

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

**Лабораторна робота № 1**

з дисципліни

**«Бази даних та засоби управління»**

Виконав: студент групи КВ-11

Угнівенко Ярослав

Контакт в telegram: @yar4ik4ik

Посилання на гіт: [yaro-krasav4ik228/DataBases: dlya laboratornyh robit (github.com)](https://github.com/yaro-krasav4ik228/DataBases/)

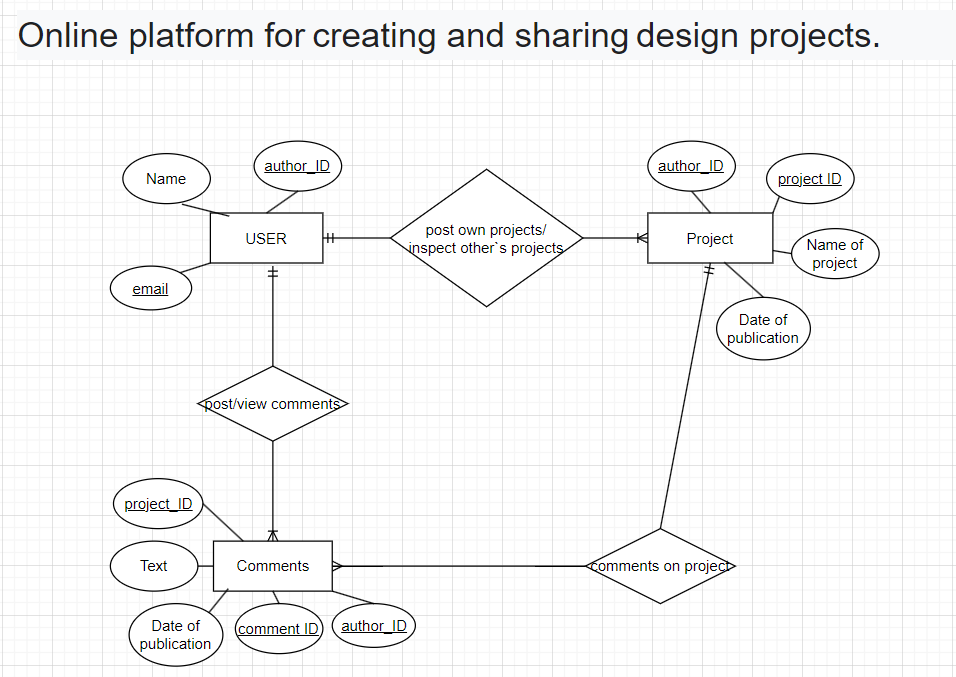
Київ

2023

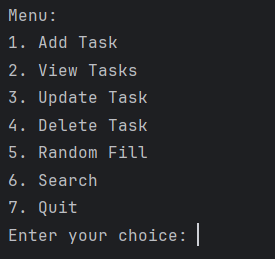
**Про базу даних**

Для даної роботи було вибрано тему “ Онлайн-платформа для зберігання та обміну дизайнерськими проектами”. Тут наявні 3 сутності: User, Project і Comments. Перша сутність – User, має 3 атрибути: нік і ключові – ID, електронна пошта. Пошта є ключовими, для того, щоб на сайт можна було заходити за її допомоги, а також, щоб не було повторень і не можна було зареєструвати більше 1 аккаунта на одну пошту. ID – унікальний ключ, який присвоюється Usery у внутрішній базі і потрібен для його ідентифікації. Ім’я не є унікальним. Друга сутність – Project, має 4 атрибути: ім’я проекту, дата публікації проекту, ID проекту, який є ключовим і по ньому можна здійснити пошук саме цього проекту, а також зовнішній ключ – author\_ID. Остання сутність – Comments, в якій є атрибути тексту(сам коментар), дата публікації і 3 ключових поля, 2 з яких – зовнішні, а саме є comment\_id, який є унікальним полем для самої сутності, author\_id, який посилається на автора коментаря і project\_id, який посилається на проект, до якого доданий коментар.

Нотація “Чена”:



**Про програму**

Так виглядає меню програми, всього 7 можливих “кнопок”. Щоб вибрати пункт з меню потрібно в графі Enter your choice вписати обрану цифру.

