

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”

Виконав:

студент ІII курсу

групи КВ-02

Єгоров М. А.

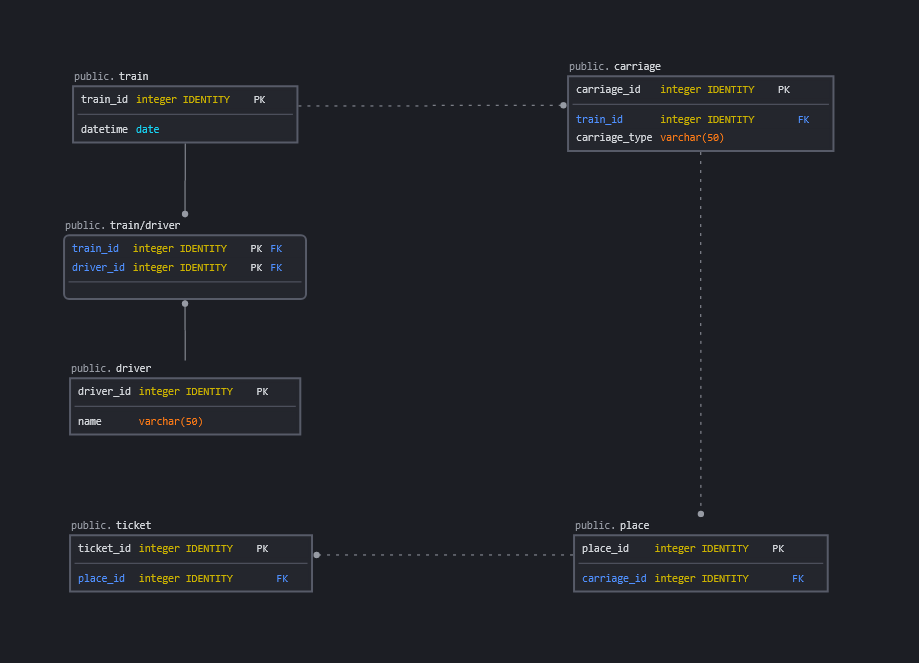
Перевірив: Петрашенко А. В.

Київ – 2022

**Мета роботи:** здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних

**Виконання роботи**

Логічна модель бази даних

****

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

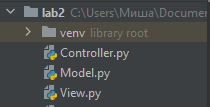
Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моій програмі відповідають всі компоненти які знаходять у папці Models.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то ій відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

**Структура програми та її опис**





Програма умовно поділена на 3 модулі: файл Controller.py, файл Model.py, файл View.py.

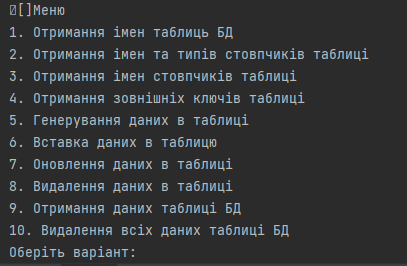
Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлі Model (.py) описаний клас моделі, що займається регулюванням підключення до бази даних, та виконанням низькорівневих запитів до неї.

У файлі Controller (.py) описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі View (.py) описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

**Структура меню програми**

****

Меню програми

Меню для користувача складається з дев’яти пунктів (Рисунок 2).

Перший пункт пропонує отримання імен таблиць бази даних

Другий пункт пропонує отримання імен та типів стовпчиків таблиці

Третій пункт пропонує отримання імен стовпчиків таблиці

Четвертий пункт пропонує отримання зовнішніх ключів таблиці

П’ятий пункт пропонує генерування даних в таблиці

Шостий пункт пропонує вставку даних в таблицю

Сьомий пункт пропонує оновлення даних в таблиці

Восьмий пункт пропонує видалення даних з таблиці

Дев’ятий пункт пропонує отримання даних таблиці

Десятий пункт пропонує видалення всіх даних таблиці БД

**Фрагмент коду (файл Controller.py), в якому наведено виклик функцій та значення, які їм передаються**

**import** View

**import** Model

**import** time

**while** 1==1:

View.hello()

choice = input("Оберіть варіант: ")

model = Model.db\_model("lab", "postgres", "111", "")

match choice:

case "1":

mas = model.get\_table\_names()

