

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»**

**Тема:** «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Github: https://github.com/imaihs/bd\_lab/tree/main/lab2

Виконав: студент ІІI курсу ФПМ групи КВ-03

Черненький А.О.

Перевірив:

Київ – 2022

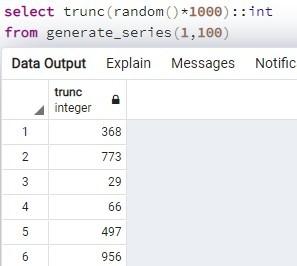
*Загальне завдання* роботи полягає в такому:

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).

*Деталізоване завдання*:

1. Забезпечити можливість уведення/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 підлеглій 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**!

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061) [за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

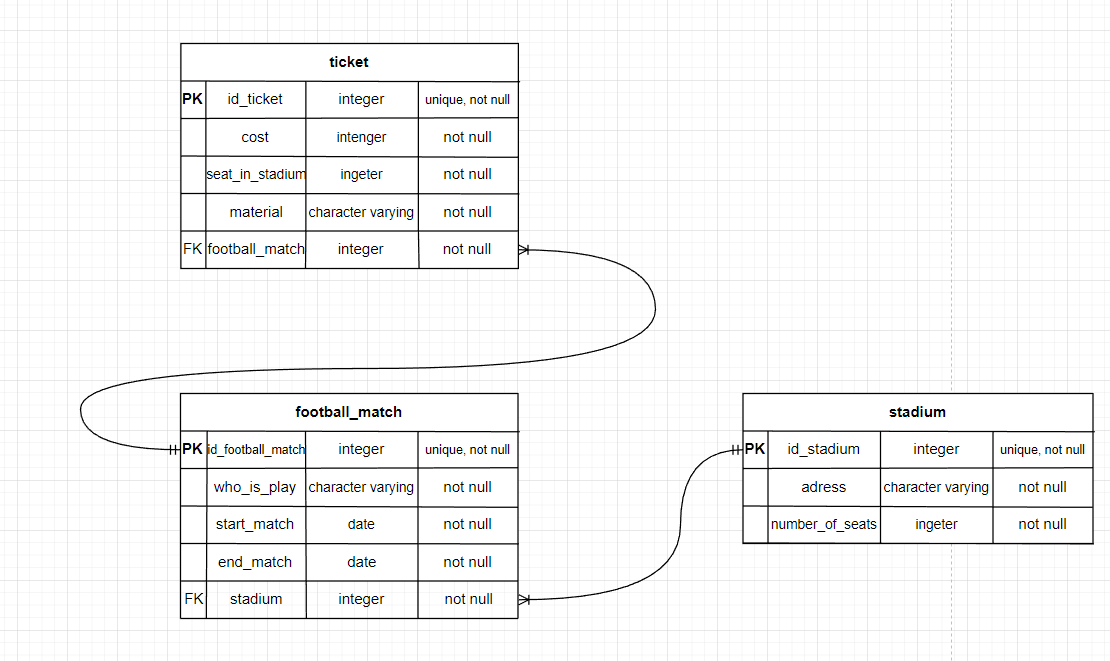
Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності foreign key).

1. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення 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.

1. Програмний код організувати згідно шаблону Model-View- Controller MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: <http://initd.org/psycopg/docs/usage.html)>



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сутність | Атрибут | Опис Атрибуту | Тип | Обмеженя |
| ticket | id | unique identifier | integer | not null  unique |
| cost | ціна в ₴ | integer | not null |
| seat\_in\_the\_stadium | Місце на стадіонні для вболівальник | integer | not null |
| material | Матеріал з якого виготовлено квиток | character varying | not null |
| football\_match | посилання на характеристику | integer | not null |
| football\_match | id | unique identifier | integer | not null  unique |
| who\_is\_play | які команди грають? | character  varying | not null |
| start\_match | час початку матчу | date | not null |
| end\_match | час кінця матчу | date | not null |
| stadium | посилання на характеристику | integer | not null |
| stadium | id | unique identifier | integer | not null  unique |
| adress | адреса стадіону | character varying | not null |
| number\_of\_seats | кількість місць | integer | not null |

