



**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ
ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп'ютерних
систем**

Лабораторна робота №2

**з дисципліни Бази даних і засоби управління
на тему: “Створення додатку бази даних, орієнтованого на взаємодію з
СУБД PostgreSQL”**

Виконала:
студентка III курсу
групи KB-94
Романенко М. В.
Перевірів:
Петрашенко А. В.

Київ – 2021

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

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, 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).

Деталізоване завдання:

1. Забезпечити можливість введення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв'язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглий таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом!**
3. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість введення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних

необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

4. Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](https://github.com/marromanenko/data-base-course). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL (без ORM).

URL репозиторію з вихідним кодом

<https://github.com/marromanenko/data-base-course>

Модель «сутність-зв'язок» галузі постів якоїсь соціальної мережи

Entities

Post

User

Comments

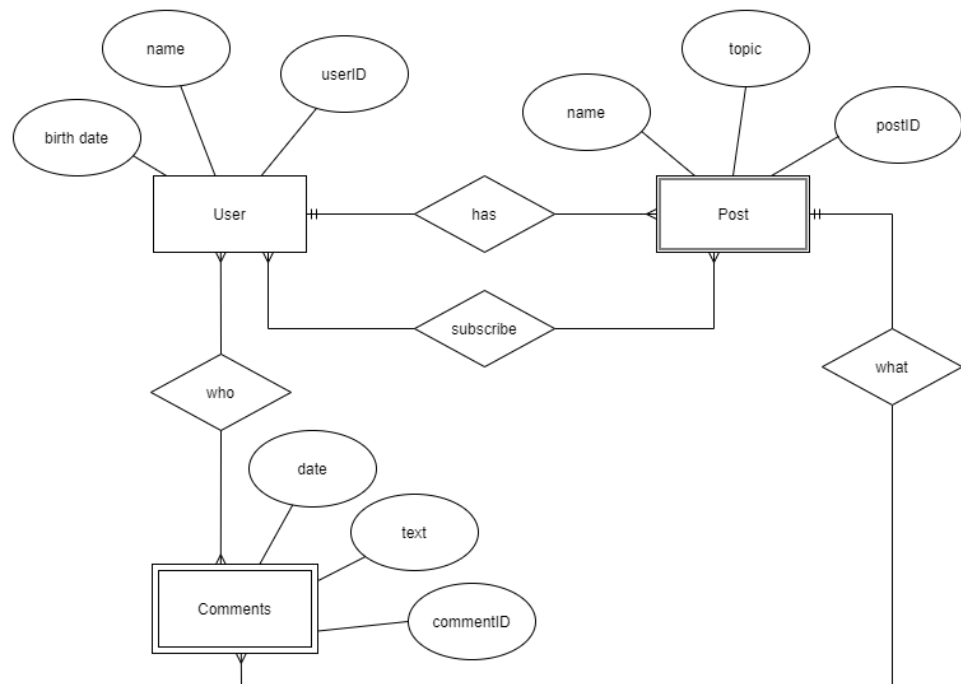


Рисунок 1. ER-діаграма, побудована за нотацією "Пташиної лапки"

Сутності, їх призначення та опис зв'язків:

Маємо три сутності: User, Post, Comments. Сутність User описує користувачів соціальної мережи. Кожен користувач має ім'я, дату народження та свій ID. Сутність Post – це пости цих самих користувачів. Тобто кожен окремий користувач може мати скільки завгодно постів в своєму блозі, але скільки завгодно користувачів можуть бути підписаними на ці пости. Кожний пост також має свій ID, назву поста та тему, на яку написано пост. Третя сутність – це Comments. Це коментарі під постами. Коментар має два зв'язки: чий коментар (користувача) та що це за коментар (тобто під яким саме постом).

Багато користувачів можуть написати скільки завгодно коментарів, а також під одним постом може бути багато коментарів. Коментар має свій ID, свою дату та сам текст.

