МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

# **Лабораторна робота №** 3

з дисципліни “*Бази даних. Частина 2*”

на тему “*Практика використання графової бази даних Neo4J*”

Виконав
студент III курсу
групи КП-81

Подлеснюк Богдан Анатолійович

Зарахована: Петрашенко А. В.

Київ 2021

**Мета роботи:** здобуття практичних навичок створення програм, орієнтованих на використання графової бази даних Neo4J за допомогою мови Python.
**Завдання роботи** полягає у наступному:
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.

# Код програми

**Controller.py**

```
from inspect import signature
from view import View

from enum import Enum


class Tags(Enum):
    work = 1,
    family = 2

    @classmethod
    def has_member(cls, value):
        return value in Tags._member_names_


class Controller(object):
    @staticmethod
    def make_choice(menu_list: list, name_of_menu: str):
        try:
            View.draw_menu(menu_list, name_of_menu)
            return Controller.get_uint_value("Make your choice: ",
len(menu_list))

        except Exception as e:
            View.show_error(str(e))

    @staticmethod
    def considering_choice(controller, choice: int, list_of_func:
list):
        try:
            if choice > len(list_of_func) - 1:
                raise Exception("func is not exist")

            desired_func = list_of_func[choice]
            desired_func(controller)
        except Exception as e:
            View.show_error(str(e))

    @staticmethod
    def get_func_arguments(func, amount_of_missing_arguments=0) ->
list:
        from data import special_parameters
        list_of_parameters = signature(func).parameters
        list_of_arguments = []
        length = len(list_of_parameters)
        for i in range(length - amount_of_missing_arguments):
            list_of_arguments.append(Controller.get_value(
```

```
                f"Enter
{list(list_of_parameters)[i]}{special_parameters[list(list_of_para
meters)[i]] if list(list_of_parameters)[i] in special_parameters
else ''}: ",
                str))
        # for parameter in list_of_parameters:
        #
list_of_arguments.append(Controller.get_value(f"Enter {parameter}:
", str))
        return list_of_arguments

    @staticmethod
    def get_uint_value(msg: str, top_line: int = None):
        while True:
            number = input(msg)
            if number.isdigit():
                number = int(number)
                if top_line is None or 0 <= number < top_line:
                    return number

    @staticmethod
    def get_value(msg: str, type_of_var):
        while True:
            try:
                usr_input = input(msg)
                if type_of_var == str:
                    if len(usr_input) != 0:
                        return type_of_var(usr_input)
                else:
                    return type_of_var(usr_input)
            except Exception as e:
                View.show_error(str(e))

    @staticmethod
    def stop_loop(controller):
        controller.loop = False
```

**EmulationController.py**

```
class View(object):

    @staticmethod
    def draw_menu(menu_list, name_of_menu: str):
        print(f"\n{name_of_menu}")
        number = 0
        for menu_item in menu_list:
            print(f" {number}: {menu_item}")
            number += 1
```

```
    @staticmethod
    def show_item(item):
        print(f"Item: {item}")

    @staticmethod
    def show_way(nodes: list):
        way = ""
        for node in nodes:
            way += f"{node} ->"
        print(way[:-3])

    @staticmethod
    def show_items(items: list):
        count = 1
        for item in items:
            print(f"{count}: {item}")
            count += 1

    @staticmethod
    def show_error(err: str):
        print(f"Error: {err}")

    @staticmethod
    def show_text(text: str):
        print(text)

    @staticmethod
    def print_line():

print('-----------------------------------------------------------------
------------------------------')

    @staticmethod
    def print_list(name_of_list, list):
        print(name_of_list)
        count = 1
        for item in list:
            print(f"{count}: {item}")
            count += 1
```

