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 № 2

з дисципліни “Бази даних”

Виконав
Студент 2 курсу
групи КП-03
Шалак Данило Володимирович
(*прізвище, ім’я, по батькові*)

Перевірив
“____” “_____” 20___р.
викладач
Радченко Костянтин Олександрович
(*прізвище, ім’я, по батькові*)

варіант

Київ 2021

Мета

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Завдання

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Хід роботи

Тексти коду програми:

app.py

```
import psycopg2
from controller.controller import Controller
from model.postsRepository import PostsRepository
from model.commentRepository import CommentsRepository
from model.usersRepository import UsersRepository
from view import View


conn = psycopg2.connect(
   host="localhost",
   database="bd_labs",
   user="postgres",
```

```
   password="danylo1")



postRepo = PostsRepository(conn)
commentRepo = CommentsRepository(conn)
userRepo = UsersRepository(conn)
view = View()


contr = Controller(postRepo, userRepo, commentRepo, view)


while(True):
   command = input('Enter command\n')
   if command == 'exit':
       break


   try:
       contr.handleCommand(command)
   except Exception as e:
       print(e)
```

view.py

```
class View:
   def perform_log(self, command):
       print('Command `{0}` successfuly performed'.format(command))

   def fetch_data_log(self, titles, data):
       t = ""
       for title in titles:
           t += "{:25}".format(title)
       print(t)

       for row in data:
           str_row = ""
           for el in row:
               str_row += "{:25}".format(str(el))
           print(str_row)
```

querydata.py

```
from datetime import date
import datetime
```

```
def get_query_data(command):
   if command.startswith('delete') or command.startswith('generate'):
       return get_id_paramData(command)
   if command.startswith('update'):
       return get_update_data(command)
   if command.startswith('insertUser'):
       return get_userData(command)
   if command.startswith('insertComment'):
       return get_commentData(command)
   if command.startswith('insertPost'):
       return get_postData(command)
   if command.startswith('analytics'):
       return get_analytics_data(command)
   if command.startswith('activity'):
       return get_activity_data(command)
   if command.startswith('userPost'):
       return get_user_post_data(command)



def get_update_data(command):
   data = command.split(' ')
   if len(data) != 3:
       raise Exception("Incorrect entered command '{0}'".format(data[0]))
   if len(data[1]) == 0:
       raise Exception("String parameter can not be empty")
   updateId = get_int_param(data[2])
   return (data[1].replace('-', ' '), updateId,)


def get_id_paramData(command):
   data = command.split(' ')
   if len(data) != 2:
       raise Exception("Wrong entered command '{0}'".format(data[0]))

   deleteId = get_int_param(data[1])
   return (deleteId, )


def get_userData(command):
   data = command.split(' ')
   if len(data) != 3:
       raise Exception("Wrong entered command 'insertUser'")
   if len(data[1]) == 0 or len(data[2]) == 0:
       raise Exception("String parameter can not be empty")

   return (data[1], data[2].replace('_', ' '), get_strDate(), )


def get_commentData(command):
   data = command.split(' ')
   if len(data) != 4:
       raise Exception("Wrong entered command 'insertComment'")
   if len(data[1]) == 0:
```

```
        raise Exception("String parameter can not be empty")
    postId = get_int_param(data[2])
    authorId = get_int_param(data[3])
    return (data[1].replace('_', ' '), postId, authorId, get_strDate(), )


def get_postData(command):
    data = command.split(' ')
    if len(data) != 4 and len(data) != 3:
        raise Exception("Incorrect entered command 'insertPost'")
    if len(data[1]) == 0:
        raise Exception("String parameter can not be empty")
    authorId = get_int_param(data[2])
    date = get_strDate()
    tags = []
    if len(data) == 4 and len(data[3]) != 0:
        tags = get_tags(data[3])
    return (data[1].replace('_', ' '), authorId, date, tags)


def get_analytics_data(command):
    data = command.split(' ')
    if len(data) != 3 and len(data) != 4:
        raise Exception("Incorrect entered command 'analytics'")
    posible_parameters = ['desc', 'asc']
    if data[1] not in posible_parameters or data[2] not in
posible_parameters:
        raise Exception("Incorrect entered 'analytics' filter parameters")
    lim = 'all'
    if len(data) == 4:
        lim = get_int_param(data[3])
    return (data[1], data[2], lim, )


def get_activity_data(command):
    data = command.split(' ')
    if len(data) != 4 and len(data) != 5:
        raise Exception("Incorrect entered command 'analytics'")
    posible_parameters = ['desc', 'asc']
    if data[1] not in posible_parameters or data[2] not in
posible_parameters:
        raise Exception("Incorrect entered 'analytics' filter parameters")
    lim = 'all'
    if len(data) == 5:
        lim = get_int_param(data[4])
    date_ = date.fromisoformat(data[3])
    return (data[3], data[1], data[2], lim,)


def get_user_post_data(command):
    data = command.split(' ')
    if len(data) != 4 and len(data) != 5:
        raise Exception("Incorrect entered command 'analytics'")
    posible_parameters = ['desc', 'asc']
```

```
   if data[2] not in posible_parameters or data[3] not in
posible_parameters:
       raise Exception("Incorrect entered 'analytics' filter parameters")
   lim = 'all'
   if len(data) == 5:
       lim = get_int_param(data[4])
   userId = get_int_param(data[1])
   return (userId, data[2], data[3], lim, )


def get_int_param(rawInt):
   number = 0
   try:
       number = int(rawInt)
   except :
       raise Exception("Incorrect type of integer parameter")
   return number

def get_strDate():
   date = datetime.datetime.now()
   strDate = str(date.year) + '-' + str(date.month) + '-' + str(date.day)
   return strDate


def get_tags(tagData):
   if tagData.startswith('tag:') == False:
       raise Exception('Incorrect entered tag data')
   tags = tagData.replace('tag:', '').split('#')
   return tags
```