View.show(mas)

time.sleep(4)

case "2":

table = input("Введіть назву таблиці: ")

mas = model.get\_column\_types(table)

View.show(mas)

time.sleep(4)

case "3":

table = input("Введіть назву таблиці: ")

mas = model.get\_column\_names(table)

View.show(mas)

time.sleep(4)

case "4":

table = input("Введіть назву таблиці: ")

mas = model.get\_foreign\_key\_info(table)

View.show(mas)

time.sleep(4)

case "5":

table = input("Введіть назву таблиці: ")

count = input("Введіть count: ")

model.generate\_data(table, count)

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "6":

table = input("Введіть назву таблиці: ")

columns = input("Введіть назви колонок: ").split(' ')

val = input("Введіть відповідні значення: ").split(' ')

values = {key:value **for** (key,value) **in** zip(columns,val)}

model.insert\_data(table, values)

**print**("result:**\n**")

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "7":

table = input("Введіть назву таблиці: ")

columns = input("Введіть назви колонок: ").split(' ')

val = input("Введіть відповідні значення: ").split(' ')

values = {key:value **for** (key,value) **in** zip(columns,val)}

model.change\_data(table, values)

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "8":

table = input("Введіть назву таблиці: ")

column = input("Назва колонки: ")

param = input("Параметр: ")

model.delete\_data(table, column, param)

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "9":

table = input("Введіть назву таблиці: ")

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "10":

table = input("Введіть назву таблиці: ")

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case \_:

**print**("Error")

**Фрагменти програм внесення, редагування та вилучення даних у базі даних**

* Фрагмент програми для внесення даних
* *#Вставка даних в таблицю*
* **def** insert\_data(self, table\_name, values):
* *#Формування рядка значень VALUES запиту INSERT*
* line = ''
* columns = '('
* **for** key **in** values:
* **if** values[key]:
* line += '%(' + key + ')s,'
* columns += key + ','
* columns = columns[:-1] + ')'
* *#Виконання SQL-запиту на вставку даних в таблицю*
* self.\_\_cursor.execute(
* sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')
* .format(sql.Identifier(table\_name), sql.SQL(columns)),
* values)
* self.\_\_context.commit()
* Фрагмент програми для редагування даних
* *#Оновлення даних в таблиці*
* **def** change\_data(self, table\_name, values):
* *#Формування рядка значень SET запиту UPDATE*
* line = ''
* condition = values.pop('condition')
* **for** key **in** values:
* **if** values[key]:
* line += key + '=%(' + key + ')s,'
* *#Виконання SQL-запиту на оновлення даних в таблиці*
* self.\_\_cursor.execute(
* sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ')
* .format(sql.Identifier(table\_name), sql.SQL(condition)),
* values)
* self.\_\_context.commit()
* Фрагмент програми для видалення даних
* *#Видалення даних в таблиці*
* **def** delete\_data(self, table\_name, value, cond):
* self.\_\_cursor.execute(
* sql.SQL('DELETE FROM {} WHERE {} = {}')
* .format(sql.Identifier(table\_name), sql.Identifier(value), sql.SQL(cond)))
* self.\_\_context.commit()
* **def** delete\_all\_data(self, table\_name):
* self.\_\_cursor.execute(
* sql.SQL('DELETE \* FROM {}')
* .format(sql.Identifier(table\_name)))
* self.\_\_context.commit()
* **def** join\_general(self, main\_query, condition=""):
* new\_cond = condition
* **if** condition:
* new\_cond = "WHERE " + condition
* t1 = time.time()
* self.\_\_cursor.execute(main\_query.format(new\_cond))
* t2 = time.time()
* **return** ((t2 - t1) \* 1000, self.\_\_cursor.fetchall())
* Фрагмент для генерації випадкових значень
* **def** generate\_data(self, table\_name, count):
* types = self.get\_column\_types(table\_name)
* fk\_array = self.get\_foreign\_key\_info(table\_name)
* select\_subquery = ""
* insert\_query = "INSERT INTO " + table\_name + " ("
* **for** i **in** range(1, len(types)):
* t = types[i]
* name = t[0]
* type = t[1]
* fk = [x **for** x **in** fk\_array **if** x[0] == name]
* **if** fk:
* select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))
* **elif** type == 'integer':
* select\_subquery += 'trunc(random()\*100)::INT'
* **elif** type == 'character varying':
* select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + ran-dom()\*25)::INT)'
* **elif** type == 'date':
* select\_subquery += """ date(timestamp '2014-01-10' +
* random() \*
* (timestamp '2020-01-20' - timestamp '2014-01-10'))"""
* **elif** type == 'time without time zone':
* select\_subquery += "time '00:00:00' + DATE\_TRUNC('second',RANDOM() \* time '24:00:00')"
* **else**:
* **continue**
* insert\_query += name
* **if** i != len(types) - 1:
* select\_subquery += ','
* insert\_query += ','
* **else**:
* insert\_query += ') '
* self.\_\_cursor.execute(
* insert\_query + "SELECT " + select\_subquery +
* "FROM generate\_series(1," + str(count) + ") as ser")
* self.\_\_context.commit()