**Neo4jController.py**

```
from controller.Controller import Controller
from servers.neo4j_server.Neo4jServer import Neo4jServer
from view import View
```

```
class Neo4jController(object):
    def __init__(self):
        self.__server = Neo4jServer()
        self.__menu = 'Neo4j menu'
        self.loop = True
        self.start()

    def start(self):
        from data import menu_list
        try:
            while self.loop:
                choice =
Controller.make_choice(menu_list[self.__menu].keys(), self.__menu)
                Controller.considering_choice(self, choice,
list(menu_list[self.__menu].values()))
        except Exception as e:
            View.show_error(str(e))

    def get_users_with_tagged_messages(self):
        res =
self.__server.get_users_with_tagged_messages(*Controller.get_func_
arguments(
            self.__server.get_users_with_tagged_messages))
        View.print_list("Users: ", res)

    def shortest_way_between_users(self):
        res =
self.__server.shortest_way_between_users(*Controller.get_func_argu
ments(
            self.__server.shortest_way_between_users))
        View.show_way(res)

    def get_users_with_n_long_relations(self):
        res =
self.__server.get_users_with_n_long_relations(*Controller.get_func
_arguments(
            self.__server.get_users_with_n_long_relations))
        View.print_list("Pairs of users: ", res)

    def get_users_wicth_have_only_spam_conversation(self):
        res =
self.__server.get_users_wicth_have_only_spam_conversation()
        View.print_list("Pairs of users: ", res)

    def get_unrelated_users_with_tagged_messages(self):
        res =
self.__server.get_unrelated_users_with_tagged_messages(*Controller
.get_func_arguments(

self.__server.get_unrelated_users_with_tagged_messages))
        View.print_list("Groups of unrelated users: ", res)
```

## Listener.py

```
import datetime
from threading import Thread
import logging

import redis


class EventListener(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.__r = redis.Redis(charset="utf-8",
decode_responses=True)
        self.__events = []

    def run(self):
        pubsub = self.__r.pubsub()
        pubsub.subscribe(['users', 'spam'])
        for item in pubsub.listen():
            if item['type'] == 'message':
                message = "\nEVENT: %s | %s" % (item['data'],
datetime.datetime.now())
                self.__events.append(message)
                logging.info(message)

    def get_events(self):
        return self.__events
```