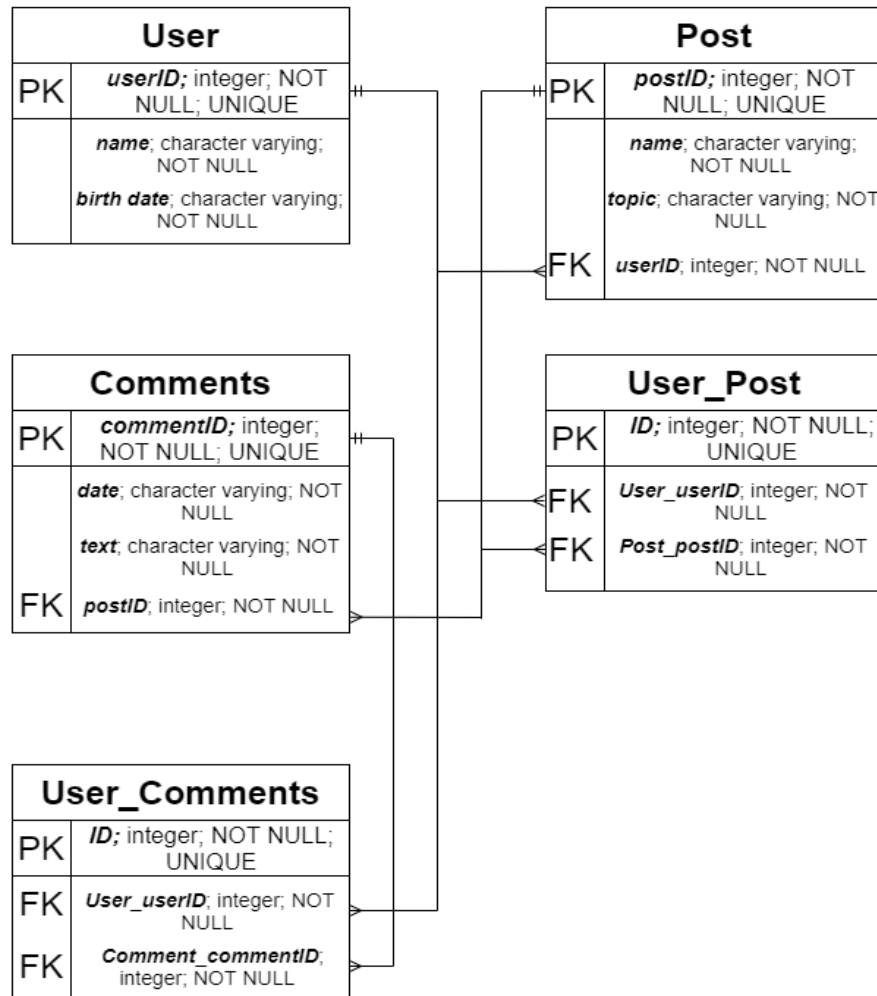


Рисунок 2. Схема бази даних у графічному вигляді

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.7

Середовище розробки програмного забезпечення – PyCharm Community Edition.

Структура меню програми

```
Welcome!

Main menu
0 => Show one table
1 => Show all table
2 => Insert data
3 => Delete data
4 => Update data
5 => Select data
6 => Randomize data
7 => Exit
Make your choice =>
```

Завдання 1

Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.

Перегляд однієї таблиці

```

Make your choice => 0

1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 2
Post
SQL query => select * from public."Post"

-----

Post ID = 1
User ID = 1
Name = family
Topic = about the relationships with family
-----

Post ID = 2
User ID = 2
Name = work
Topic = about some projects and other stuff at work
-----

```

```

Post ID = 3
User ID = 3
Name = friends
Topic = about the closest people after family
-----

Continue work with DB? 1 - Yes; 2 - No. = >|