Дані фрагменти програми, які наведені вижче, відповідають за функціонал додвання даних, редагування та вилучення даних у базі даних.

Взаємодія відбувається через клас Model, який займається підключенням до БД, а самі функції знаходяться у файлі Controller.

**Повний текст програми:**

Модуль Controller.py

**import** View

**import** Model

**import** time

**while** 1==1:

View.hello()

choice = input("Оберіть варіант: ")

model = Model.db\_model("lab", "postgres", "111", "")

match choice:

case "1":

mas = model.get\_table\_names()

View.show(mas)

time.sleep(4)

case "2":

table = input("Введіть назву таблиці: ")

mas = model.get\_column\_types(table)

View.show(mas)

time.sleep(4)

case "3":

table = input("Введіть назву таблиці: ")

mas = model.get\_column\_names(table)

View.show(mas)

time.sleep(4)

case "4":

table = input("Введіть назву таблиці: ")

mas = model.get\_foreign\_key\_info(table)

View.show(mas)

time.sleep(4)

case "5":

table = input("Введіть назву таблиці: ")

count = input("Введіть count: ")

model.generate\_data(table, count)

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "6":

table = input("Введіть назву таблиці: ")

columns = input("Введіть назви колонок: ").split(' ')

val = input("Введіть відповідні значення: ").split(' ')

values = {key:value **for** (key,value) **in** zip(columns,val)}

model.insert\_data(table, values)

**print**("result:**\n**")

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "7":

table = input("Введіть назву таблиці: ")

columns = input("Введіть назви колонок: ").split(' ')

val = input("Введіть відповідні значення: ").split(' ')

values = {key:value **for** (key,value) **in** zip(columns,val)}

model.change\_data(table, values)

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "8":

table = input("Введіть назву таблиці: ")

column = input("Назва колонки: ")

param = input("Параметр: ")

model.delete\_data(table, column, param)

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "9":

table = input("Введіть назву таблиці: ")

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case "10":

table = input("Введіть назву таблиці: ")

mas = model.get\_table\_data(table)

View.show(mas)

time.sleep(4)

case \_:

**print**("Error")

Модуль Model.py

**import** psycopg2

**from** psycopg2 **import** sql

**import** time

**class** db\_model():

**def** \_\_init\_\_(self, dbname, user\_name, password, host):

self.\_\_context = psycopg2.connect(dbname=dbname, user=user\_name, password=password, host=host)

self.\_\_cursor = self.\_\_context.cursor()

self.\_\_table\_names = None

**return** None

**def** \_\_del\_\_(self):

self.\_\_cursor.close()

self.\_\_context.close()

**def** clear\_transaction(self):

self.\_\_context.rollback()

*#Отримання імен таблиць БД*

**def** get\_table\_names(self):

**if** self.\_\_table\_names **is** None:

self.\_\_cursor.execute("""SELECT table\_name

FROM information\_schema.tables

WHERE table\_schema = 'public'""")

self.\_\_table\_names = [table[0] **for** table **in** self.\_\_cursor]

**return** self.\_\_table\_names

*#Отримання імен та типів стовпчиків таблиці*

**def** get\_column\_types(self, table\_name):

self.\_\_cursor.execute("""SELECT column\_name, data\_type

FROM information\_schema.columns

WHERE table\_schema = 'public' AND table\_name = %s

ORDER BY table\_schema, table\_name""", (table\_name,))

**return** self.\_\_cursor.fetchall()

*#Отримання імен стовпчиків таблиці*

**def** get\_column\_names(self, table\_name):

self.\_\_cursor.execute("""