## Neo4jServer.py

```
from neo4j import GraphDatabase

from view import View
from controller.Controller import Tags


class Neo4jServer(object):
    def __init__(self):
        self.__driver =
GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j",
"123"))

    def close(self):
        self.__driver.close()

    def __truncate_db(self):
```

```
        with self.__driver.session() as session:
            session.run("MATCH (n) DETACH DELETE n")

    def registration(self, username, redis_id):
        with self.__driver.session() as session:
            session.run("MERGE (u:user {name: $username, redis_id:
$redis_id})"
                        "ON CREATE SET u.online = false",
username=username, redis_id=redis_id)

    def sign_in(self, redis_id):
        with self.__driver.session() as session:
            session.run("MATCH (u:user {redis_id: $redis_id}) SET
u.online = true", redis_id=redis_id)

    def sign_out(self, redis_id):
        with self.__driver.session() as session:
            session.run("MATCH (u:user {redis_id: $redis_id}) SET
u.online = false", redis_id=redis_id)

    def create_message(self, sender_id, consumer_id, message:
dict):
        with self.__driver.session() as session:
            try:
                #
session.write_transaction(self.__create_message_as_node,
message["id"], message["tags"])
                messages_id =
session.write_transaction(self.__create_message_as_relation,
int(sender_id),

int(consumer_id), message["id"])
                for tag in message["tags"]:

session.write_transaction(self.__add_tag_to_messages, messages_id,
tag)
            except Exception as e:
                View.show_error(str(e))

    @staticmethod
    def __create_message_as_relation(tx, sender_id, consumer_id,
message_id):
        result = tx.run("MATCH(a: user {redis_id: $sender_id}),
(b:user {redis_id: $consumer_id})"
                        "MERGE(a) - [r: messages]->(b)"
                        "ON CREATE SET r.all = [$message_id],
r.spam = [], r.tags = []"
                        "ON MATCH SET r.all = r.all + $message_id
"
                        "RETURN id(r)",
                        sender_id=sender_id,
consumer_id=consumer_id, message_id=message_id)
        return result.single()[0]

    @staticmethod
```

```
    def __add_tag_to_messages(tx, messages_id, tag):
        tx.run("MATCH ()-[r]-() where ID(r) = $messages_id "
               "FOREACH(x in CASE WHEN $tag in r.tags THEN [] ELSE
[1] END | "
               "SET r.tags = coalesce(r.tags,[]) + $tag)",
messages_id=messages_id, tag=tag)

    def deliver_message(self, redis_id):
        with self.__driver.session() as session:
            session.run("MATCH (m:messages {redis_id: $redis_id })
SET m.delivered = true", redis_id=redis_id)

    def mark_message_as_spam(self, redis_id):
        with self.__driver.session() as session:
            session.run("MATCH (u1:user)-[r:messages]->(u2:user) "
                        "WHERE $redis_id IN r.all AND NOT
$redis_id IN r.spam "
                        "SET r.spam = r.spam + $redis_id",
redis_id=redis_id)

    def get_users_with_tagged_messages(self, tags):
        return
self.__record_to_list(self.__get_users_with_tagged_messages_from_d
b(tags), 'name')

    def get_unrelated_users_with_tagged_messages(self, tags):
        list_of_names =
self.__record_to_list(self.__get_users_with_tagged_messages_from_d
b(tags), 'name')
        unrelated_users = []
        for name1 in list_of_names:
            group = [name1]
            for name2 in list_of_names:
                if name1 != name2:
                    res =
self.__check_relation_between_users(name1, name2)
                    if not res and name1 not in group:
                        group.append(name2)
            unrelated_users.append(group)

        return unrelated_users

    def __get_users_with_tagged_messages_from_db(self, tags):
        with self.__driver.session() as session:
            tags = tags.split(", ")
            for tag in tags:
                if not Tags.has_member(tag):
                    raise ValueError(f"Tag: {tag} doesnt exist")

            query = "MATCH (u:user)-[r:messages]-() WHERE"
            for tag in tags:
                query += f" \'{tag}\' IN r.tags AND"

            # removing last AND
            query = query[:-3] + "RETURN u"
```

```
            return session.run(query)

    def __check_relation_between_users(self, username1,
username2):
        with self.__driver.session() as session:
            res = session.run("MATCH  (u1:user {name:
$username1}), (u2:user {name: $username2}) "
                              "RETURN
EXISTS((u1)-[:messages]-(u2))", username1=username1,
username2=username2)
            return res.single()[0]

    def shortest_way_between_users(self, username1, username2):
        users = self.get_users()
        if username1 not in users or username2 not in users:
            raise ValueError('Invalid users names')
        with self.__driver.session() as session:
            shortest_path = session.run("MATCH p =
shortestPath((u1:user)-[*..10]-(u2:user)) "
                                        "WHERE u1.name =
$username1 AND u2.name = $username2 "
                                        "RETURN p",
username1=username1, username2=username2)
            if shortest_path.peek() is None:
                raise Exception(f"Way between {username1} and
{username2} doesnt exist")
            for record in shortest_path:
                nodes = record[0].nodes
                path = []
                for node in nodes:
                    path.append(node._properties['name'])
                return path

    def get_users_with_n_long_relations(self, n):
        with self.__driver.session() as session:
            res = session.run(f"MATCH p = (u1:user)-[*]-(u2:user)"
                              f"WHERE u1 <> u2 AND "
                              f"reduce(total_len = 0, r IN
relationships(p)| total_len + size(r.all)) = {n} "
                              f"RETURN u1, u2")
            return self.__pair_record_to_list(res, 'name')

    def get_users_wicth_have_only_spam_conversation(self):
        with self.__driver.session() as session:
            res = session.run("MATCH p = (u1:user)-[]-(u2:user)"
                              "WHERE u1 <> u2 AND all(x in
relationships(p) WHERE x.all = x.spam)"
                              "RETURN u1, u2")
            return self.__pair_record_to_list(res, 'name')

    def __pair_record_to_list(self, res, pull_out_value):
        my_list = list(res)
        my_list = list(dict.fromkeys(my_list))
        new_list = []
        for el in my_list:
```

```
            list_el = list(el)
            if list_el not in new_list and list_el[::-1] not in
new_list:
                new_list.append(el)

        return [[el[0]._properties[pull_out_value],
el[1]._properties[pull_out_value]] for el in new_list]

    def get_users(self):
        with self.__driver.session() as session:
            res = session.run("MATCH (u:user) RETURN u")
            return self.__record_to_list(res, 'name')

    def __record_to_list(self, res, pull_out_value):
        my_list = list(res)
        my_list = list(dict.fromkeys(my_list))
        return [el[0]._properties[pull_out_value] for el in
my_list]
```