```

Перегляд всіх таблиць

```
Make your choice => 1
User
SQL query =>  select * from public."User"

-----
ID = 1
Name = Maria
Birth date = 2002-04-14
-----
ID = 2
Name = Gleb
Birth date = 2002-07-26
-----
ID = 3
Name = Sofia
Birth date = 2001-10-03
-----
Post
SQL query =>  select * from public."Post"

-----
Post ID = 1
User ID = 1
Name of the post = family
Topic = about the relationships with family
-----
```

```
Post ID = 2
User ID = 2
Name of the post = work
Topic = about some projects and other stuff at work
```

```
-----
Post ID = 3
User ID = 3
Name of the post = friends
Topic = about the closest people after family
```

```
-----
Comments
SQL query => select * from public."Comments"
```

```
-----
Comment ID = 1
Post ID = 1
Date = 2019-09-04
Text = amasing!
```

```
-----
Comment ID = 2
Post ID = 2
Date = 2020-10-15
Text = awful
```

```
-----
Comment ID = 3
Post ID = 3
Date = 2021-04-26
Text = your cats are very beautiful
```

```
-----
```


User_Comments

SQL query => select * from public."User_Comments"

User = 1

Comment = 1

ID = 1

User = 2

Comment = 2

ID = 2

User = 3

Comment = 3

ID = 3

User_Post

SQL query => select * from public."User_Post"

User = 1

Post = 1

ID = 1

User = 2

Post = 2

ID = 2

```

-----
User = 3
Post = 3
ID = 3
-----
Continue work with DB? 1 - Yes; 2 - No. = >|

```

Внесення даних

Якщо такий первинний ключ вже є, виводиться помилка, яка каже користувачу про це:

```

Make your choice => 1
1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 3
Comment ID = 3
Date = 2021-11-14
Text = comment
PostID = 2
Comments
SQL query => DO $$ BEGIN IF EXISTS (select "postID" from "Post" where "postID" = '2') and not exists (select "commentID" from "Comments" where "commentID" = '3') THEN INSERT INTO
['ЗАМЕЧАНИЕ: This Comment ID already exists or this Post ID does not exist\n']
Continue insertion? 1 - Yes; 2 - No =>
Incorrect input, try again.
Continue insertion? 1 - Yes; 2 - No =>|

```

Якщо такого первічного ключа немає, але також немає такого вторинного ключа, то також виводиться помилка, яка каже користувачеві про це (після виведення помилок програма не зупиняється):

```

Continue insertion? 1 - Yes; 2 - No =>
1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 4
Comment ID = 4
Date = 2021-11-14
Text = comment
PostID = 9
Comments
SQL query => DO $$ BEGIN IF EXISTS (select "postID" from "Post" where "postID" = '9') and not exists (select "commentID" from "Comments" where "commentID" = '4') THEN INSERT INTO
['ЗАМЕЧАНИЕ: This Comment ID already exists or this Post ID does not exist\n']
Continue insertion? 1 - Yes; 2 - No =>|

```

Якщо з ключами все добре, програма працює без помилок:





```

Continue insertion? 1 - Yes; 2 - No =>

1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 3
Comment ID = 4
Date = 2021-11-15
Text = jhbfuygedfb
PostID = 3
Comments
SQL query => DO $$ BEGIN IF EXISTS (select "postID" from "Post" where "postID" = '3') and not exists (select "commentID" from "Comments" where "commentID" = '4') THEN INSERT INTO
["ЗАМЕЧАНИЕ: added\n"]
Continue insertion? 1 - Yes; 2 - No =>

```

Результат	План выполнения		Сообщения		Notifications	
	 commentID [PK] integer	 postID integer	 date date	 text character varying		
1	1	1	2019-09-04	amasing!		
2	2	2	2020-10-15	awful		
3	3	3	2021-04-26	your cats are very beautiful		
4	4	3	2021-11-15	jhbfuygedfb		

Лістинг операції insert:

```

@staticmethod
def insert_for_table1(usrname, usbirth_date, usid):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "This User ID already exists"
        insert = 'DO $$ BEGIN if not exists (select "userID" from "User" where
"userID" = {}) then INSERT ' \
                'INTO "User"("userID", "name", "birth date") VALUES
({}, {}, {}); ' \
                'raise notice {}; else raise notice {}; ' \
                'end if; end $$;'.format(usid, usid, usrname, usbirth_date,
"added", notice)
        restart = False
        print('SQL query => ', insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.closeConnect(connection)

@staticmethod
def insert_for_table2(poid, poname, potopic, pouser):