SELECT column\_name FROM information\_schema.columns

WHERE table\_schema = 'public' AND table\_name = %s

ORDER BY table\_schema, table\_name""", (table\_name,))

**return** [x[0] **for** x **in** self.\_\_cursor.fetchall()]

*#Отримання зовнішніх ключів таблиці*

**def** get\_foreign\_key\_info(self, table\_name):

self.\_\_cursor.execute("""

SELECT kcu.column\_name, ccu.table\_name AS

foreign\_table\_name,

ccu.column\_name AS foreign\_column\_name

FROM information\_schema.table\_constraints AS tc

JOIN information\_schema.key\_column\_usage AS kcu

ON tc.constraint\_name = kcu.constraint\_name

AND tc.table\_schema = kcu.table\_schema

JOIN information\_schema.constraint\_column\_usage AS ccu

ON ccu.constraint\_name = tc.constraint\_name

AND ccu.table\_schema = tc.table\_schema

WHERE tc.constraint\_type = 'FOREIGN KEY' AND

tc.table\_name=%s;""", (table\_name,))

**return** self.\_\_cursor.fetchall()

*#Отримання даних таблиці БД*

**def** get\_table\_data(self, table\_name):

*#Отримання Id таблиці*

id\_column = self.get\_column\_types(table\_name)[0][0]

cursor = self.\_\_cursor

**try**:

*#Отримання даних таблиці по ключу*

cursor.execute(

sql.SQL('SELECT \* FROM {} ORDER BY {} ASC').format(sql.Identifier(table\_name), sql.SQL(id\_column)))

**except** Exception **as** e:

**return** str(e)

**return** ([col.name **for** col **in** cursor.description], cursor.fetchall())

*#Вставка даних в таблицю*

**def** insert\_data(self, table\_name, values):

*#Формування рядка значень VALUES запиту INSERT*

line = ''

columns = '('

**for** key **in** values:

**if** values[key]:

line += '%(' + key + ')s,'

columns += key + ','

columns = columns[:-1] + ')'

*#Виконання SQL-запиту на вставку даних в таблицю*

self.\_\_cursor.execute(

sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')

.format(sql.Identifier(table\_name), sql.SQL(columns)),

values)

self.\_\_context.commit()

*#Генерація випадкових даних і заповнення таблиці*

**def** generate\_data(self, table\_name, count):

types = self.get\_column\_types(table\_name)

fk\_array = self.get\_foreign\_key\_info(table\_name)

select\_subquery = ""

insert\_query = "INSERT INTO " + table\_name + " ("

**for** i **in** range(1, len(types)):

t = types[i]

name = t[0]

type = t[1]

fk = [x **for** x **in** fk\_array **if** x[0] == name]

**if** fk:

select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))

**elif** type == 'integer':

select\_subquery += 'trunc(random()\*100)::INT'

**elif** type == 'character varying':

select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + ran-dom()\*25)::INT)'

**elif** type == 'date':

select\_subquery += """ date(timestamp '2014-01-10' +

random() \*

(timestamp '2020-01-20' - timestamp '2014-01-10'))"""

**elif** type == 'time without time zone':

select\_subquery += "time '00:00:00' + DATE\_TRUNC('second',RANDOM() \* time '24:00:00')"

**else**:

**continue**

insert\_query += name

**if** i != len(types) - 1:

select\_subquery += ','

insert\_query += ','

**else**:

insert\_query += ') '

self.\_\_cursor.execute(

insert\_query + "SELECT " + select\_subquery +

"FROM generate\_series(1," + str(count) + ") as ser")

self.\_\_context.commit()

*#Оновлення даних в таблиці*

**def** change\_data(self, table\_name, values):

*#Формування рядка значень SET запиту UPDATE*

line = ''

condition = values.pop('condition')

**for** key **in** values:

**if** values[key]:

line += key + '=%(' + key + ')s,'

*#Виконання SQL-запиту на оновлення даних в таблиці*

self.\_\_cursor.execute(

sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ')

.format(sql.Identifier(table\_name), sql.SQL(condition)),

values)

self.\_\_context.commit()

*#Видалення даних в таблиці*

**def** delete\_data(self, table\_name, value, cond):

self.\_\_cursor.execute(

sql.SQL('DELETE FROM {} WHERE {} = {}')