**RedisServer.py**

```
import redis
import datetime
import logging

from servers.neo4j_server.Neo4jServer import Neo4jServer

logging.basicConfig(filename="./events.log", level=logging.INFO,
filemode="w")


class RedisServer(object):
    def __init__(self, neo4j_server: Neo4jServer):
        self.__r = redis.Redis(charset="utf-8",
decode_responses=True)
        self.__neo4j_server = neo4j_server

    def registration(self, username):
        if self.__r.hget('users:', username):
            raise Exception(f"User with name: \'{username}\'
already exists")

        user_id = self.__r.incr('user:id:')
        pipeline = self.__r.pipeline(True)
        pipeline.hset('users:', username, user_id)
        pipeline.hmset(f"user:{user_id}", {
            'login': username,
            'id': user_id,
            'queue': 0,
```

```
            'checking': 0,
            'blocked': 0,
            'sent': 0,
            'delivered': 0
        })

        pipeline.execute()
        self.__neo4j_server.registration(username, user_id)
        logging.info(f"User {username} registered at
{datetime.datetime.now()} \n")
        return user_id

    def sign_in(self, username):
        user_id = self.__r.hget("users:", username)

        if not user_id:
            raise Exception(f"User {username} does not exist ")

        self.__r.sadd("online:", username)
        logging.info(f"User {username} logged in at
{datetime.datetime.now()} \n")
        self.__r.publish('users', "User %s signed in" %
self.__r.hmget(f"user:{user_id}", 'login')[0])
        self.__neo4j_server.sign_in(user_id)
        return int(user_id)

    def sign_out(self, user_id) -> int:
        logging.info(f"User {user_id} signed out at
{datetime.datetime.now()} \n")
        self.__r.publish('users', "User %s signed out" %
self.__r.hmget(f"user:{user_id}", 'login')[0])
        self.__neo4j_server.sign_out(user_id)
        return self.__r.srem("online:",
self.__r.hmget(f"user:{user_id}", 'login')[0])

    def create_message(self, message_text, tags: list, consumer,
sender_id) -> int:

        message_id = int(self.__r.incr('message:id:'))
        consumer_id = self.__r.hget("users:", consumer)

        if not consumer_id:
            raise Exception(f"{consumer} user does not exist, user
can't send a message")

        pipeline = self.__r.pipeline(True)

        pipeline.hmset('message:%s' % message_id, {
            'text': message_text,
            'id': message_id,
            'sender_id': sender_id,
            'consumer_id': consumer_id,
            'tags': ','.join(tags),
            'status': "created"
        })
```

```
        pipeline.lpush("queue:", message_id)
        pipeline.hmset('message:%s' % message_id, {
            'status': 'queue'
        })
        pipeline.zincrby("sent:", 1, "user:%s" %
self.__r.hmget(f"user:{sender_id}", 'login')[0])
        pipeline.hincrby(f"user:{sender_id}", "queue", 1)
        pipeline.execute()

        self.__neo4j_server.create_message(sender_id, consumer_id,
{"id": message_id, "tags": tags})
        return message_id

    def get_messages(self, user_id):
        messages = self.__r.smembers(f"sentto:{user_id}")
        messages_list = []
        for message_id in messages:
            message = self.__r.hmget(f"message:{message_id}",
["sender_id", "text", "status", "tags"])
            sender_id = message[0]
            messages_list.append("From: %s - %s" %
(self.__r.hmget("user:%s" % sender_id, 'login')[0], message[1]))
            # messages_list.append("From: %s - %s, tags: %s" %
(self.__r.hmget("user:%s" % sender_id, 'login')[0], message[1],
message[3]))
            if message[2] != "delivered":
                pipeline = self.__r.pipeline(True)
                pipeline.hset(f"message:{message_id}", "status",
"delivered")
                pipeline.hincrby(f"user:{sender_id}", "sent", -1)
                pipeline.hincrby(f"user:{sender_id}", "delivered",
1)
                pipeline.execute()
                self.__neo4j_server.deliver_message(message_id)

        return messages_list

    def get_message_statistics(self, user_id):
        current_user = self.__r.hmget(f"user:{user_id}", ['queue',
'checking', 'blocked', 'sent', 'delivered'])
        return "In queue: %s\nChecking: %s\nBlocked: %s\nSent:
%s\nDelivered: %s" % tuple(current_user)

    def get_online_users(self) -> list:
        return self.__r.smembers("online:")

    def get_top_senders(self, amount_of_top_senders) -> list:
        return self.__r.zrange("sent:", 0,
int(amount_of_top_senders) - 1, desc=True, withscores=True)

    def get_top_spamers(self, amount_of_top_spamers) -> list:
        return self.__r.zrange("spam:", 0,
int(amount_of_top_spamers) - 1, desc=True, withscores=True)
```