1. Add Task – з’являється покрокове меню з заповненням одного рядка якоїсь із трьох таблиць на вибір.

2. View Tasks – меню при виборі якого можна подивитись повністю вміст якоїсь таблиці.

3. Update Task – функція зміни даних в таблиці, працює по вибору ключа.

4. Delete Task – видаляє(якщо це можливо) рядок таблиці за вибором. Видалення не може відбутись, якщо один з параметрів є зовнішнім ключом і використовується в іншій таблиці.

5. Random Fill – заповнює обрану таблицю випадковими значеннями по шаблону, в кількості, заданій користувачем.

6. Search – шукає за параметрами дані в таблицях.

7. Quit – вихід.

**Назва мови програмування та бібліотек**

IDE - PyCharm 2023.2.3 (Professional Edition)

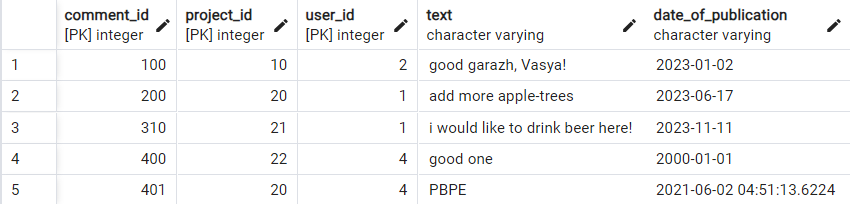
Мова – Python 3.9

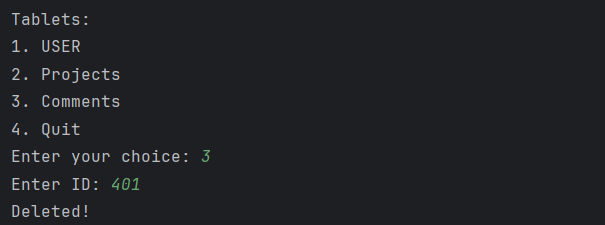
Бібліотеки:

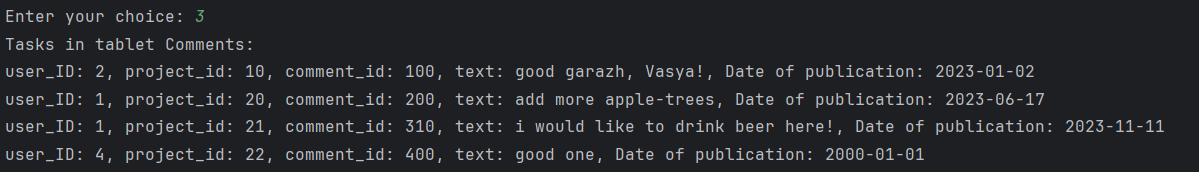
import time – бібліотека для роботи з часом  
import psycopg2 – бібліотека для роботи з базами даних  
from psycopg2 import DataError – для виявлення типів помилок

**ПУНКТ 1**

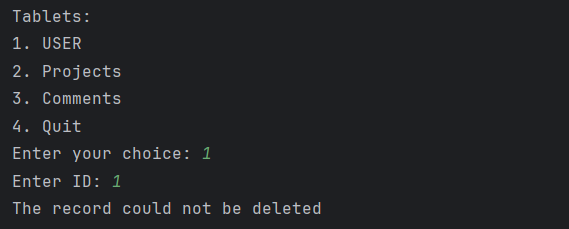
* лістинги та скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення, а якщо воно неможливе, то результат перехоплення помилки з виведенням повідомлення про неможливість такого видалення за наявності залежних даних. Причини помилок мають бути пояснені;
* def delete\_task\_tab1(self, task\_id):  
   c = self.conn.cursor()  
   c.execute("""  
   WITH referenced\_rows AS (  
   SELECT "user\_id" FROM public."Comments" WHERE "user\_id" = %s  
   UNION  
   SELECT "user\_ID" FROM public."Projects" WHERE "user\_ID" = %s  
   )  
   DELETE FROM public."USER" WHERE "user\_ID" = %s  
   AND NOT EXISTS ( SELECT 1 FROM referenced\_rows);  
   """, (task\_id, task\_id, task\_id))  
   self.conn.commit()  
    