```

```

connection = controller.makeConnect()
cursor = connection.cursor()
restart = True
while restart:
    notice = "'This Post ID already exists or this User ID does not exist'"
    insert = 'DO $$ BEGIN IF EXISTS (select "userID" from "User" where
"userID" = {}) and not exists ' \
        '(select "postID" from "Post" where "postID" = {}) THEN ' \
        'INSERT INTO "Post"("postID", "namepost", "topic", "userID")
values ({} , {} , {} , {}); RAISE NOTICE {}; ' \
        ' ELSE RAISE NOTICE {}; END IF; ' \
        'END $$;'.format(pouser, poid, poid, poname, potopic, pouser,
"'added'", notice)
    restart = False
    print('SQL query => ', insert)
    cursor.execute(insert)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.closeConnect(connection)

@staticmethod
def insert_for_table3(coid, codate, cotext, copost):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'This Comment ID already exists or this Post ID does not
exist'"
        insert = 'DO $$ BEGIN IF EXISTS (select "postID" from "Post" where
"postID" = {}) and not exists ' \
            '(select "commentID" from "Comments" where "commentID" = {})
THEN ' \
            'INSERT INTO "Comments"("commentID", "date", "text", "postID")
values ({} , {} , {} , {}); RAISE NOTICE {}; ' \
            ' ELSE RAISE NOTICE {}; END IF; END $$;'.format(copost, coid,
coid, codate, cotext, copost,
"'added'",
notice)
        restart = False
        print('SQL query => ', insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.closeConnect(connection)

```

Редагування даних

Таблиця до:

Результат	План выполнения	Сообщения	Notif
	userID [PK] integer	name character varying	birth date date
1	1	Maria	2002-04-14
2	2	Gleb	2002-07-26
3	3	Sofia	2001-10-03

```

Make your choice => 1
1 => User
2 => Post
3 => Comments

Choose table number => 1
Attribute to update(where) User ID = 1
1 => name
2 => birth date

Number of attribute => 1
New value of attribute = Yarina
User
SQL query => DO $$ BEGIN IF EXISTS (select "userID" from "User" where "userID" = '3') THEN update "User" set "name" = 'Yarina' where "userID" = '3'; RAISE NOTICE 'Yarina'; ELSE
Data updated successfully!
Continue updation? 1 - Yes; 2 - No =>

```

Таблиця після:

Результат	План выполнения	Сообщения	Notif
	userID [PK] integer	name character varying	birth date date
1	1	Maria	2002-04-14
2	2	Gleb	2002-07-26
3	3	Yarina	2001-10-03

Якщо заданого користувачем айді немає, виводиться помилка.

```

Make your choice => 1
1 => User
2 => Post
3 => Comments

Choose table number => 1
Attribute to update(where) Post ID = 6
1 => namepost
2 => topic






Number of attribute => 1
New value of attribute = jhbrfjrb
Post
SQL query => DO $$ BEGIN IF EXISTS (select "postID" from "Post" where "postID" = '6') THEN update "Post" set "namepost"= 'jhbrfjrb' where "postID" = '6'; RAISE NOTICE 'jhbrfjrb';
['ЗАМЕЧАНИЕ: There is nothing to update\n']
Continue updation? 1 - Yes; 2 - No =>

```

Вилучення даних

В мене є таблиця **Comments** та одна її дочірня таблиця **User_Comments**

Таблиці до:

Результат						План выполнения	Сообщения	Notifications
	 commentID [PK] integer	 postID integer	 date date	 text character varying				
1	1	1	2019-09-04	amasing!				
2	2	2	2020-10-15	awful				
3	3	3	2021-04-26	your cats are very beautiful				
4	4	3	2021-11-15	jhbfuygedfb				

Результат	План выполнения	Сообщения	Notifications
	User_userID integer	Comments_commentID integer	ID [PK] integer
1	1	1	1
2	2	2	2
3	3	3	3

```

Make your choice => 3

1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 3
Attribute to delete Comment ID = 3
Comments
SQL query => delete from "User_Comments" where "Comments_commentID" = '2';delete from "Comments" where "commentID" = '2';
Data deleted successfully!
Continue deletion? 1 - Yes; 2 - No =>