**main.py**

```
from controller.Controller import Controller
from controller.EmulationController import EmulationController
from controller.Neo4jController import Neo4jController
from view import View
from faker import Faker
import random


def emulation():

    fake = Faker()
    users_count = 5
    users = [fake.profile(fields=['username'],
sex=None)['username'] for u in range(users_count)]
    threads = []
    try:

        for i in range(users_count):
            threads.append(EmulationController(users[i], users,
users_count, random.randint(1, 2)))
        for thread in threads:
            thread.start()

    except Exception as e:
        View.show_error(str(e))
    finally:
        for thread in threads:
            if thread.is_alive():
                thread.stop()


if __name__ == "__main__":
    choice = Controller.make_choice(["Neo4j", "Emulation(use one
time with worker for generate db)"], "Program mode")
    if choice == 0:
        Neo4jController()
    elif choice == 1:
        emulation()
```

**data.py**

```
from controller.Neo4jController import Neo4jController
from controller.Controller import Controller, Tags
from servers.neo4j_server.Neo4jServer import Neo4jServer

menu_list = {
```

```
    'Neo4j menu': {
        'Tagged messages(6.1)':
Neo4jController.get_users_with_tagged_messages,
        'N long relations(6.2)':
Neo4jController.get_users_with_n_long_relations,
        'Shortest way(6.3)':
Neo4jController.shortest_way_between_users,
        'Only spam conversation(6.4)':
Neo4jController.get_users_wicth_have_only_spam_conversation,
        'Tagged messages without relations(6.5)':
Neo4jController.get_unrelated_users_with_tagged_messages,
        'Exit': Controller.stop_loop,
    }
}

roles = {
    'utilizer': 'Utilizer menu',
    'admin': 'Admin menu'
}

neo4j = Neo4jServer()
special_parameters = {
    'role': '(admin or utilizer)',
    'tags': '('+', '.join(x.name for x in list(Tags))+')(Enter
comma-separated values)',
    'username1': '(' + ', '.join(x for x in neo4j.get_users()) +
')',
    'username2': '(' + ', '.join(x for x in neo4j.get_users()) +
')'
}
```

