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”

Виконав
студент ІІІ курсу
групи КП-82
Мельничук Олексій Геннадійович
(*прізвище, ім’я, по батькові*)

Зарахована
“____” “_____” 20___ р.
викладачем
Петрашенко Андрій Васильович
(*прізвище, ім’я, по батькові*)

варіант № 12

Київ 2021

Мета роботи

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.

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 A відправив повідомлення користувачу B, а B відправив повідомлення C, то довжина зв’язку між A і C 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.

Код програми

Main.py – консольний інтерфейс

```
from ops.neo4 import neo4j
import os
import emulation as emul
from ops.tag import Tag

def start_menu():
    print("MAIN MENU")
    print("1. Launch emulation")
    print("2. Users with tag set")
    print("3. User pairs with N relation")
    print("4. Path between 2 users")
    print("5. Spammer pairs")
    print("6. Unrelated user list with tag set")
    print("0. Exit")
    return int(input("Enter the number of action: "))

def main():
    while True:
        action = start_menu()

        if action == 1:
            os.system('python3 emulation.py')
            print("Emulation completed!\n")

        elif action == 2:
            enums = list(map(lambda c: c.value, Tag))
            newlist = list()
            res = ''
            for e in enums:
                while True:
                    res = input(f'Have "{e[1]}" tag? y or n: ')
                    if res == 'y':
                        newlist.append(e[1].lower())
                        break
                    elif res == 'n':
                        break

            users = neo4j.get_related_u_by_tags(newlist)
            print(f"Users: ")
            iter = 1
            for user in users:
                print(f"{iter}. {user}")
                iter += 1

        elif action == 3:
            n = int(input("Enter length of relations: "))
            users = neo4j.get_u_with_n_relation(n)
            print("User pairs: ")
            iter = 1
            for user in users:
                print(f"{iter}. {user[0]} - {user[1]}")
                iter += 1

        elif action == 4:
            username1 = input("Enter username1: ")
            username2 = input("Enter username2: ")
            way = neo4j.shortest_way(username1, username2)
            text = ""
            print("Shortest path: ")
            for step in way:
```

```
                    text += f"{step} -> "
                print(text[:-3])

            elif action == 5:
                spammers = neo4j.get_spammer_u()
                print("Spammer pairs: ")
                iter = 1
                for user in spammers:
                    print(f"{iter}. {user[0]} - {user[1]}")
                    iter += 1

            elif action == 6:
                enums = list(map(lambda c: c.value, Tag))
                newlist = list()
                res = ''
                for e in enums:
                    while True:
                        res = input(f'Have "{e[1]}" tag? y or n: ')
                        if res == 'y':
                            newlist.append(e[1].lower())
                            break
                        elif res == 'n':
                            break
                unrelated_users = neo4j.get_u_with_tags(newlist)
                print("Messages: ")
                iter = 1
                for user in unrelated_users:
                    print(f"{iter}. {user[0]}")
                    iter += 1

            elif action == 0:
                print("Farewell!")
                break

            else:
                print("Enter correct choice (num 0 to 4): ")


    if __name__ == '__main__':
        main()
```

emulation.py – генерування даних

```
import ops.connection as redis

redis.rconnect()
rconnection = redis.rconnection

from ops.user import User
from ops.message import Message
from ops.tag import Tag
import ops.neo4 as neo4

from threading import Thread
from faker import Faker
from random import randint
import atexit

class Emulation(Thread):
    def __init__(self, name, users):
        Thread.__init__(self)
        self.conn = rconnection
        self.name = name
        self.users = users
        self.user_id = User.register(name)

    def run(self):
        for i in range(amount):
            sentence = fake.sentence(nb_words=5, variable_nb_words=True,
ext_word_list=None)
            receiver = self.users[randint(0, amount - 1)]
            print(f"Message {sentence} was sent to {receiver}")
            Message.create_message(self.user_id, sentence, receiver,
Tag.get_random())

def exit():
    online = rconnection.smembers("online")
    for i in online:
        rconnection.srem("online", i)
        rconnection.publish("logout", f"User {i} signed out.")
        print(f"{i} logged off.")

if __name__ == '__main__':
    fake = Faker()
    atexit.register(exit)
    amount = 5

    users = [fake.profile(fields=["username"], sex=None)["username"] for user in
range(amount)]
    threads = []

    for i in range(amount):
        print(f"User: {users[i]}")
        threads.append(Emulation(users[i], users))

    for t in threads:
        t.start()
```