```

Таблиця після:

Результат

План выполнения

Сообщения

Notifications

	<div>commentID</div> <div>[PK] integer</div>	<div>postID</div> <div>integer</div>	<div>date</div> <div>date</div>	<div>text</div> <div>character varying</div>
1	1	1	2019-09-04	amasing!
2	3	3	2021-04-26	your cats are very beautiful
3	4	3	2021-11-15	jhbfoygedfb

Результат	План выполнения	Сообщения	Notifications
	User_userID integer	Comments_commentID integer	ID [PK] integer
1	1	1	1
2	3	3	3

Так само програма працює для всіх зв'язків між батьківськими та дочірніми таблицями.

Лістинг операції delete:

```

@staticmethod
def delete_for_table1(usid):
    connection = controller.makeConnect()
    cursor = connection.cursor()

```

```

        restart = True
        while restart:
            delete = 'delete from "User_Post" where "User_userID" = {}; ' \
                    'delete from "User_Comments" where "User_userID" = {}; ' \
                    'delete from "Comments" where "postID" in (select "postID"
from "Post" where "userID" = {}); ' \
                    'delete from "Post" where "userID" = {}; ' \
                    'delete from "User" where "userID" = {};'.format(usid,
usid, usid, usid, usid)
            restart = False
            print("SQL query => ", delete)
            cursor.execute(delete)
            connection.commit()
            cursor.close()
            controller.closeConnect(connection)

    @staticmethod
    def delete_for_table2(poid):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:
            delete = 'delete from "User_Comments" where "Comments_commentID" in
(select "commentID" from "Comments" where "postID" = {}); ' \
                    'delete from "User_Post" where "Post_postID" = {}; ' \
                    'delete from "Comments" where "postID" = {}; ' \
                    'delete from "Post" where "postID" = {};'.format(poid, po
poid, po
poid, po
poid)
            restart = False
            print("SQL query => ", delete)
            cursor.execute(delete)
            connection.commit()
            cursor.close()
            controller.closeConnect(connection)

    @staticmethod
    def delete_for_table3(coid):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:
            delete = 'delete from "User_Comments" where "Comments_commentID" = {}; ' \
                    'delete from "Comments" where "commentID" = {};'.format(coid,
coid)
            restart = False
            print("SQL query => ", delete)
            cursor.execute(delete)
            connection.commit()
            cursor.close()
            controller.closeConnect(connection)

```

Завдання 2

Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.


```

Make your choice => 1
How much datas do you want to add => 4

1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 1
User
SQL query => INSERT INTO "User"("name", "birth date") select chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int),
               timestamp '2014-01-10 20:00:00' + random() * (timestamp '2014-01-20 20:00:00' -
               timestamp '2014-01-10 10:00:00') from generate_series(1,4)
Inserted randomly
Continue randomization? 1 - Yes; 2 - No =>

```

Результат	План выполнения	Сообщения	Noti
userID [PK] integer	name character varying	birth date date	
1	1 Maria	2002-04-14	
2	2 Gleb	2002-07-26	
3	3 Yarina	2001-10-03	
4	15 OZ	2014-01-12	
5	16 KA	2014-01-11	
6	17 PG	2014-01-18	
7	18 CE	2014-01-16	

SQL query => INSERT INTO "User"("name", "birth date") select chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int),

timestamp '2014-01-10 20:00:00' + random() * (timestamp '2014-01-20 20:00:00' -

timestamp '2014-01-10 10:00:00') from generate_series(1,4)

Завдання 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.

```

Make your choice => 5
1 => Show name and topic of post which created by *user name*
2 => Show date and text of comment which is under the *post topic*
3 => Show date and text of comment which created by *user name*
Your choice is 1
Enter required user name = Maria
SQL query =>  select "name", "namepost", "topic" from (select c."name", p."namepost", p."topic"
              from "Post" p left join "User" c on c."userID" = p."userID"
              where c."name" LIKE 'Maria' group by c."name", p."namepost", p."topic") as foo

User = Maria
Name of the post = family
Topic = about the relationships with family

-----
Time of request = 15 ms
Data selected successfully!
Continue selection? 1 - Yes; 2 - No =>|