commentRepository.py

```
import psycopg2

class CommentsRepository:

   def __init__(self, connection):
       self.connection = connection;

   def insertComment(self, commentData):
       query = """INSERT INTO comments (comment_content, post_id,
author_id,
        publication_date) VALUES (%s, %s, %s, %s)"""

       cursor = self.connection.cursor()

       cursor.execute(query, commentData)

       cursor.close()
       self.connection.commit()
```

```
   def updateComment(self, commentData):
       query = """UPDATE comments
                   SET comment_content = %s
                   WHERE comment_id = %s"""
       cursor = self.connection.cursor()
       cursor.execute(query, commentData)

       cursor.close()
       self.connection.commit()


   def deleteComment(self, commentData):
       query = """DELETE FROM comments WHERE comment_id = %s"""
       cursor = self.connection.cursor()
       cursor.execute(query, commentData)

       cursor.close()
       self.connection.commit()


   def generateComments(self, commentsCount):
       query = """insert into comments (comment_content, post_id,
            author_id, publication_date)

       With id_table(post_id, author_id) AS
       (
           select posts.post_id, users.user_id from posts, users
       ),
       generated_data(content, publication_date) AS
       (
           select
               chr(trunc(65+random()*25)::int) ||
               chr(trunc(65+random()*25)::int) as fullname,
               (timestamp '2000-01-01' + random() *
           (timestamp '2021-10-10' - timestamp '2000-01-01'))::date
               from generate_series(1, %s) seq
       )

       select generated_data.content, id_table.post_id,
id_table.author_id,
       generated_data.publication_date from id_table, generated_data
       limit %s"""

       cursor = self.connection.cursor()
       cursor.execute(query, (commentsCount[0], commentsCount[0],))

       cursor.close()
       self.connection.commit()
```

