

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 III курсу  групи КП-82  Мельничук Олексій Геннадійович  (*прізвище, ім’я, по батькові*)  варіант № 12 |  |  | Зарахована  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладачем  Петрашенко Андрій Васильович (*прізвище, ім’я, по батькові*) |

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.
   1. Задано список тегів (tags). Знайти всіх користувачів, що відправили або отримали повідомлення з набором тегів tags.
   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 С, то довжина зв’язку між А і С є N=2.
   3. Задано два користувача. Знайти на графі найкоротший шлях між ними через відправлені або отримані повідомлення.
   4. Знайти авторів повідомлень, які пов’язані між собою лише повідомленнями, позначеними як “спам”.
   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**

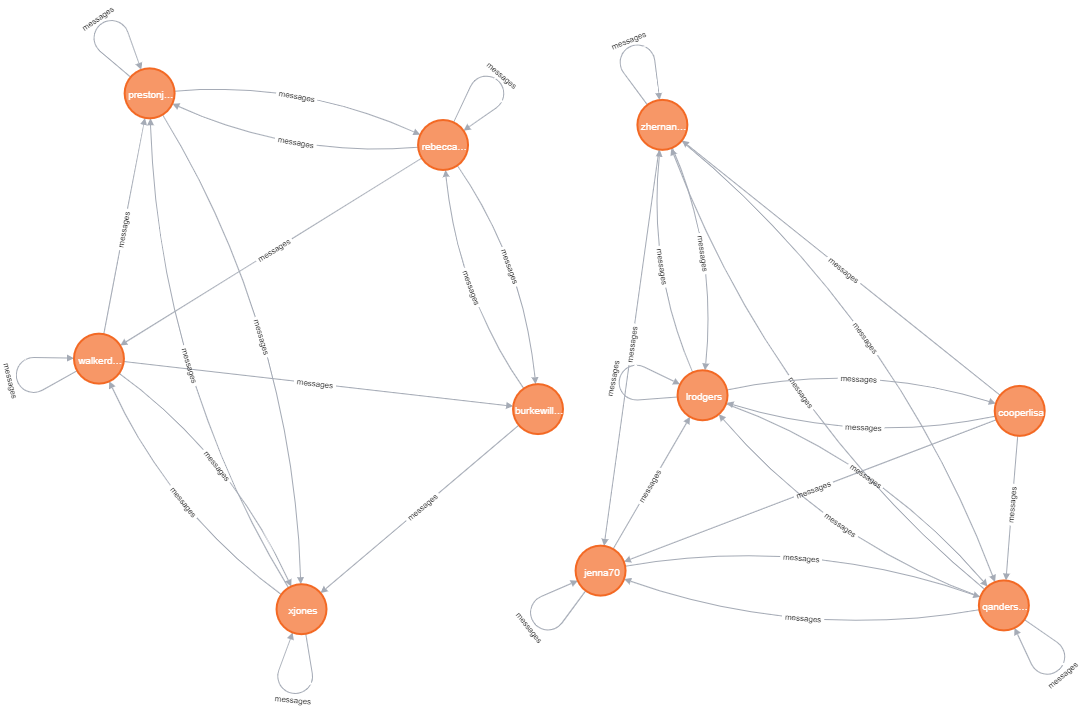
****

Рис. 1. Графова структура БД після запуску двох емуляцій

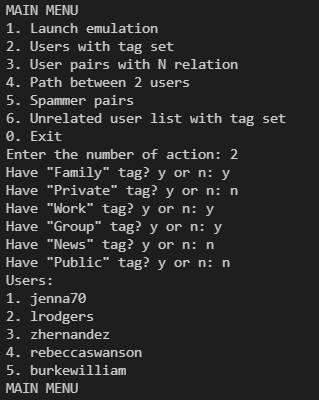


Рис. 2. Пошук користувачів що відправили/прийняли повідомлення з набором тегів

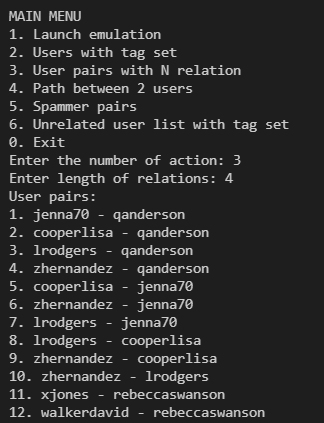


Рис. 3. Пари користувачів зі зв’язком N

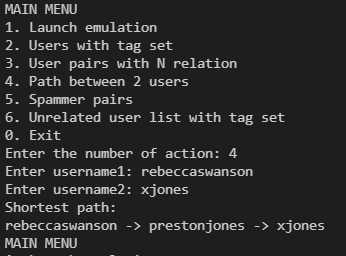


Рис. 4. Найкоротший шлях між rebeccaswanson та xjones

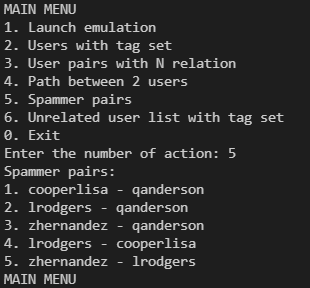


Рис. 5. Пари спамерів

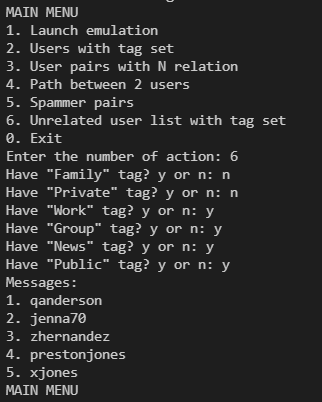
****

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

1. 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.