```

```

1 => Show name and topic of post which created by *user name*
2 => Show date and text of comment which is under the *post topic*
3 => Show date and text of comment which created by *user name*
Your choice is 2
Enter required post topic = about the relationships with family
SQL query =>  select "topic", "date", "text" from (select c."topic", p."date", p."text"
              from "Comments" p left join "Post" c on c."postID" = p."postID"
              where c."topic" LIKE 'about the relationships with family' group by c."topic", p."date", p."text") as foo

Topic = about the relationships with family
Date = 2019-09-04
Text = amasing!

-----
Time of request = 0 ms
Data selected successfully!
Continue selection? 1 - Yes; 2 - No =>|

```

```

1 => Show name and topic of post which created by *user name*
2 => Show date and text of comment which is under the *post topic*
3 => Show date and text of comment which created by *user name*
Your choice is 3
Enter required user name = Maria
SQL query =>  select "name", "date", "text" from (select c."name", p."date", p."text"
              from "Comments" p left join "User" c on c."userID" = p."postID"
              where c."name" LIKE 'Maria' group by c."name", p."date", p."text") as foo

User = Maria
Date = 2019-09-04
Text = amasing!

-----
Time of request = 5 ms
Data selected successfully!
Continue selection? 1 - Yes; 2 - No =>|

```

Всі SQL-запити прописані в консолі на ілюстраціях.

Вихід із програми

```
Continue selection? 1 - Yes; 2 - No =>2

Continue work with DB? 1 - Yes; 2 - No. = >2
PostgreSQL connection is closed

Process finished with exit code 0
```

Програмний модуль model.py

Цей програмний модуль відповідає за всю ЛОГІКУ проекту.

```
import controller
import time

tables = {
    1: 'User',
    2: 'Post',
    3: 'Comments',
    4: 'User_Comments',
    5: 'User_Post',
}

class Model:
    @staticmethod
    def validTable(table):
        incorrect = True
        while incorrect:
            if str(table).isdigit():
                table = int(table)
                if table >= 1 and table <= 5:
                    incorrect = False
            else:
                print('Incorrect input, try again.')
        else:
            print('Incorrect input, try again.')
        return table

    @staticmethod
    def showOneTable(table):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        table_name = '""' + tables[table] + '""'
        print(tables[table])
        show = 'select * from public.{}'.format(table_name)
        print("SQL query => ", show)
        print('')
        cursor.execute(show)
        records = cursor.fetchall()
        cursor.close()
        controller.closeConnect(connection)
        return records
```

```

@staticmethod
def insert_for_table1(usname, usbirth_date, usid):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'This User ID already exists'"
        insert = 'DO $$ BEGIN if not exists (select "userID" from "User"
where "userID" = {}) then INSERT ' \
                'INTO "User"("userID", "name", "birth date") VALUES
({},{},{})'; ' \
                'raise notice {}; else raise notice {}; ' \
                'end if; end $$;'.format(usid, usid, usname,
usbirth_date, "'added'", notice)
        restart = False
    print('SQL query => ', insert)
    cursor.execute(insert)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.closeConnect(connection)

@staticmethod
def insert_for_table2(poid, poname, potopic, pouser):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'This Post ID already exists or this User ID does not
exist'"
        insert = 'DO $$ BEGIN IF EXISTS (select "userID" from "User" where
"userID" = {}) and not exists ' \
                '(select "postID" from "Post" where "postID" = {}) THEN ' \
                'INSERT INTO "Post"("postID", "namepost", "topic",
"userID") values ({}, {}, {}, {}); RAISE NOTICE {};' \
                ' ELSE RAISE NOTICE {}; END IF; ' \
                'END $$;'.format(pouser, poid, poid, poname, potopic,
pouser, "'added'", notice)
        restart = False
    print('SQL query => ', insert)
    cursor.execute(insert)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.closeConnect(connection)

@staticmethod
def insert_for_table3(coid, codate, cotext, copost):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'This Comment ID already exists or this Post ID does not
exist'"
        insert = 'DO $$ BEGIN IF EXISTS (select "postID" from "Post" where
"postID" = {}) and not exists ' \
                '(select "commentID" from "Comments" where "commentID" =
{}) THEN ' \
                'INSERT INTO "Comments"("commentID", "date", "text",