  def delete\_task\_tab2(self, task\_id):  
   c = self.conn.cursor()  
   c.execute("""  
   WITH referenced\_rows AS (  
   SELECT DISTINCT "project\_id"   
   FROM public."Comments" WHERE "project\_id" = %s)  
   DELETE FROM public."Projects" WHERE "project\_ID" = %s  
   AND NOT EXISTS (SELECT 1 FROM referenced\_rows);""", (task\_id, task\_id))  
   self.conn.commit()  
    
  def delete\_task\_tab3(self, task\_id):  
   c = self.conn.cursor()  
   c.execute('DELETE FROM public."Comments" WHERE "comment\_id"=%s', (task\_id, ))  
   self.conn.commit()

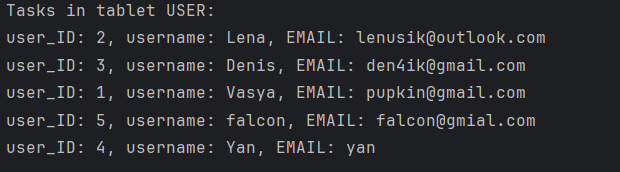






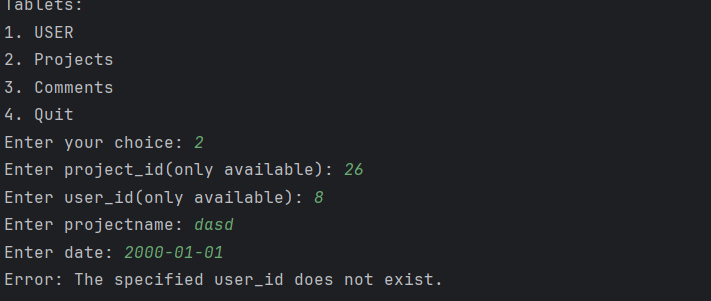
Як бачимо, видалення відбулося, але це сталося тому, що даний user\_id не використовувався у інших таблицях, щоб видалити запис, який присутній і в інших таблицях, за рахунок зовнішнього ключа, треба спочатку видалити запис в дочірніх таблицях, а вже потім виконувати в основній. Спробуємо видалити саме такий рядок:



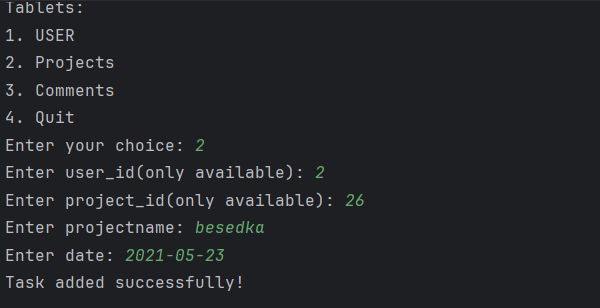


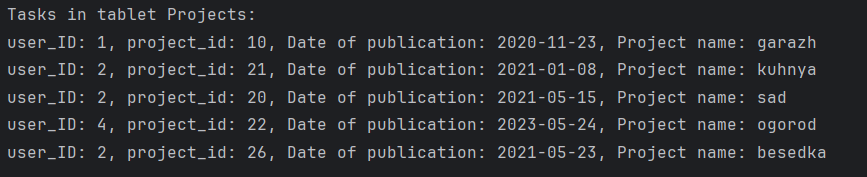
* лістинги та скріншоти результатів виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо в батьківські таблиці немає відповідного запису.
* def add\_task\_tab1(self, userid, title, description):  
   c = self.conn.cursor()  
   try:  
   query = 'INSERT INTO public."USER" ("user\_ID", username, email) VALUES (%s, %s, %s)'  
   data = (userid, title, description)  
   c.execute(query, data)  
   self.conn.commit()  
   print("Task added successfully!")  
   except DataError as e:  
   print(f"Error: {e}")  
    