postsRepository.py

```
import psycopg2

class PostsRepository:

   def __init__(self, connection):
       self.connection = connection;

   def insertPost(self, data):
       query = """INSERT INTO posts (post_content, author_id,
publication_date)
       VALUES (%s, %s, %s) RETURNING post_id"""

       cursor = self.connection.cursor()

       postData = data[:3]

       cursor.execute(query, postData)
       postId = cursor.fetchall()[0][0]

       cursor.close()
       self.connection.commit()

       self.subscribeTags(data[3], postId)



   def updatePost(self, postData):
       query = """UPDATE posts
                   SET post_content = %s
                   WHERE post_id = %s"""
       cursor = self.connection.cursor()
       cursor.execute(query, postData)

       cursor.close()
       self.connection.commit()


   def deletePost(self, postData):
       query = """DELETE FROM posts WHERE post_id = %s"""
       cursor = self.connection.cursor()
       cursor.execute(query, postData)

       cursor.close()
       self.connection.commit()


   def is_tag_not_exists(self,tag):
       checkQuery = """SELECT COUNT(*) FROM tags WHERE tag_name = %s"""
       cursor = self.connection.cursor()
       cursor.execute(checkQuery, (tag, ))
       count = cursor.fetchall()[0][0]
       cursor.close()
```

```
        return count == 0


    def subscribeTags(self, tags, postId):
        cursor = self.connection.cursor()
        insertQuery = """INSERT INTO tags (tag_name) VALUES (%s)
                         RETURNING tag_id"""

        idQuery = """SELECT (tag_id) FROM tags WHERE tag_name = %s"""

        subscribeQuery = """INSERT INTO posts_tags (post_id, tag_id)
                        VALUES (%s, %s)"""

        tag_name_id = 1

        for tag in tags:
            if self.is_tag_not_exists(tag) == True:
                cursor.execute(insertQuery, (tag, ))
                tag_name_id = cursor.fetchall()[0][0]
                self.connection.commit()
            else:
                cursor.execute(idQuery, (tag, ))
                tag_name_id = cursor.fetchall()[0][0]

            cursor.execute(subscribeQuery, (postId, tag_name_id, ))
            self.connection.commit()

        cursor.close()


    def generatePosts(self, postsCount):
        query = """insert into posts (post_content, author_id,
publication_date)

        With id_table(user_id) AS
        (
            select user_id from users
        ),
        generated_data(content, publication_date) AS
        (
            select
                chr(trunc(65+random()*25)::int) ||
                chr(trunc(65+random()*25)::int) as fullname,
                (timestamp '2000-01-01' + random() *
                 (timestamp '2021-10-10' - timestamp '2000-01-01'))::date
                from generate_series(1, %s) seq
        )


        select generated_data.content, id_table.user_id,
        generated_data.publication_date from id_table,
        generated_data  order by random() limit %s"""
```

```
        cursor = self.connection.cursor()
        cursor.execute(query, (postsCount, postsCount,))
        cursor.close()
        self.connection.commit()


    def get_users_posts(self, queryData):
        query = """select posts.post_id, posts.post_content,
         posts.publication_date, count(comments.comment_id) as
comment_count
        from posts
        join comments on posts.post_id = comments.post_id and
posts.author_id = %s
        group by posts.post_id
        order by comment_count {0}, publication_date {1}
        limit {2}""".format(queryData[1], queryData[2], queryData[3])

        cursor = self.connection.cursor()
        cursor.execute(query, (queryData[0], ))
        data = cursor.fetchall()
        cursor.close()

        return data
```