Worker.py – переглядувач черги + перевірка на спам

```
import ops.config as cfg
import ops.connection as redis
redis.rconnect()
rconnection = redis.rconnection

import ops.neo4 as neo4


from ops.message import Message
from ops.user import User

from threading import Thread
from random import randint
import time
import random

import logging
import datetime
logging.basicConfig(filename="logs.txt", level=logging.INFO)



def is_spam():
    return random.random() > 0.5

class Worker(Thread):
    def __init__(self):
        Thread.__init__(self)

    def run(self):
        message = rconnection.brpop("queue")

        if message:
            message_id = message[1]
            message_key = f"message{message_id}"
            rconnection.hset(message_key, "status", "checking")

            message = rconnection.hmget(message_key, ["sender_id", "receiver_id"])
            sender_id = message[0]
            receiver_id = message[1]
            sender_name = User.get_username(sender_id)

            rconnection.hincrby(f"user{sender_id}", "queue", -1)
            print("Message enqueued")
            rconnection.hincrby(f"user{sender_id}", "checking", 1)
            time.sleep(randint(0, 2))

            pipeline = rconnection.pipeline(True)
            pipeline.hincrby(f"user{sender_id}", "checking", -1)


            if is_spam():
                print(f"{sender_name} sent spam: id={message_id}")

                message_text = rconnection.hmget(message_key, ["text"])[0]
                logging.info(f"({datetime.datetime.now()}): User {sender_name} sent
spam: {message_text}")

                pipeline.zincrby(f"spam", 1, f"user{sender_id}")
                pipeline.hset(message_key, "status", "blocked")
                pipeline.hincrby(f"user{sender_id}", "blocked", 1)
```

```
                        pipeline.publish("spam",    f"User    {sender_name}    sent    spam:
{message_text}.")
                        neo4.neo4j.mark_message_as_spam(message_id)

                    else:
                        print(f"Checked and sent message[{message_id}] from {sender_name}.")
                        pipeline.hset(message_key, "status", "sent")
                        pipeline.hincrby(f"user{sender_id}", "sent", 1)
                        pipeline.sadd(f"sent_to:{receiver_id}", message_id)

                    pipeline.execute()

        def main():
            handlers = 5
            for i in range(handlers):
                worker = Worker()
                worker.daemon = True
                worker.start()

            while True:
                pass

        if __name__ == '__main__':
            main()
```

ops/User.py – операції з користувачами

```
        import ops.connection as redis
        import logging
        import datetime
        import ops.neo4 as neo4
        logging.basicConfig(filename="logs.txt", level=logging.INFO)


        redis.rconnect()
        rconnection = redis.rconnection

        class User:
            def register(username):
                if rconnection.hget("users", username):
                    print(f"{username} already exists.")
                    return -1

                user_id = rconnection.incr("user_id")
                user_key = f"user{user_id}"
                user_info = {
                    "id": user_id,
                    "name": username,
                    "queue": 0,
                    "checking": 0,
                    "blocked": 0,
                    "sent": 0,
                    "delivered": 0
                }

                rconnection.hset("users", username, user_id)
                for key in user_info.keys():
                    rconnection.hset(user_key, key, user_info[key])

                rconnection.publish("register", f"User {username} registered")
```

```
            rconnection.sadd("online", username)


            neo4.neo4j.register(username, user_id)


            logging.info(f"({datetime.datetime.now()}): User {username} registered")
            return user_id

        def login(username):
            user_id = rconnection.hget("users", username)

            if not user_id:
                print(f"{username} does not exist. Register?")
                return -1

            rconnection.publish("login", f"User {username} logged in")
            rconnection.sadd("online", username)

            neo4.neo4j.login(user_id)

            logging.info(f"({datetime.datetime.now()}): User {username} logged in")
            return user_id

        def logout(user_id):
            username = User.get_username(user_id)
            rconnection.publish("logout", f"User {username} logged out")
            rconnection.srem("online", username)

            neo4.neo4j.logout(user_id)

            logging.info(f"({datetime.datetime.now()}): User {username} logged out")


        def get_username(user_id):
            return rconnection.hmget(f"user{user_id}", ["name"])[0]

        def is_logged_in(user_id):
            return user_id != -1
```