  def add\_task\_tab2(self, projectid, userid, title, description):  
   c = self.conn.cursor()  
   try:  
   check\_user\_query = 'SELECT 1 FROM public."USER" WHERE "user\_ID" = %s'  
   c.execute(check\_user\_query, (userid,))  
   user\_exists = c.fetchone()  
   if not user\_exists:  
   print("Error: The specified user\_id does not exist.")  
   else:  
   query = 'INSERT INTO public."Projects" ("project\_ID", "user\_ID", projectname, "Date\_of\_publication") VALUES (%s, %s, %s, %s)'  
   data = (projectid, userid, title, description)  
   c.execute(query, data)  
   self.conn.commit()  
   print("Task added successfully!")  
   except psycopg2.DataError as e:  
   print(f"Error: {e}")  
   except Exception as e:  
   print(f"Error: {e}")  
    
  def add\_task\_tab3(self, commentid, projectid, userid, title, description):  
   c = self.conn.cursor()  
   try:  
   check\_user\_query = 'SELECT 1 FROM public."USER" WHERE "user\_ID" = %s'  
   c.execute(check\_user\_query, (userid,))  
   user\_exists = c.fetchone()  
   check\_project\_query = 'SELECT 1 FROM public."Projects" WHERE "project\_ID" = %s'  
   c.execute(check\_project\_query, (projectid,))  
   project\_exists = c.fetchone()  
   if not user\_exists:  
   print("Error: The specified user\_id does not exist.")  
   elif not project\_exists:  
   print("Error: The specified project\_id does not exist.")  
   else:  
   query = 'INSERT INTO public."Comments" ("comment\_id", "user\_id", "project\_id", "text", "date\_of\_publication") VALUES (%s, %s, %s, %s, %s)'  
   data = (commentid, userid, projectid, title, description)  
   c.execute(query, data)  
   self.conn.commit()  
   print("Comment added successfully!")  
   except psycopg2.DataError as e:  
   print(f"Error: {e}")  
   except Exception as e:  
   print(f"Error: {e}")

Вище наявний скріншот таблиці USER, таблиця Projects є дочірньою таблицею, спробуємо додати туди запис з user\_id = 8:



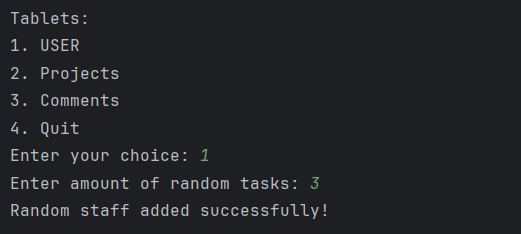
Тепер спробуємо додати запис з наявним значенням:



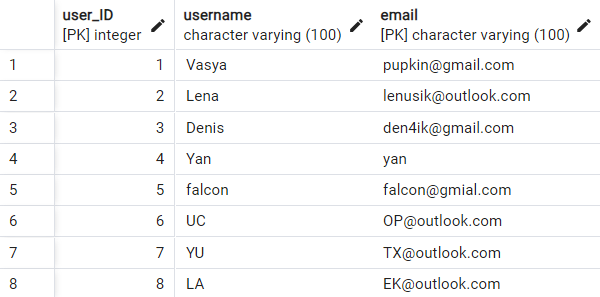


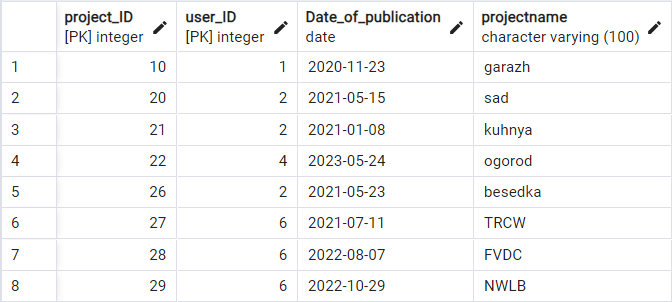
**ПУНКТ 2**

Згенеруємо лише по 3 випадкових рядки в кожній таблиці:

(Аналогічно для двох інших таблиць)

Результат(останні три записи – випадкові):





На останньому скріншоті date\_of\_publication має формат character varying замість date, це було зроблено помилково ще в першій лабораторній роботі, але навіть так можна побачити коректне створення випадкової дати.