.format(sql.Identifier(table\_name), sql.Identifier(value), sql.SQL(cond)))

self.\_\_context.commit()

**def** delete\_all\_data(self, table\_name):

self.\_\_cursor.execute(

sql.SQL('DELETE \* FROM {}')

.format(sql.Identifier(table\_name)))

self.\_\_context.commit()

**def** join\_general(self, main\_query, condition=""):

new\_cond = condition

**if** condition:

new\_cond = "WHERE " + condition

t1 = time.time()

self.\_\_cursor.execute(main\_query.format(new\_cond))

t2 = time.time()

**return** ((t2 - t1) \* 1000, self.\_\_cursor.fetchall())

Модуль View.py

**import** os

clear = **lambda**: os.system('cls')

**def** hello():

clear()

**print**("[]Меню")

**print**("1. Отримання імен таблиць БД")

**print**("2. Отримання імен та типів стовпчиків таблиці")

**print**("3. Отримання імен стовпчиків таблиці")

**print**("4. Отримання зовнішніх ключів таблиці")

**print**("5. Генерування даних в таблиці")

**print**("6. Вставка даних в таблицю")

**print**("7. Оновлення даних в таблиці")

**print**("8. Видалення даних в таблиці")

**print**("9. Отримання даних таблиці БД")

**print**("10. Видалення всіх даних таблиці БД")

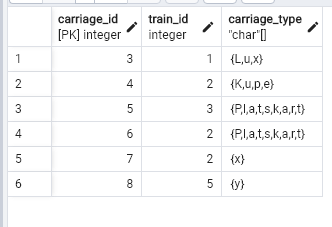
**def** show(mas):

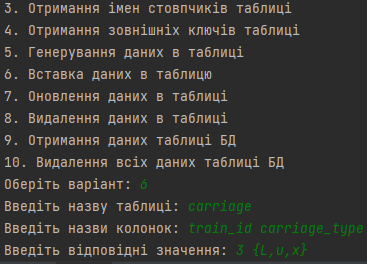
**for** element **in** mas:

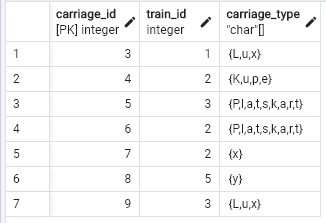
**print**(element)

**print**("**\n**");

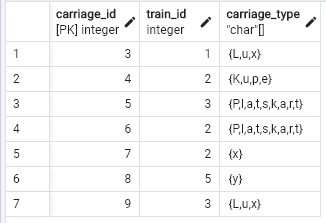
**Скріншоти результатів виконання операції вставки запису в дочірню таблицю**

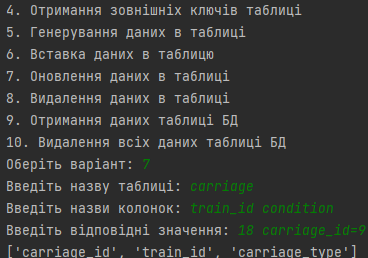


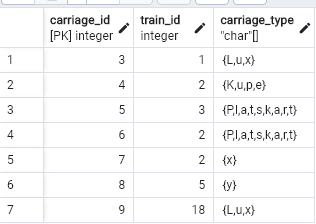




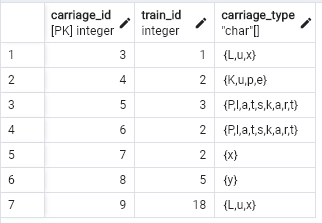
**Скріншоти результатів виконання операції редагування таблиці**

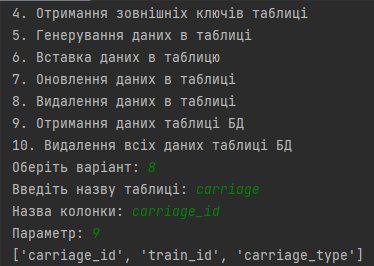


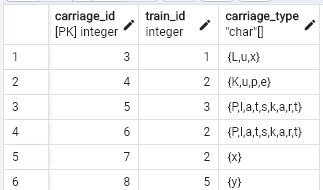




**Скріншоти результатів виконання операції видалення**







**Скріншоти результатів виконання операції генерування**

