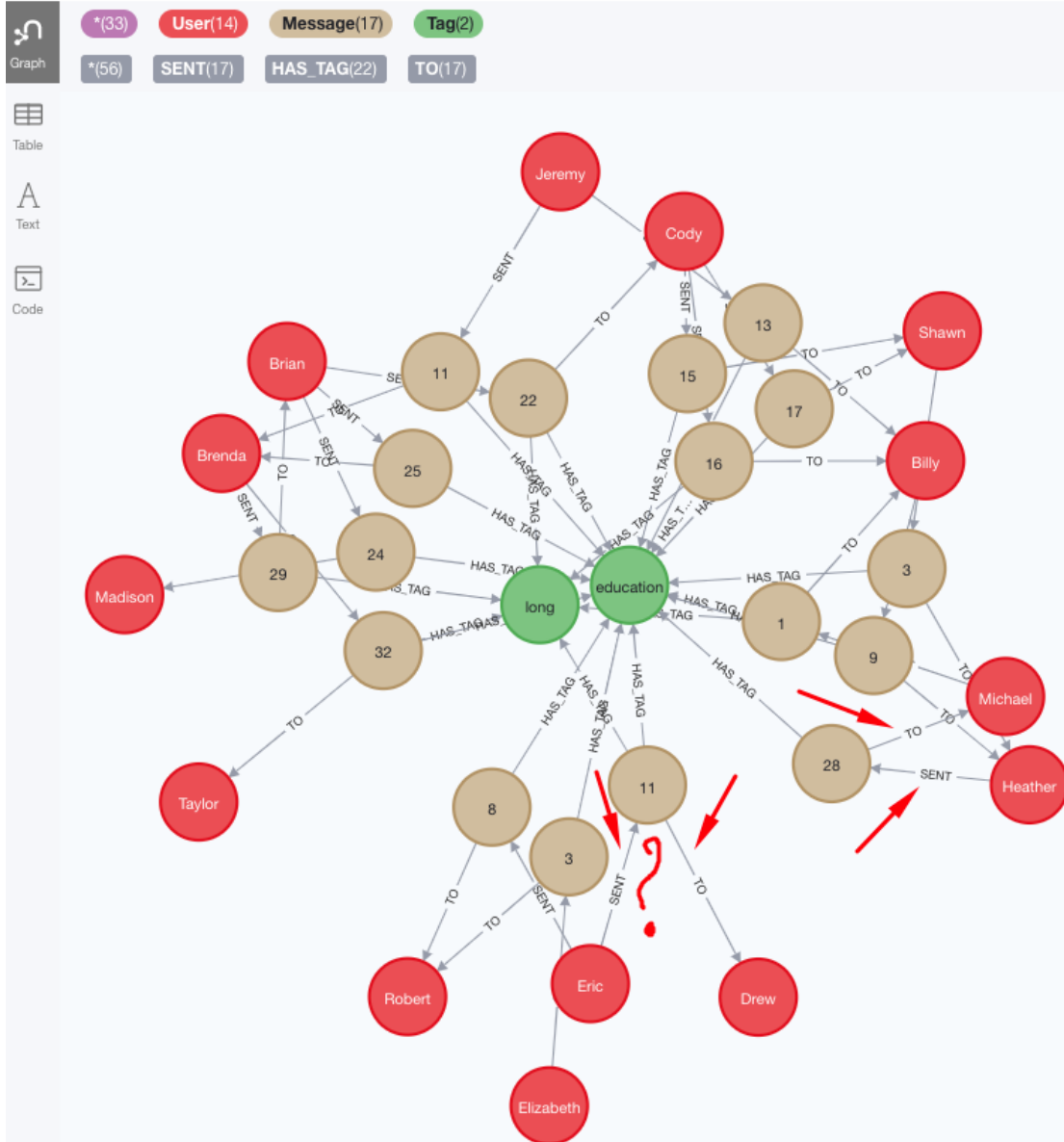
```

match (tag:Tag) where tag.name in ['long', 'education']
match (msg:Message)-[:HAS_TAG]->(tag)
match (sender:User)-[:SENT]->(msg)
match (msg)-[:TO]->(receiver:User)
// match (sender)-[:SENT|TO*4..]->(receiver) // ?????
// where not (sender)-[:SENT|TO*2]->(receiver) // ?????
return sender, receiver, msg, tag