Код SQL для першої таблиці:

DO $$  
DECLARE  
max\_user\_id INT;  
BEGIN  
SELECT MAX("user\_ID") INTO max\_user\_id FROM public."USER";  
INSERT INTO public."USER" ("user\_ID", username, email)  
SELECT  
COALESCE(max\_user\_id, 0) + generate\_series,  
chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int),  
chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) || ('@outlook.com')  
FROM generate\_series(1, %s);  
END $$;

замість %s – 3.

Код SQL для другої таблиці:

DO $$  
DECLARE  
max\_project\_id INT;  
random\_user\_id INT;  
BEGIN  
SELECT MAX("project\_ID") INTO max\_project\_id FROM public."Projects";   
SELECT "user\_ID" FROM public."USER" ORDER BY random() LIMIT 1 INTO random\_user\_id;   
INSERT INTO public."Projects" ("project\_ID", "user\_ID", "Date\_of\_publication", "projectname")  
SELECT  
COALESCE(max\_project\_id, 0) + generate\_series,  
random\_user\_id,  
current\_date - (random() \* interval '5 years'),  
chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)  
FROM generate\_series(1, %s);  
END $$;

замість %s – 3.

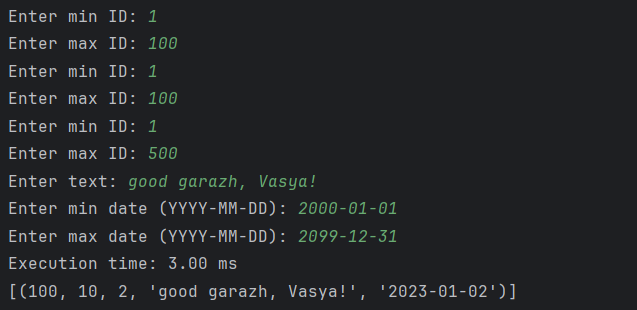
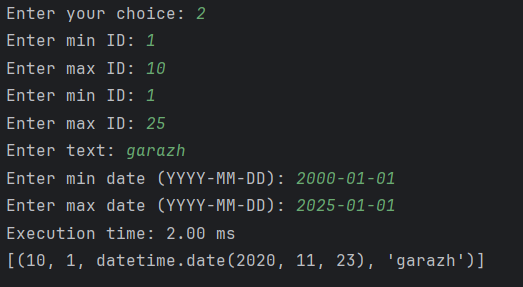
Код SQL для третьої таблиці:

DO $$  
 DECLARE  
 max\_comment\_id INT;  
 random\_user\_id INT;  
random\_project\_id INT;  
 BEGIN  
 SELECT MAX("comment\_id") INTO max\_comment\_id FROM public."Comments";   
 SELECT "user\_ID" FROM public."USER" ORDER BY random() LIMIT 1 INTO random\_user\_id;  
SELECT "project\_ID" FROM public."Projects" ORDER BY random() LIMIT 1 INTO random\_project\_id;  
 INSERT INTO public."Comments" ("comment\_id", "project\_id", "user\_id", "text" , "date\_of\_publication")  
 SELECT  
 COALESCE(max\_comment\_id, 0) + generate\_series,  
random\_project\_id,  
 random\_user\_id,  
chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),  
 current\_date - (random() \* interval '5 years')  
 FROM generate\_series(1, %s);  
 END $$;

замість %s – 3.

**ПУНКТ 3**

Для пошуку потрібно вводити всі параметри, а саме текст, він має бути 1в1 з шуканим.