usersRepository.py

```
from typing import Counter
import psycopg2

class UsersRepository:

    def __init__(self, connection):
        self.connection = connection;

    def insertUser(self, userData):
        query = """INSERT INTO users (login, fullname, registration_date)
        VALUES (%s, %s, %s)"""

        cursor = self.connection.cursor()

        cursor.execute(query, userData)

        cursor.close()
        self.connection.commit()


    def renameUser(self, userData):
        query = """UPDATE users
                    SET fullname = %s
                    WHERE user_id = %s"""
        cursor = self.connection.cursor()
```

```
        cursor.execute(query, userData)

        cursor.close()
        self.connection.commit()


    def deleteUser(self, userData):
        query = """DELETE FROM users WHERE user_id = %s"""
        cursor = self.connection.cursor()
        cursor.execute(query, userData)

        cursor.close()
        self.connection.commit()


    def generateUsers(self, usersCount):
        query = """insert into users (login, fullname, registration_date)
                    select
                        'user_' || seq as login,
                        chr(trunc(65+random()*25)::int) ||
                        chr(trunc(65+random()*25)::int) as fullname,
                        (timestamp '2000-01-01' + random() *
                        (timestamp '2021-10-10' - timestamp
'2000-01-01'))::date
                        from generate_series(1, %s) seq;"""

        cursor = self.connection.cursor()
        cursor.execute(query, (usersCount, ))
        self.connection.commit()


    def getAnalytics(self, queryData):
        query = """
        with comments_ (user_id, comment_count) as
        (
            select  users.user_id, count(comments.comment_id)
            from users
            join comments on comments.author_id = users.user_id
            group by users.user_id
        ),
        posts_ (user_id, login, post_count) as
        (
            select users.user_id, users.login , count(posts.post_id) as
posts_count
            from users
            join posts on posts.author_id = users.user_id
            group by users.user_id, users.login
        )

        select posts_.user_id, posts_.login, posts_.post_count,
comments_.comment_count
        from posts_
        join comments_ on comments_.user_id = posts_.user_id
        order by post_count {0}, comment_count {1}
```

```
        limit {2}""".format(queryData[0], queryData[1], queryData[2])

        cursor = self.connection.cursor()
        cursor.execute(query)
        data = cursor.fetchall()

        return data


    def getActivity(self, activityData):
        query = """with comments_ (user_id, comment_count) as
        (
            select  users.user_id, count(comments.comment_id)
            from users
            join comments on comments.author_id = users.user_id
            group by users.user_id
        ),
        posts_ (user_id, login, reg_date, post_count) as
        (
            select users.user_id, users.login, users.registration_date,
                 count(posts.post_id) as posts_count
            from users
            join posts on posts.author_id = users.user_id
            group by users.user_id, users.login
        )

        select posts_.user_id, posts_.login, posts_.reg_date,
posts_.post_count + comments_.comment_count as points
        from posts_
        join comments_ on comments_.user_id = posts_.user_id and
posts_.reg_date > %s
        order by points {0}, posts_.reg_date {1}

        limit {2}""".format(activityData[1], activityData[2],
activityData[3])

        cursor = self.connection.cursor()
        cursor.execute(query, (activityData[0], ))
        data = cursor.fetchall()

        return data
```

controller.py

```
import querydata
import time

class Controller:
    def __init__(self, pRepo, uRepo, cRepo, view):
        self.usersRepository = uRepo
        self.postsRepository = pRepo
```

```
       self.commentsRepository = cRepo
       self.view = view
       self.controller_dict = {}
       self.controller_dict["insertPost"] =
self.postsRepository.insertPost
       self.controller_dict["insertComment"] =
self.commentsRepository.insertComment
       self.controller_dict["insertUser"] =
self.usersRepository.insertUser
       self.controller_dict["deleteUser"] =
self.usersRepository.deleteUser
       self.controller_dict["deletePost"] =
self.postsRepository.deletePost
       self.controller_dict["deleteComment"] =
self.commentsRepository.deleteComment
       self.controller_dict["generateUser"] =
self.usersRepository.generateUsers
       self.controller_dict["generatePost"] =
self.postsRepository.generatePosts
       self.controller_dict["generateComment"] =
self.commentsRepository.generateComments
       self.controller_dict["updateUser"] =
self.usersRepository.renameUser
       self.controller_dict["updatePost"] =
self.postsRepository.updatePost
       self.controller_dict["updateComments"] =
self.commentsRepository.updateComment
       self.controller_dict["analytics"] =
self.usersRepository.getAnalytics
       self.controller_dict["activity"] = self.usersRepository.getActivity
       self.controller_dict["userPost"] =
self.postsRepository.get_users_posts



   def handleCommand(self,command):
       commands = ["insertPost", "insertComment", "insertUser",
        "deletePost", "deleteComment", "deleteUser", "updatePost",
        "updateUser", "updateComment", "generateComment", "generatePost",
        "generateUser", "analytics", "activity", "userPost"]
       commandName = command.split(' ')[0]

       titles = {"analytics" : ["user_id", "login", "posst_count",
"comment_count"],
       "activity" : ["user_id", "login", "registration_date", "points"],
       "userPost" : ["post_id", "post_content", "publish_date",
"comment_count"]}

       if commandName not in commands:
           raise Exception("Command '{0}' do not
exists".format(commandName))

       commandData = querydata.get_query_data(command)
```

```
        if commandName in ["analytics", "activity", "userPost"]:
            start = time.time()
            fetched_data = self.controller_dict[commandName](commandData)
            end = time.time()
            print("Query time = {0} ms".format(get_time(start, end)))
            self.view.fetch_data_log(titles[commandName], fetched_data)
        else:
            self.controller_dict[commandName](commandData)
            self.view.perform_log(command)


def get_time(start, end):
   elapsed = int((end - start) * 1000)
   return elapsed
```