ops/Message.py – операції з повідомленнями

```
from ops.tag import Tag
from os import pipe
import ops.connection as redis
import ops.tag as tag
redis.rconnect()
rconnection = redis.rconnection
from ops.user import User

import ops.neo4 as neo4


import random

class Message:
    def create_message(user_id, message, receiver, tags):
        message_id = rconnection.incr("message_id")
        receiver_id = rconnection.hget("users", receiver)
        if not receiver_id:
            print(f"{receiver} does not exist, can't send a message.")
            return False
```

```
            message_key = f"message{message_id}"
            message_info = {
                "id": message_id,
                "text": message,
                "sender_id": user_id,
                "receiver_id": receiver_id,
                "status": "created",
                "tags": ",".join(tags)
            }

            pipeline = rconnection.pipeline(True)

            for key in message_info.keys():
                pipeline.hset(message_key, key, message_info[key])

            pipeline.lpush("queue", message_id)
            pipeline.hset(message_key, "status", "queue")

            pipeline.hincrby(f"user{user_id}", "queue", 1)
            pipeline.zincrby("sent", 1, f"user{user_id}")

            pipeline.execute()

            neo4.neo4j.create_message(user_id, receiver_id, {"id": message_id, "tags":
tags})

            return message_id

        def get_inbox(user_id):
            messages = rconnection.smembers(f"sent_to{user_id}")

            if len(messages) == 0:
                print("No messages")
                return

            for message_id in messages:
                message = rconnection.hmget(f"message{message_id}", ["text", "status",
"sender_id"])

                if message[1] != "delivered":
                    rconnection.hset(f"message{message_id}", "status", "delivered")
                    rconnection.hincrby(f"user{message[2]}", "sent", -1)
                    rconnection.hincrby(f"user{message[2]}", "delivered", 1)

                print(f"{message[0]} -> FROM: {User.get_username(message[2])}")
```

ops/connection.py – конфіг підключення

```
import redis
import sys
import ops.config as cfg
from neo4j import GraphDatabase

rediscfg = cfg.redis
neocfg = cfg.neo4j


rconnection = redis.Redis(host=rediscfg["host"], port=rediscfg["port"], db=0,
decode_responses=True)
nconnection = GraphDatabase.driver("bolt://localhost:7687", auth=(neocfg["user"],
neocfg["pass"]))
```

```
def rconnect():
    try:
        rconnection.ping()
    except Exception as err:
        sys.exit(err)
```

ops/tag.py – набір тегів для повідомлень

```
import enum
from random import randint, choice

class Tag(enum.Enum):
    family = (1, "Family")
    private = (2, "Private")
    work = (3, "Work")
    group = (4, "Group")
    news = (5, "News")
    public = (6, "Public")

    def get_member(data):
        return data in Tag._member_names_

    def get_random():
        tags = []
        num = randint(0, len(Tag))
        for i in range(num):
            tag = choice(list(Tag)).name
            if tag not in tags:
                tags.append(tag)
        return tags
```

ops/neo4.py – створення запитів Cypher до бд Neo4J

```
from neo4j import GraphDatabase
import ops.tag as Tag
import ops.config as cfg

neo4j = cfg.neo4j

neoconnection = GraphDatabase.driver("bolt://localhost:7687", auth=(neo4j["user"],
neo4j["pass"]))


class Neo4j:
    def __init__(self):
        self.__driver = neoconnection

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

    def register(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 login(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 logout(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, user_id, receiver_id, message: dict):
                with self.__driver.session() as session:
                    try:
                        messages_id                                                    =
session.write_transaction(self.create_message_relation, int(user_id),
                                                                int(receiver_id),
message["id"])
                        for tag in message["tags"]:
                            session.write_transaction(self.add_tags_to_messages,
messages_id, tag)
                    except Exception as e:
                        print(str(e))


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

            def add_tags_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_related_u_by_tags(self, tags):
                # print(tags)
                res = self.get_users_by_tags_from_db(tags)
                ress = [record for record in res.data()]
                newl = []
                for line in ress:
                    newl.append(line.get('u')["name"])
                newl = list(dict.fromkeys(newl))
                return newl

            def get_users_by_tags_from_db(self, tags):
                for tag in tags:
```

```
                    if not Tag.Tag.get_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"
                # print(query)
                return self.__driver.session().run(query)




            def get_u_with_tags(self, tags):
                list_of_names  =  self.record_to_list(self.get_users_by_tags_from_db(tags),
'name')
                unrelated_users = []
                for name1 in list_of_names:
                    group = [name1]
                    for name2 in list_of_names:
                        if name1 != name2:
                            res = self.check_u_relation(name1, name2)
                            if not res and name1 not in group:
                                group.append(name2)
                    unrelated_users.append(group)
                return unrelated_users

            def check_u_relation(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(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_u_with_n_relation(self, n):
                with self.__driver.session() as session:
                    res = session.run(f"MATCH p = (u1:user)-[*{n}]-(u2:user) "
                                      f"WHERE u1 <> u2 "
                                      f"RETURN u1, u2")
                    return self.pair_to_list(res, 'name')
```

```
            def get_spammer_u(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_to_list(res, 'name')


            def pair_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):
                # for record in res:
                #     print(record["name"])
                my_list = list(res)
                my_list = list(dict.fromkeys(my_list))
                # print(my_list)
                return [el[0]._properties[pull_out_value] for el in my_list]

        neo4j = Neo4j()
```