Пошук в таблиці 1:

SELECT \*  
FROM public."USER"  
WHERE  
 "user\_ID" BETWEEN %s AND %s  
 AND "username" LIKE %s  
 AND "email" LIKE %s;

Пошук в таблиці 2:

SELECT \*  
FROM public."Projects"  
WHERE  
"project\_ID" BETWEEN %s AND %s  
AND "user\_ID" BETWEEN %s AND %s  
AND "projectname" LIKE %s  
AND "Date\_of\_publication" BETWEEN %s AND %s;

Пошук в таблиці 3:

SELECT \*  
FROM public."Comments"  
WHERE  
"comment\_id" BETWEEN %s AND %s  
AND "project\_id" BETWEEN %s AND %s  
AND "user\_id" BETWEEN %s AND %s  
AND "text" LIKE %s  
AND "date\_of\_publication" BETWEEN %s AND %s;

Також можна реалізувати пошук одразу в декількох таблицях та не за усіма параметрами, наприклад пошук user\_id в таблиці USER і projectname в таблиці Projects, тобто знайти по номеру користувача його проект з якимось ім’ям:

SELECT \* FROM  
"USER" u  
JOIN Projects p

ON u.user\_ID = p.user\_ID  
WHERE (u.user\_ID BETWEEN min\_user\_ID AND max\_user\_ID)  
AND (p.projectname LIKE '%s')

(замість %s – ім’я проекту, min\_user\_ID I max\_user\_id – відповідні межі значень в пошуках ID).

**ПУНКТ 4**

Код файла model.py:

import time  
import psycopg2  
from psycopg2 import DataError  
  
class Model:  
 def \_\_init\_\_(self):  
 self.conn = psycopg2.connect(  
 dbname='postgres',  
 user='postgres',  
 password='',  
 host='localhost',  
 port=5432  
 )  
  
 def add\_task\_tab1(self, userid, title, description):  
 c = self.conn.cursor()  
 try:  
 query = 'INSERT INTO public."USER" ("user\_ID", username, email) VALUES (%s, %s, %s)'  
 data = (userid, title, description)  
 c.execute(query, data)  
 self.conn.commit()  
 print("Task added successfully!")  
 except DataError as e:  
 print(f"Error: {e}")  
  
 def add\_task\_tab2(self, projectid, userid, title, description):  
 c = self.conn.cursor()  
 try:  
 check\_user\_query = 'SELECT 1 FROM public."USER" WHERE "user\_ID" = %s'  
 c.execute(check\_user\_query, (userid,))  
 user\_exists = c.fetchone()  
 if not user\_exists:  
 print("Error: The specified user\_id does not exist.")  
 else:  
 query = 'INSERT INTO public."Projects" ("project\_ID", "user\_ID", projectname, "Date\_of\_publication") VALUES (%s, %s, %s, %s)'  
 data = (projectid, userid, title, description)  
 c.execute(query, data)  
 self.conn.commit()  
 print("Task added successfully!")  
 except psycopg2.DataError as e:  
 print(f"Error: {e}")  
 except Exception as e:  
 print(f"Error: {e}")  
  
 def add\_task\_tab3(self, commentid, projectid, userid, title, description):  
 c = self.conn.cursor()  
 try:  
 check\_user\_query = 'SELECT 1 FROM public."USER" WHERE "user\_ID" = %s'  
 c.execute(check\_user\_query, (userid,))  
 user\_exists = c.fetchone()  
 check\_project\_query = 'SELECT 1 FROM public."Projects" WHERE "project\_ID" = %s'  
 c.execute(check\_project\_query, (projectid,))  
 project\_exists = c.fetchone()  
 if not user\_exists:  
 print("Error: The specified user\_id does not exist.")  
 elif not project\_exists:  
 print("Error: The specified project\_id does not exist.")  
 else:  
 query = 'INSERT INTO public."Comments" ("comment\_id", "user\_id", "project\_id", "text", "date\_of\_publication") VALUES (%s, %s, %s, %s, %s)'  
 data = (commentid, userid, projectid, title, description)  
 c.execute(query, data)  
 self.conn.commit()  
 print("Comment added successfully!")  
 except psycopg2.DataError as e:  
 print(f"Error: {e}")  
 except Exception as e:  
 print(f"Error: {e}")  
  
  
 def get\_all\_tasks\_tab1(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM public."USER"')  
 return c.fetchall()  
 def get\_all\_tasks\_tab2(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM public."Projects"')  
 return c.fetchall()  
 def get\_all\_tasks\_tab3(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM public."Comments"')  
 return c.fetchall()  
  