Команди консольного інтерфейсу:

Додавання нових сутностей в базу даних(при розділенні слів в параметрах “content” та “fullname” використовувати “_”)
sertPost content author_id tag:tag#tag1
insertComment content post_id author_id
insertUser login fullname

Видалення сутностей з БД за id сутності
deletePost id
deleteComment id
deleteUser id

Оновлення даних сутностей за id сутності(при розділенні слів в параметрах “content” та “fullname” використовувати “_”)
updatePost content id
updateComment content id
updateUser fullname id

Генерація сутностей
generateComment number
generateUser number
generatePost number

Команди запиту до БД
analytics asc/desc asc/desc limitNum -- показує аналітику по користувачах (кількість постів, коментарів користувача) перший параметр впорядковує по кількості постів, другий -- по кількості коментарів, limNum - кількість записів, які буде відібрано, за замовчуванням стоїть all
userPost user_id asc/desc asc/desc limitNum -- показує пости користувача з id user_id, перший параметр впорядковує по кількості коментарів під

постом, другий -- за датою публікації постів, limNum - кількість записів, які буде відібрано, за замовчуванням стоїть al
activity asc/desc asc/desc date limitNum -- показує інформацію про користувачів, які зареєструвались починаючи з певної дати, кількість очок активності, перший параметр впорядковує по кількості очок, другий -- по даті реєстрації користувача, limNum - кількість записів, які буде відібрано, за замовчуванням стоїть all

Приклади роботи програми:

Користувач ввів неправильно id поста, який видаляється

```
Enter command
deletePost kjfnkslnfd
Incorrect type of integer parameter
```

Користувач ввів не повністю дані команди(не ввів id автора коментаря)

```
Enter command
insertComment Hello_world 33
Wrong entered command 'insertComment'
```

Приклад згенерованих користувачів в БД (generateUser 10)

16	119	user_1	DL	2012-06-03
17	120	user_2	HP	2013-12-08
18	121	user_3	CX	2002-04-20
19	122	user_4	AX	2020-04-07
20	123	user_5	BT	2001-11-02
21	124	user_6	GL	2011-06-17
22	125	user_7	IO	2007-01-05
23	126	user_8	DK	2003-08-26
24	127	user_9	OT	2009-06-02
25	128	user_10	GS	2021-06-27

```
Enter command
generateUser 10
Command `generateUser 10` successfuly performed
```

Лог в консолі програми

Приклад пошукового запиту(userPost 3 desc asc 10)

```
Enter command
userPost 3 desc asc 10
Query time = 27 ms
post_id                 post_content            publish_date            comment_count
20                      Hello world!            2020-10-12              18
2897                    GI                      2014-08-21              17
18                      Hello world!            2020-10-12              17
679                     WD                      2000-09-02              16
2222                    VY                      2001-05-10              16
1091                    GM                      2001-05-18              16
1059                    EQ                      2001-09-25              16
2575                    AA                      2001-12-06              16
90                      YC                      2003-03-24              16
980                     TE                      2003-10-14              16
Enter command
```

Структура лабораторної роботи в github

Висновки

В ході лабораторної роботи я навчився виконувати SQL запити до бази даних, використовуючи мову python та psycopg2. Навчився створювати складні запити, де беруться дані з декількох таблиць, фільтрувати запити за певними параметрами, генерувати сутності, які мають зв’язки з іншими таблицями.