```

```

"postID") values ({} , {} , {} , {}); RAISE NOTICE {}; ' \
        ' ELSE RAISE NOTICE {}; END IF; END $$;'.format(copost,
coid, coid, codate, cotext, copost,
                                                                    "'added'",
notice)

        restart = False
        print('SQL query => ', insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.closeConnect(connection)

    @staticmethod
    def delete_for_table1(usid):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:
            delete = 'delete from "User_Post" where "User_userID" = {}; ' \
                    'delete from "User_Comments" where "User_userID" = {}; ' \
                    'delete from "Comments" where "postID" in (select
"postID" from "Post" where "userID" = {}); ' \
                    'delete from "Post" where "userID" = {}; ' \
                    'delete from "User" where "userID" = {}; '.format(usid,
usid, usid, usid, usid)
            restart = False
            print("SQL query => ", delete)
            cursor.execute(delete)
            connection.commit()
            cursor.close()
            controller.closeConnect(connection)

    @staticmethod
    def delete_for_table2(poid):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:
            delete = 'delete from "User_Comments" where "Comments commentID" in
(select "commentID" from "Comments" where "postID" = {}); ' \
                    'delete from "User_Post" where "Post_postID" = {}; ' \
                    'delete from "Comments" where "postID" = {}; ' \
                    'delete from "Post" where "postID" = {}; '.format(poid,
poid, poid, poid)
            restart = False
            print("SQL query => ", delete)
            cursor.execute(delete)
            connection.commit()
            cursor.close()
            controller.closeConnect(connection)

    @staticmethod
    def delete_for_table3(coid):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:

```

```

        delete = 'delete from "User_Comments" where "Comments_commentID" =
{}; ' \
        'delete from "Comments" where "commentID" =
{};'.format(coid, coid)
        restart = False
        print("SQL query => ", delete)
        cursor.execute(delete)
        connection.commit()
        cursor.close()
        controller.closeConnect(connection)

    @staticmethod
    def update_for_table1(usid, set):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:
            notice = "'There is nothing to update'"
            update = 'DO $$ BEGIN IF EXISTS (select "userID" from "User" where
"userID" = {}) THEN ' \
                    'update "User" set {} where "userID" = {}; ' \
                    'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; ' \
                    'END $$;'.format(usid, set, usid, "'updated'", notice)
            restart = False
            pass
            print("SQL query => ", update)
            cursor.execute(update)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.closeConnect(connection)
            pass

    @staticmethod
    def update_for_table2(poid, set):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        restart = True
        while restart:
            notice = "'There is nothing to update'"
            update = 'DO $$ BEGIN IF EXISTS (select "postID" from "Post" where
"postID" = {}) THEN ' \
                    'update "Post" set {} where "postID" = {}; ' \
                    'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; ' \
                    'END $$;'.format(poid, set, poiid, "'updated'", notice)
            restart = False
            pass
            print("SQL query => ", update)
            cursor.execute(update)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.closeConnect(connection)
            pass

    @staticmethod
    def update_for_table3(coid, set):
        connection = controller.makeConnect()
        cursor = connection.cursor()