 def update\_task\_tab1(self, task\_id, title, description):  
 c = self.conn.cursor()  
 c.execute('UPDATE public."USER" SET username=%s, email=%s WHERE "user\_ID"=%s', (title, description, task\_id))  
 self.conn.commit()  
  
  
 def update\_task\_tab2(self, task\_id, title, description):  
 c = self.conn.cursor()  
 c.execute('UPDATE public."Projects" SET projectname=%s, "Date\_of\_publication"=%s WHERE "project\_ID"=%s', (title, description, task\_id))  
 self.conn.commit()  
  
 def update\_task\_tab3(self, task\_id, title, description):  
 c = self.conn.cursor()  
 c.execute('UPDATE public."Comments" SET text=%s, Date\_of\_publication=%s WHERE comment\_ID=%s', (title, description, task\_id))  
 self.conn.commit()  
  
 def delete\_task\_tab1(self, task\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute("""  
 WITH referenced\_rows AS (  
 SELECT "user\_id" FROM public."Comments" WHERE "user\_id" = %s  
 UNION  
 SELECT "user\_ID" FROM public."Projects" WHERE "user\_ID" = %s  
 )  
 DELETE FROM public."USER" WHERE "user\_ID" = %s  
 AND NOT EXISTS (SELECT 1 FROM referenced\_rows);  
 """, (task\_id, task\_id, task\_id))  
 self.conn.commit()  
 print("Deleted!" if c.rowcount > 0 else "The record could not be deleted")  
 except Exception as e:  
 self.conn.rollback()  
  
 def delete\_task\_tab2(self, task\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute("""  
 WITH referenced\_rows AS (  
 SELECT DISTINCT "project\_id"   
 FROM public."Comments" WHERE "project\_id" = %s)  
 DELETE FROM public."Projects" WHERE "project\_ID" = %s  
 AND NOT EXISTS (SELECT 1 FROM referenced\_rows);  
 """, (task\_id, task\_id))  
 self.conn.commit()  
 print("Deleted!" if c.rowcount > 0 else "The record could not be deleted")  
 except Exception as e:  
 self.conn.rollback()  
  
 def delete\_task\_tab3(self, task\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM public."Comments" WHERE "comment\_id"=%s', (task\_id, ))  
 self.conn.commit()  
 print("Deleted!" if c.rowcount > 0 else "The record could not be deleted")  
 except Exception as e:  
 self.conn.rollback()  
  
 def random\_add\_task\_tab1(self, amount):  
 c = self.conn.cursor()  
 c.execute("""  
 DO $$  
 DECLARE  
 max\_user\_id INT;  
 BEGIN  
 SELECT MAX("user\_ID") INTO max\_user\_id FROM public."USER";  
 INSERT INTO public."USER" ("user\_ID", username, email)  
 SELECT  
 COALESCE(max\_user\_id, 0) + generate\_series,  
 chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int),  
 chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) || ('@outlook.com')  
 FROM generate\_series(1, %s);  
 END $$;  
 """, (amount, ))  
 self.conn.commit()  
  
 def random\_add\_task\_tab2(self, amount):  
 c = self.conn.cursor()  
 c.execute("""  
 DO $$  
 DECLARE  
 max\_project\_id INT;  
 random\_user\_id INT;  
 BEGIN  
 SELECT MAX("project\_ID") INTO max\_project\_id FROM public."Projects";   
 SELECT "user\_ID" FROM public."USER" ORDER BY random() LIMIT 1 INTO random\_user\_id;   
 INSERT INTO public."Projects" ("project\_ID", "user\_ID", "Date\_of\_publication", "projectname")  
 SELECT  
 COALESCE(max\_project\_id, 0) + generate\_series,  
 random\_user\_id,  
 current\_date - (random() \* interval '5 years'),  
 chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int)  
 FROM generate\_series(1, %s);  
 END $$;""", (amount, ))  
 self.conn.commit()  
  