ops/config.py – конфіг підключення до Redis та Neo4J

```
        neo4j = {
            "user": "neo4j",
            "pass": "fractal-clara-random-avatar-little-3997"
        }

        redis = {
            "host": 'localhost',
            "port": 6379
        }
```

Результати роботи програм

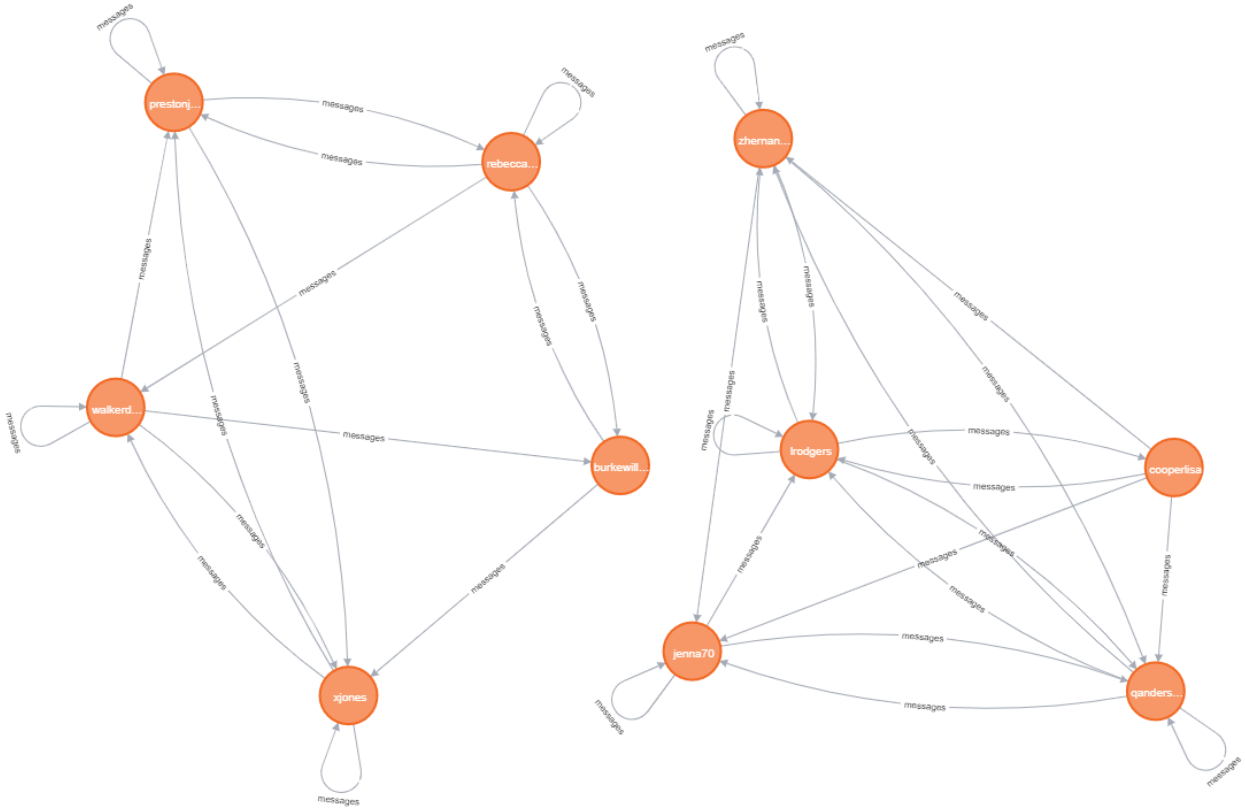

Рис. 1. Графова структура БД після запуску двох емуляцій

```
MAIN MENU
1. Launch emulation
2. Users with tag set
3. User pairs with N relation
4. Path between 2 users
5. Spammer pairs
6. Unrelated user list with tag set
0. Exit
Enter the number of action: 2
Have "Family" tag? y or n: y
Have "Private" tag? y or n: n
Have "Work" tag? y or n: y
Have "Group" tag? y or n: y
Have "News" tag? y or n: n
Have "Public" tag? y or n: n
Users:
1. jenna70
2. lrodgers
3. zhernandez
4. rebeccaswanson
5. burkewilliam
MAIN MENU
```

Рис. 2. Пошук користувачів що відправили/прийняли повідомлення з набором тегів

```
MAIN MENU
1. Launch emulation
2. Users with tag set
3. User pairs with N relation
4. Path between 2 users
5. Spammer pairs
6. Unrelated user list with tag set
0. Exit
Enter the number of action: 3
Enter length of relations: 4
User pairs:
1. jenna70 - qanderson
2. cooperlisa - qanderson
3. lrodgers - qanderson
4. zhernandez - qanderson
5. cooperlisa - jenna70
6. zhernandez - jenna70
7. lrodgers - jenna70
8. lrodgers - cooperlisa
9. zhernandez - cooperlisa
10. zhernandez - lrodgers
11. xjones - rebeccaswanson
12. walkerdavid - rebeccaswanson
```

Рис. 3. Пари користувачів зі зв’язком N

```
MAIN MENU
1. Launch emulation
2. Users with tag set
3. User pairs with N relation
4. Path between 2 users
5. Spammer pairs
6. Unrelated user list with tag set
0. Exit
Enter the number of action: 4
Enter username1: rebeccaswanson
Enter username2: xjones
Shortest path:
rebeccaswanson -> prestonjones -> xjones
MAIN MENU
```

Рис. 4. Найкоротший шлях між rebeccaswanson та xjones

```
MAIN MENU
1. Launch emulation
2. Users with tag set
3. User pairs with N relation
4. Path between 2 users
5. Spammer pairs
6. Unrelated user list with tag set
0. Exit
Enter the number of action: 5
Spammer pairs:
1. cooperlisa - qanderson
2. lrodgers - qanderson
3. zhernandez - qanderson
4. lrodgers - cooperlisa
5. zhernandez - lrodgers
MAIN MENU
```

Рис. 5. Пари спамерів

```
MAIN MENU
1. Launch emulation
2. Users with tag set
3. User pairs with N relation
4. Path between 2 users
5. Spammer pairs
6. Unrelated user list with tag set
0. Exit
Enter the number of action: 6
Have "Family" tag? y or n: n
Have "Private" tag? y or n: n
Have "Work" tag? y or n: y
Have "Group" tag? y or n: y
Have "News" tag? y or n: y
Have "Public" tag? y or n: y
Messages:
1. qanderson
2. jenna70
3. zhernandez
4. prestonjones
5. xjones
MAIN MENU
```

Рис. 6. Непов’язані користувачі з набором тегів

Відповіді на контрольні запитання

1) Визначити сфери застосування графової бази даних Neo4J

- Аналіз та створення рекомендацій
- Аналіз на шахрайство
- Керування мастер-даними
- Побудова графів для соціальних мереж
- Телекомунікації
- Торгівля та керування ланцюгами доставок

2) Пояснити призначення ключових слів MATCH, WHERE, RETURN мови запитів Cypher.

MATCH – шаблон за яким буде здійснюватись пошук Neo4J по бд

WHERE – у зв’язці з MATCH: додання уточнень/обмежень до пошуку, у зв’язці з WITH: фільтрація результату пошуку

RETURN – визначення потрібних частин з результату пошуку

Висновки

Виконавши дану лабораторну роботу я ознайомився з особливостями та принципом роботи графової бази даних Neo4J та мовою запитів Cypher, використав здобуті знання для проектування графової бази даних та виконання аналізу отриманих даних.