Опис функціоналу меню:

Update – оновлення данних в певній колонці,

Add – додання нової колонки,

Delete – видалення нової колонки

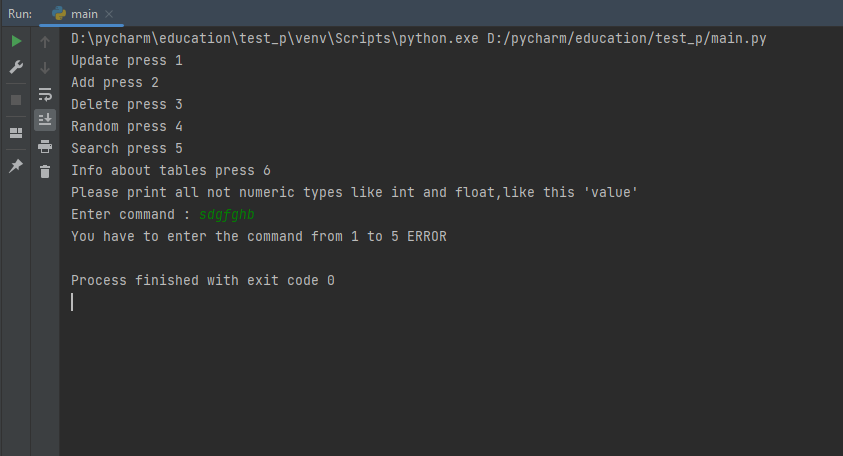
Random - рандомна генерація нових значень для колонок на n раз

Search – пошук по базовим данним якоїсь колонки

Info about tables – повна інформація по бд

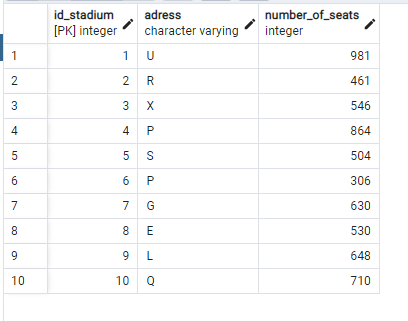
# Відповідь на вимоги до пункту №1 деталізованого завдання:

# Ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних:

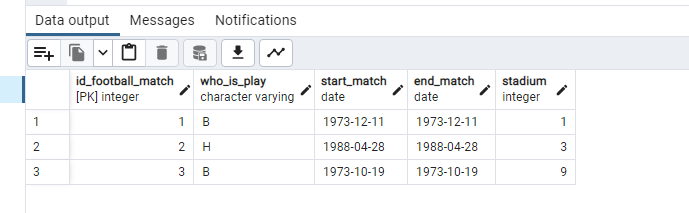


Видалення данних

Даннi стадіона до видалення:

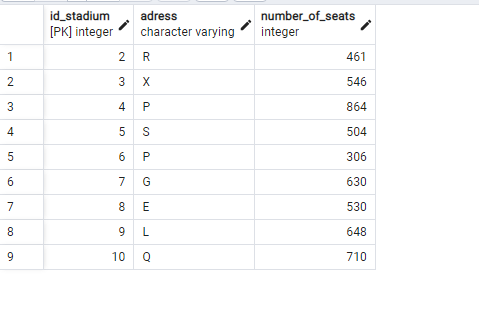


Даннi матчі до видалення:

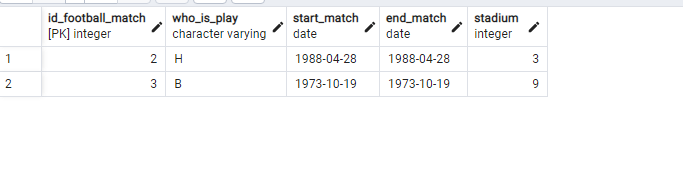




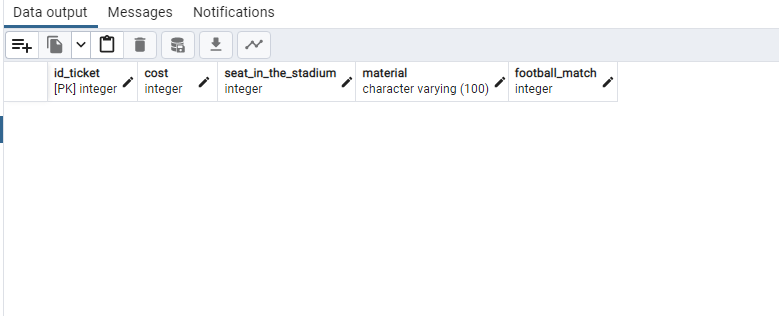
Стадіони після видалення:

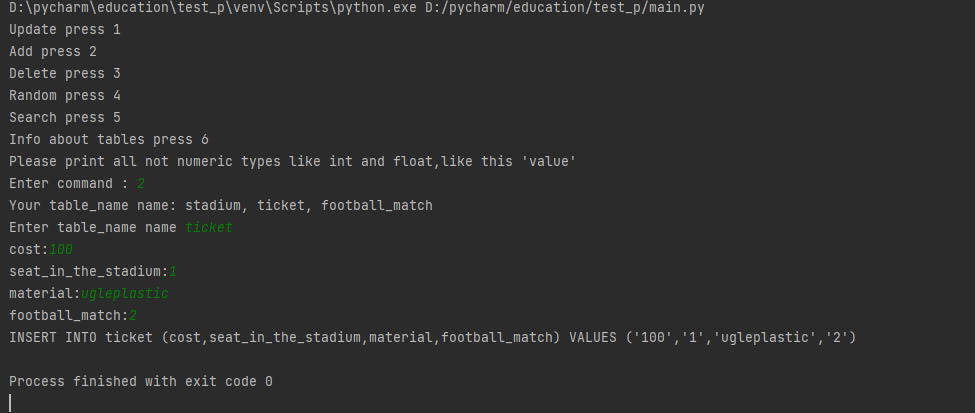


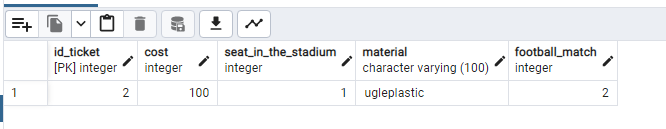
Матчи після видалення:



Додавання данних:

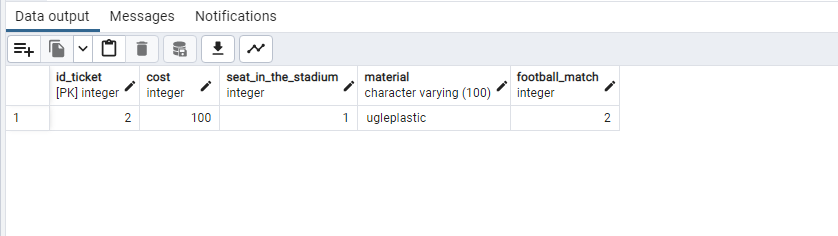


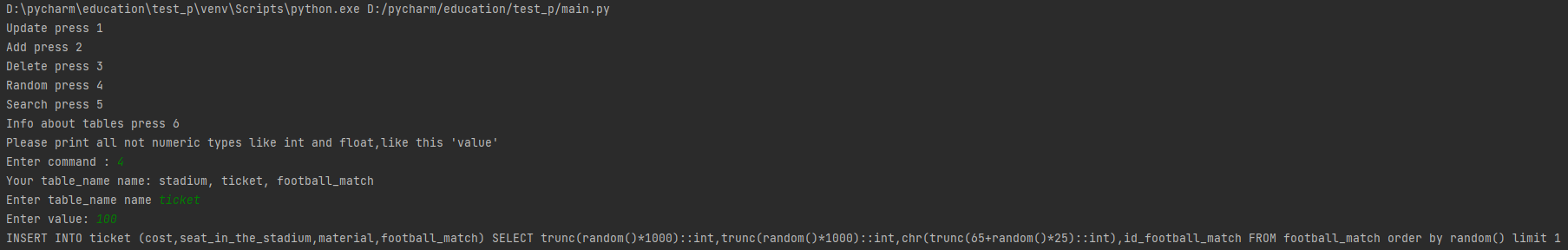
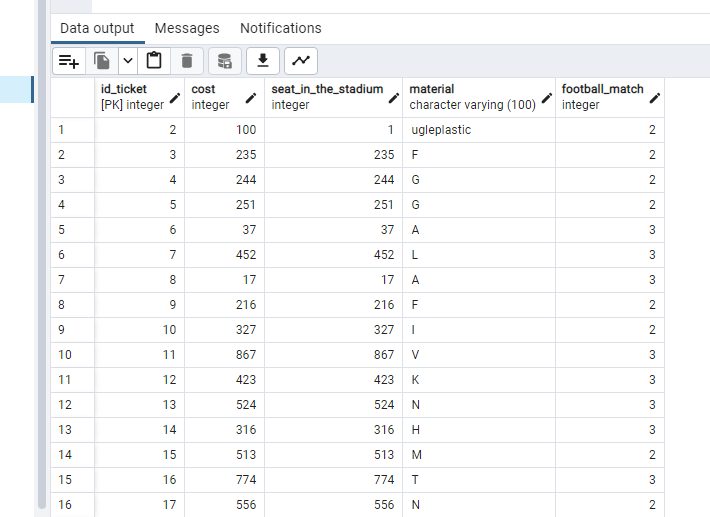




# Вимоги до пункту №2 деталізованого завдання:

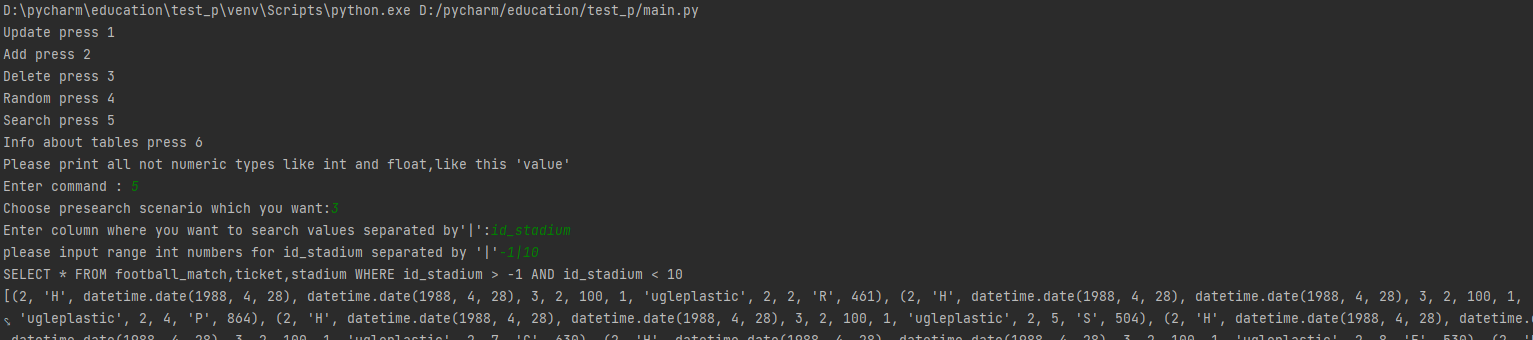
Меню генерації:



# Вимоги до пункту №3 деталізованого завдання:

Ілюстрації уведення пошукового запиту та результатів виконання запитів:



# Вимоги до пункту №4 деталізованого завдання:

Ілюстрації програмного коду з репозиторію Git:

main.py:

from model import \*

from view import \*

import sys

def request():

input\_command = comand\_identification()

if (input\_command == '1'):

table = table\_name()

str\_of\_columns = input\_colums\_str\_upd()

str\_of\_updating\_column= take\_inf\_about\_param(str\_of\_columns)

str\_of\_based\_column = take\_inf\_based()

update(table, str\_of\_updating\_column, str\_of\_based\_column)

elif (input\_command == '2'):

table = table\_name()

inf = take\_inf\_for\_adding(table)

add\_information(table, inf)

elif (input\_command == '3'):

table = table\_name()

str\_of\_based\_column = take\_inf\_based()

delete(table,str\_of\_based\_column)

elif (input\_command == '4'):

table = table\_name()

data = Data()

random(table, data)

elif (input\_command == '5'):

scenario = choose\_scenario\_search()

str\_of\_columns = input\_colums\_str\_search()

dict\_of\_searching\_var = take\_searching\_rows(str\_of\_columns)

list\_of\_searching = search(scenario, dict\_of\_searching\_var)

print\_searching\_values(list\_of\_searching)

elif (input\_command == '6'):

print\_info(information())

elif (input\_command is not str(range(6))):

comndErr()

sys.exit()

menu()

request()

my\_inf(словник для інформації)

parametrs = {

"stadium": [("id\_stadium", "int"), ("adress", "str"), ("number\_of\_seats", "int")],

"football\_match": [("id\_football\_match", "int"), ("who\_is\_play", "str"), ("start\_match", "time"), ("end\_match", "time"), ("stadium", "int")],

"ticket": [("id\_ticket", "int"), ("cost", "int"), ("seat\_in\_the\_stadium", "int"), ("material", "str"), ("football\_match", "int")]

}

my\_inf = {

"random":

{

"stadium":"INSERT INTO stadium (parametrs) SELECT chr(trunc(65+random()\*25)::int), trunc(random()\*1000)::int",

"football\_match":"INSERT INTO football\_match (parametrs) SELECT chr(trunc(65+random()\*25)::int), "

"timestamp '1970-01-10 20:00:00'+ random() \* (timestamp '2033-01-20 20:00:00' - timestamp '1970-01-10 10:00:00'), "

"timestamp '1970-01-10 20:00:00'+ random() \* (timestamp '2033-01-20 20:00:00' - timestamp '1970-01-10 10:00:00'), "

"id\_stadium From stadium order by random() limit 1",

"ticket":"INSERT INTO ticket (parametrs) SELECT trunc(random()\*1000)::int,trunc(random()\*1000)::int,"

"chr(trunc(65+random()\*25)::int),id\_football\_match FROM football\_match order by random() limit 1"

},

"delete":"delete FROM table\_name WHERE str\_of\_based\_column",

"add\_inf":"INSERT INTO table\_name (parametrs) VALUES (values)",

"upd\_inf":"UPDATE table\_name SET str\_of\_updating\_column WHERE str\_of\_based\_column;",

"inf":"SELECT param FROM table",

"presearch":

[

"SELECT \* FROM stadium, ticket WHERE ",

"SELECT id\_football\_match,id\_ticket,material,cost FROM football\_match, ticket WHERE ",

"SELECT \* FROM football\_match,ticket,stadium WHERE "

]

}

model.py(логіка програми)

**import** psycopg2

**import** my\_inf

**def** decorator\_for\_go\_to\_database(func):

**def** wrapper(\*args, \*\*kwargs):

con = psycopg2.connect(

database="football ticket sales service",

user="postgres",

password="password",

host="localhost",

port="5432"

)

con.set\_session(autocommit=True)

curso\_r = con.cursor()

some\_shit = func(con, curso\_r, \*args, \*\*kwargs)

curso\_r.close()

con.close()

**return** some\_shit

**return** wrapper

@decorator\_for\_go\_to\_database

**def** random(con, cursor\_r, table\_name, n):

param = ",".join([\*map(**lambda** x:my\_inf.parametrs[table\_name][x][0], range(1, len(my\_inf.parametrs[table\_name])))])

**try**:

**for** i **in** range(int(n)):

take\_sql\_string\_to\_two\_stream(cursor\_r, my\_inf.my\_inf["random"][table\_name].replace("parametrs", param))