 def random\_add\_task\_tab3(self, amount):  
 c = self.conn.cursor()  
 c.execute("""  
 DO $$  
 DECLARE  
 max\_comment\_id INT;  
 random\_user\_id INT;  
 random\_project\_id INT;  
 BEGIN  
 SELECT MAX("comment\_id") INTO max\_comment\_id FROM public."Comments";   
 SELECT "user\_ID" FROM public."USER" ORDER BY random() LIMIT 1 INTO random\_user\_id;  
 SELECT "project\_ID" FROM public."Projects" ORDER BY random() LIMIT 1 INTO random\_project\_id;  
 INSERT INTO public."Comments" ("comment\_id", "project\_id", "user\_id", "text" , "date\_of\_publication")  
 SELECT  
 COALESCE(max\_comment\_id, 0) + generate\_series,  
 random\_project\_id,  
 random\_user\_id,  
 chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),  
 current\_date - (random() \* interval '5 years')  
 FROM generate\_series(1, %s);  
 END $$;""", (amount, ))  
 self.conn.commit()  
  
 def search\_tab1(self, min\_user\_ID, max\_user\_ID, username, email):  
 start\_time = time.time()  
 c = self.conn.cursor()  
 c.execute("""  
 SELECT \*  
 FROM public."USER"  
 WHERE  
 "user\_ID" BETWEEN %s AND %s  
 AND "username" LIKE %s  
 AND "email" LIKE %s;  
 """, (min\_user\_ID, max\_user\_ID, username, email))  
 execution\_time = (time.time() - start\_time) \* 1000  
 print("Execution time: %.2f ms" % execution\_time)  
 print(c.fetchall())  
 def search\_tab2(self, min\_project\_ID, max\_project\_ID, min\_user\_ID, max\_user\_ID, projectname, start\_date, end\_date):  
 start\_time = time.time()  
 c = self.conn.cursor()  
 c.execute("""SELECT \*  
 FROM public."Projects"  
 WHERE  
 "project\_ID" BETWEEN %s AND %s  
 AND "user\_ID" BETWEEN %s AND %s  
 AND "projectname" LIKE %s  
 AND "Date\_of\_publication" BETWEEN %s AND %s;""", (min\_project\_ID, max\_project\_ID, min\_user\_ID, max\_user\_ID, projectname, start\_date, end\_date))  
 execution\_time = (time.time() - start\_time) \* 1000  
 print("Execution time: %.2f ms" % execution\_time)  
 print(c.fetchall())  
  
 def search\_tab3(self, min\_comment\_id, max\_comment\_id, min\_project\_ID, max\_project\_ID, min\_user\_ID, max\_user\_ID, text, start\_date, end\_date):  
 start\_time = time.time()  
 c = self.conn.cursor()  
 c.execute("""  
 SELECT \*  
 FROM public."Comments"  
 WHERE  
 "comment\_id" BETWEEN %s AND %s  
 AND "project\_id" BETWEEN %s AND %s  
 AND "user\_id" BETWEEN %s AND %s  
 AND "text" LIKE %s  
 AND "date\_of\_publication" BETWEEN %s AND %s;  
 """, (min\_comment\_id, max\_comment\_id, min\_project\_ID, max\_project\_ID, min\_user\_ID, max\_user\_ID, text, start\_date, end\_date))  
 execution\_time = (time.time() - start\_time) \* 1000  
 print("Execution time: %.2f ms" % execution\_time)  
 print(c.fetchall())

Для кожної таблиці, згідно її колонок і типів значень створені відповідні функції, тому кожної функції по три, і їх можна розрізнити по тексту після останнього подчерка, а саме \_tab1, \_tab2, \_tab3. Перший метод класа у файлі має в собі конфігураційну інформацію, пароль, ім’я користувача і бази даних, адрес сервера.

Функції:

add\_task – функції по додаванню полей в таблиці

get\_all\_tasks – функції виводу інформації з таблиць

update\_task – функції зміни полей в таблицях

delete\_task – функції видалення полей з таблиць

random\_add\_task – функції заповнення випадковими значеннями таблиць

search – функціх пошуку в таблицях

Кожна функція має в собі код створення курсору(c = self.conn.cursor()) і виконання цього курсору(c.execute()). Весь код самих функцій реалізований на мові SQL і всі запити виконуються в курсорі. Також наявні функції виводу, обробки часу роботи і де треба – self.conn.commit() – команда, яка зберігає зміни в базі даних.