```

```

        restart = True
        while restart:
            notice = "'There is nothing to update'"
            update = 'DO $$ BEGIN IF EXISTS (select "commentID" from "Comments"
where "commentID" = {}) THEN ' \
                'update "Comments" set {} where "commentID" = {}; ' \
                'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; ' \
                'END $$;'.format(coid, set, coid, "'updated'", notice)
            restart = False
        pass
    print("SQL query => ", update)
    cursor.execute(update)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.closeConnect(connection)
    pass

    @staticmethod
    def select1(user):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        select = """select "name", "namepost", "topic" from (select c."name",
p."namepost", p."topic"
                        from "Post" p left join "User" c on c."userID" =
p."userID"
                        where c."name" LIKE '{}' group by c."name",
p."namepost", p."topic") as foo""".format(user)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        records = cursor.fetchall()
        print('Time of request = {} ms'.format(end))
        cursor.close()
        controller.closeConnect(connection)
        return records

    @staticmethod
    def select2(post):
        connection = controller.makeConnect()
        cursor = connection.cursor()
        select = """select "topic", "date", "text" from (select c."topic",
p."date", p."text"
                        from "Comments" p left join "Post" c on c."postID" =
p."postID"
                        where c."topic" LIKE '{}' group by c."topic",
p."date", p."text") as foo""".format(post)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        records = cursor.fetchall()
        print('Time of request = {} ms'.format(end))
        cursor.close()
        controller.closeConnect(connection)
        return records

    @staticmethod

```

```

def select3(user):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    select = """select "name", "date", "text" from (select c."name",
p."date", p."text"
                                from "Comments" p left join "User" c on c."userID" =
p."postID"
                                where c."name" LIKE '{}' group by c."name",
p."date", p."text") as foo""".format(user)
    print("SQL query => ", select)
    beg = int(time.time() * 1000)
    cursor.execute(select)
    end = int(time.time() * 1000) - beg
    records = cursor.fetchall()
    print('Time of request = {} ms'.format(end))
    cursor.close()
    controller.closeConnect(connection)
    return records

@staticmethod
def random(table, num):
    connection = controller.makeConnect()
    cursor = connection.cursor()
    incorrect = True
    while incorrect:
        if table == 1:
            insert = """INSERT INTO "User"("name", "birth date") select
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int),
                                timestamp '2014-01-10 20:00:00' + random() * (timestamp
'2014-01-20 20:00:00' -
                                timestamp '2014-01-10 10:00:00') from
generate_series(1,{})""".format(num)
            incorrect = False
        elif table == 2:
            insert = """INSERT INTO "Post" ("namepost", "topic") select
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int),
                                chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)
                                from generate_series(1,{})""".format(num)
            incorrect = False
        elif table == 3:
            insert = """INSERT INTO "Comments"("text", "date") select
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int),
                                timestamp '2014-01-10 20:00:00' + random() * (timestamp
'2014-01-20 20:00:00' -
                                timestamp '2014-01-10 10:00:00') from generate_series(1,
{})""".format(num)
            incorrect = False
        else:
            print('Incorrect input, try again.')
    print("SQL query => ", insert)
    cursor.execute(insert)
    connection.commit()
    cursor.close()
    controller.closeConnect(connection)

```

Маємо словник tables для виведення та обирання таблиць за певним числом. Також клас Model, який включає всі потрібні функції:

validTable – функція, яка переводить рядок у число, перевіряє наявність такої таблиці та повертає її;

showAllTables – функція, яка виводить всі таблиці;

showOneTable – функція, яка виводить тільки задану таблицю;

insert_for table1 – функція, яка виконує операцію внесення даних для 1 таблиці;

insert_for table2 – функція, яка виконує операцію внесення даних для 2 таблиці;

insert_for table3 – функція, яка виконує операцію внесення даних для 3 таблиці;

delete_for_table1 – функція, яка виконує операцію вилучення даних для 1 таблиці;

delete_for_table2 – функція, яка виконує операцію вилучення даних для 2 таблиці;

delete_for_table3 – функція, яка виконує операцію вилучення даних для 3 таблиці;

update_for_table1 – функція, яка виконує операцію редагування даних для 1 таблиці;

update_for_table2 – функція, яка виконує операцію редагування даних для 2 таблиці;

update_for_table3 – функція, яка виконує операцію редагування даних для 3 таблиці;

select1 – функція, яка виконує перший пошуковий запит;

select2 – функція, яка виконує другий пошуковий запит;

select3 – функція, яка виконує третій пошуковий запит;

random – функція, яка рандомно генерує дані таблиці.