**except** psycopg2.Error **as** err:

**print**(err.pgcode)

**print**(f'WARNING:Error {err}')

@decorator\_for\_go\_to\_database

**def** add\_information(con, cursor\_r, table\_name, mass):

param = ",".join([\*map(**lambda** x: my\_inf.parametrs[table\_name][x][0], range(1, len(my\_inf.parametrs[table\_name])))])

mass = [\*map(**lambda** x: "'{0}'".format(x), mass)]

values = ",".join(mass)

**try**:

take\_sql\_string\_to\_two\_stream(cursor\_r, my\_inf.my\_inf["add\_inf"].replace("table\_name", table\_name).replace("parametrs", param).replace("values", values))

**except** psycopg2.Error **as** err:

**print**(err.pgcode)

**print**(f'WARNING:Error {err}')

@decorator\_for\_go\_to\_database

**def** delete(con, cursor\_r, table\_name, str\_of\_based\_column):

**try**:

take\_sql\_string\_to\_two\_stream(cursor\_r, my\_inf.my\_inf["delete"].replace("str\_of\_based\_column", str\_of\_based\_column).replace("table\_name", table\_name))

**except** psycopg2.Error **as** err:

**print**(err.pgcode)

**print**(f'WARNING:Error {err}')

@decorator\_for\_go\_to\_database

**def** search(con, cursor\_r, scenario, dict\_of\_searching\_var):

list\_of\_commands = []

**for** key, value **in** dict\_of\_searching\_var.items():

**if**(isinstance(value, list)):

list\_of\_commands.append(key + " > " + value[0] + " AND " + key + " < " + value[1])

**else**:

list\_of\_commands.append(key + " LIKE " + value)

commands = ' AND '.join(list\_of\_commands)

**try**:

take\_sql\_string\_to\_two\_stream(cursor\_r, my\_inf.my\_inf["presearch"][scenario] + commands)

searching\_values = cursor\_r.fetchall()

**except** psycopg2.Error **as** err:

**print**(err.pgcode)

**print**(f'WARNING:Error {err}')

**else**:

**return** searching\_values

@decorator\_for\_go\_to\_database

**def** update(con, cursor\_r, table\_name, str\_of\_updating\_column, str\_of\_based\_column):

**try**:

take\_sql\_string\_to\_two\_stream(cursor\_r, my\_inf.my\_inf["upd\_inf"].replace("table\_name", table\_name).replace("str\_of\_updating\_column", str\_of\_updating\_column).replace("str\_of\_based\_column", str\_of\_based\_column))

**except** psycopg2.Error **as** err:

**print**(err.pgcode)

**print**(f'WARNING:Error {err}')

@decorator\_for\_go\_to\_database

**def** information(con, cursor\_r):

dict\_of\_all\_tables = {}

**try**:

**for** table **in** my\_inf.parametrs:

dict\_of\_all\_tables[table] = []

flag\_of\_take\_memory = 1

**for** param **in** my\_inf.parametrs[table]:

param = param[0]

cursor\_r.execute(my\_inf.my\_inf["inf"].replace("table", table).replace("param", param))

list\_of\_all\_parametrs = cursor\_r.fetchall()

**if** (flag\_of\_take\_memory):

**for** memory\_take **in** range(len(list\_of\_all\_parametrs)):

dict\_of\_all\_tables[table].append({})

flag\_of\_take\_memory = 0

**for** i **in** range(len(list\_of\_all\_parametrs)):

dict\_of\_all\_tables[table][i][param] = list\_of\_all\_parametrs[i][0]

flag\_of\_take\_memory = 1

**except** psycopg2.Error **as** err:

**print**(err.pgcode)

**print**(f'WARNING:Error {err}')

**else**:

**return** dict\_of\_all\_tables

**def** take\_sql\_string\_to\_two\_stream(cursor\_r, sql\_stream):

**print**(sql\_stream)

cursor\_r.execute(sql\_stream)

requirements.txt(використанні ліби)

attrs==22.1.0

colorama==0.4.6

exceptiongroup==1.0.1

iniconfig==1.1.1

packaging==21.3

pluggy==1.0.0

psycopg2==2.9.5

pyparsing==3.0.9

tomli==2.0.1