**worker.py**

```
import random
import time
from threading import Thread
import redis

from servers.neo4j_server.Neo4jServer import Neo4jServer
from view import View


class Worker(Thread):

    def __init__(self, delay, neo4j_server: Neo4jServer):
        Thread.__init__(self)
        self.__neo4j_server = neo4j_server
        self.__loop = True
        self.__r = redis.Redis(charset="utf-8",
```

```
decode_responses=True)
        self.__delay = delay

    def run(self):
        while self.__loop:
            message = self.__r.brpop("queue:")
            if message:
                message_id = int(message[1])

                self.__r.hmset(f"message:{message_id}", {
                    'status': 'checking'
                })
                message = self.__r.hmget(f"message:{message_id}",
["sender_id", "consumer_id"])
                sender_id = int(message[0])
                consumer_id = int(message[1])
                self.__r.hincrby(f"user:{sender_id}", "queue", -1)
                self.__r.hincrby(f"user:{sender_id}", "checking",
1)
                time.sleep(self.__delay)
                is_spam = random.random() > 0.6
                pipeline = self.__r.pipeline(True)
                pipeline.hincrby(f"user:{sender_id}", "checking",
-1)
                if is_spam:
                    sender_username =
self.__r.hmget(f"user:{sender_id}", 'login')[0]
                    pipeline.zincrby("spam:", 1,
f"user:{sender_username}")
                    pipeline.hmset(f"message:{message_id}", {
                        'status': 'blocked'
                    })
                    pipeline.hincrby(f"user:{sender_id}",
"blocked", 1)
                    pipeline.publish('spam', f"User
{sender_username} sent spam message: \"%s\"" %
                                             self.__r.hmget("message:%s" %
message_id, ["text"])[0])
                    print(f"User {sender_username} sent spam
message: \"%s\"" % self.__r.hmget("message:%s" % message_id,
["text"])[0])

self.__neo4j_server.mark_message_as_spam(message_id)
                else:
                    pipeline.hmset(f"message:{message_id}", {
                        'status': 'sent'
                    })
                    pipeline.hincrby(f"user:{sender_id}", "sent",
1)
                    pipeline.sadd(f"sentto:{consumer_id}",
message_id)
                pipeline.execute()

    def stop(self):
        self.__loop = False
```

```
if __name__ == '__main__':
    try:
        loop = True
        workers_count = 5
        workers = []
        for x in range(workers_count):
            worker = Worker(random.randint(0, 3), Neo4jServer())
            worker.setDaemon(True)
            workers.append(worker)
            worker.start()
        while True:
            pass
    except Exception as e:
        View.show_error(str(e))
```

# Результати роботи програми

## Neo4j menu:

```
Neo4j menu
 0: Tagged messages(6.1)
 1: N long relations(6.2)
 2: Shortest way(6.3)
 3: Only spam conversation(6.4)
 4: Tagged messages without relations(6.5)
 5: Exit
Make your choice:
```

## Main menu:

```
Program mode
 0: Neo4j
 1: Emulation(use one time with worker for generate db)
Make your choice: 0
```

## Task 6.1:

```
Neo4j menu
 0: Tagged messages(6.1)
 1: N long relations(6.2)
 2: Shortest way(6.3)
 3: Only spam conversation(6.4)
 4: Tagged messages without relations(6.5)
 5: Exit
Make your choice: 0
Enter tags(work, family)(Enter comma-separated values): work, family
Users:
1: qschneider
2: maurice38
```

## Task 6.2:

```
Neo4j menu
 0: Tagged messages(6.1)
 1: N long relations(6.2)
 2: Shortest way(6.3)
 3: Only spam conversation(6.4)
 4: Tagged messages without relations(6.5)
 5: Exit
Make your choice: 1
Enter n: 2
Pairs of users:
1: ['austinjones', 'qschneider']
2: ['maurice38', 'qschneider']
3: ['austinjones', 'maurice38']
4: ['pughanna', 'apierce']
```

## Task 6.3:

```
Neo4j menu
 0: Tagged messages(6.1)
 1: N long relations(6.2)
 2: Shortest way(6.3)
 3: Only spam conversation(6.4)
 4: Tagged messages without relations(6.5)
 5: Exit
Make your choice: 2
Enter username1(qschneider, maurice38, apierce, austinjones, pughanna): qschneider
Enter username2(qschneider, maurice38, apierce, austinjones, pughanna): austinjones
qschneider ->austinjones
```

## Task 6.4:

```
Neo4j menu
 0: Tagged messages(6.1)
 1: N long relations(6.2)
 2: Shortest way(6.3)
 3: Only spam conversation(6.4)
 4: Tagged messages without relations(6.5)
 5: Exit
Make your choice: 3
Pairs of users:
1: ['austinjones', 'maurice38']
```

**Task 6.5:**

```
Neo4j menu
 0: Tagged messages(6.1)
 1: N long relations(6.2)
 2: Shortest way(6.3)
 3: Only spam conversation(6.4)
 4: Tagged messages without relations(6.5)
 5: Exit
Make your choice: 4
Enter tags(work, family)(Enter comma-separated values): family
Groups of unrelated users:
1: ['qschneider']
2: ['maurice38']
3: ['apierce']
4: ['austinjones']
```

# Висновок

Під час виконання лабораторної роботи я здобув практичні навички створення програм, орієнтованих на використання графової бази даних Neo4J за допомогою мови Python.