```

Наглядною демонстрацією неправильності запиту є його результат в графі

```
1 match (tag:Tag) where tag.name in ['long', 'education']
2 match (msg:Message)-[:HAS_TAG]→(tag)
3 match (sender:User)-[:SENT]→(msg)
4 match (msg)-[:TO]→(receiver:User)
5 match (sender)-[:SENT|TO*4..]→(receiver) // ?????
6 return sender, receiver, msg, tag
```



## Програмний код

Для виконання даної роботи я оновив генератор даних та створив клас для роботи з запитами у Neo4j.

data\_generator.py

```
from faker import Faker
from random import choice, randint
from services.redis_connection import r
import services.messages_service as messages_service
import services.users_service as users_service

class DataGenerator:
    tags = [
        'long', 'extra', 'suspicious', 'animals', 'relations', 'sports',
        'education', 'private', 'another tag-like word', 'spam'
    ]
    def __init__(self):
        self.__fake = Faker()

    def start(self):
        r.flushall()
        count = 10
        users = self.__users(count)
        messages = self.__messages(count)
        for user in users:
            users_service.register(user)

        for i in range(count):
            random_messages_count = randint(0, 5)
            sender = users[i]
            other_users = users[0:i] + users[i + 1:]
            for j in range(random_messages_count):
                receiver = choice(other_users)
                message = choice(messages)
                tags_count = randint(0, len(self.tags) // 2)
                random_tags = '' if tags_count == 0 else
                ','.join(set(choice(self.tags) for _ in range(tags_count)))
                messages_service.send_message(message, sender, receiver,
                random_tags)

    def __users(self, count = 20):
        """Generates "count" of usernames"""
        return [self.__fake.unique.first_name() for _ in range(count)]

    def __messages(self, count = 20):
        """Count random message texts"""
        return [
```

```

        (self.__fake.sentence(nb_words=5) + ('spam' if choice([True,
False]) else ''))
        for _ in range(count)
    ]

def main():
    data_gen = DataGenerator()
    data_gen.start()

if __name__ == '__main__':
    main()

```

## neo4j\_connection.py

```

from neo4j import GraphDatabase

class Neo4jConnection:
    def __init__(self):
        self.__driver = GraphDatabase.driver("neo4j://localhost:7687",
auth=("neo4j", "password"))

    @staticmethod
    def __add_message(tx, message_dict):
        tags = message_dict['tags'].split(',')
        query = ["CREATE (msg:Message {id: $id})"]
        for idx, tag in enumerate(tags):
            if tag != '' and tag is not None:
                query.append(f"MERGE (tag{idx}:Tag {{ name: '{tag}' }})")
                query.append(f"CREATE (msg)-[:HAS_TAG]->(tag{idx})")
        query.append("MERGE (sender:User {username: $senderName})")
        query.append("MERGE (receiver:User {username: $receiverName})")
        query.append("MERGE (sender)-[:SENT {to: $receiverName}]->(msg)")
        query.append("MERGE (msg)-[:TO {from: $senderName}]->(receiver)")
        query_string = '\n'.join(query)
        print('\n' + query_string + '\n')
        tx.run(
            query_string,
            id=message_dict['id'],
            senderName=message_dict['sender-name'],
            receiverName=message_dict['receiver-name']
        )

    def add_message(self, message_dict):
        with self.__driver.session() as session:
            session.write_transaction(self.__add_message, message_dict)

    def __register_user(self, tx, username):
        already_existing_users = tx.run('MATCH (user:User {username: $username})
RETURN user', username=username)
        for val in already_existing_users:
            existing_username = val['user']['username']
            print(existing_username, username)

```

```
        if existing_username == username:
            return
    tx.run(
        "MERGE (user:User {username: $username})",
        username=username
    )

    def register_user(self, username):
        with self.__driver.session() as session:
            session.write_transaction(self.__register_user, username)

g = Neo4jConnection()
```



## **Висновки**

Я здобув практичні навички створення програм, орієнтованих на використання графової бази даних Neo4J за допомогою мови Python.

