

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

Шпот І.Р.

Перевірив:

Київ – 2022

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

*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).

**Логічна модель (схема) БД «Школа»**

Логічна модель (схему бази даних) наведено на Рисунок 1.

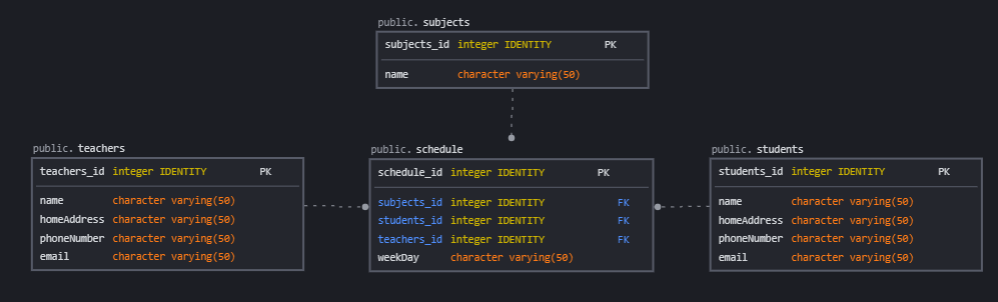


Рисунок 1 – Схема бази даних (інструмент sqldmb.com)

У базі даних були змінені назви таблиць та деяких атрибутів. Зміни полягають в тому, що всі назви були переведені в нижній регістр. Тобто Subjects → subjects, Teachers → teachers, Students → students, homeAddress → home\_address, phoneNumber → phone\_number та weekDay → week\_day.

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувались мова програмування Python 3.10 та середовище розробки IDLE (Python 3.10 64-bit)

**Шаблон проектування програмного додатку**

MVC – шаблон проектування, який використовується у програмі

Model – клас, що описує логіку використовуваних даних. Реалізований у файлі model.py

View – клас, що відповідає за ввід та вивід інформації. Реалізований у файлі view.py

Controller – клас, що оброблює ввід користувача, подає відповідну команду користувача до класу Model та виводить результат команди за допомогою класу View. Реалізований у файлі controller.py

**Структура програми та її опис**

Структуру програми наведено на Рисунок 2

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 2 – структура програми

Програма складається з 4 модулів: main.py, model.py, view.py, controller.py.

main.py – модуль початку програми, що викликає функцію для підключення до БД та подає команду на вивід меню.

model.py – модуль, що реалізовує підключення та обробку та виконання запитів до БД.

view.py – модуль, що реалізовує вивід меню, повідомлення про виконання дій та елементів таблиць на консоль.

controller.py – модуль, що оброблює ввід користувача, подає відповідні команди до модулю model.py та виводить результати за допомогою модуля view.py

**Структура меню програми**

Структуру головного меню програми наведено на Рисунок 3.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 3 – структура головного меню програми

Структури усіх меню перегляду даних таблиці, додавання даних у таблицю, видалення даних з таблиці, оновлення даних в таблиці, генерування нових даних у таблицю однакові і полягають у виведені назв існуючих таблиці бази даних у консоль. Приклад наведено на Рисунок 4.

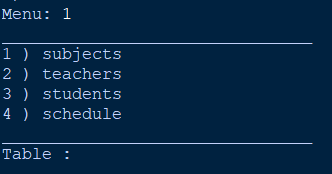


Рисунок 4

Перший пункт «Show tables» виконує перегляд та вивід даних обраної таблиці БД. Для вибору таблиці потрібно ввести номер, що відповідає таблиці, яку необхідно вивести.

Другий пункт «Insert» виконує внесення даних в обрану таблицю БД. Для вибору таблиці потрібно ввести номер, що відповідає таблиці, в яку потрібно занести дані. Після введення номеру потрібно буде заповнити колонки відповідної таблиці.

Третій пункт «Delete» виконує видалення даних з обраної таблиці БД. Для вибору таблиці потрібно ввести номер, що відповідає таблиці, з якої необхідно видалити дані. Після введення номеру потрібно буде ввести номер первинного ключа елемента, що необхідно видалити.

Четвертий пункт «Update» виконує редагування даних в обраній таблиці БД. Для вибору таблиці потрібно ввести номер, що відповідає таблиці, в які необхідно редагувати дані. Після введення номеру таблиці необхідно обрати колонку, дані якої необхідно редагувати, далі занести нові дані та написати номер первинного ключа елемента, дані якого в тій колонці необхідно редагувати

П’ятий пункт «Generate» виконує створення заданої кількості нових елементів заповнених “випадковими” даними в обраній таблиці. Для вибору таблиці потрібно ввести номер, що відповідає таблиці, в яку необхідно створити нові елементи. Після введення номеру таблиці необхідно зазначити кількість нових елементів, що будуть створені.

Пункт шість «Exit» виконує зупинку роботи програми та проводить відключення від БД.

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

Фрагмент програми для внесення даних:

model.py

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

line = ''

columns = '('

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

if **values**[**key**]:

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

columns += **key** + ','

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

**self**.\_\_cursor.**execute**(

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

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

**values**)

**self**.\_\_context.**commit**()

controller.py

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

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**self**.model.insert\_data(**table\_name**, **values**)

**self**.**view**.display\_insert(**table\_name**, **values**)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to insert data to table", error)

Пояснення до роботи фрагменту програми:

Спочатку в циклі формується рядок значень для запиту INSERT. Потім за допомогою команди execute виконується SQL-запит на вставку даних в таблицю. Для роботи функції потрібно знати назву таблиці, а також рядок з колонками та їх значенням. Оскільки в моїх таблицях усі значення колонок NOT NULL, треба дати значення усім колонкам. Перехоплення такої помилки передбачено. Для простого обрання усіх колонок, в меню для значення values є можливість додати рядок «all», який заміниться на назви колонок обраної таблиці.

Фрагмент програми для редагування даних:

model.py

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

line = ''

condition = **values**.pop('condition')

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

if **values**[**key**]:

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

**self**.\_\_cursor.**execute**(

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

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

**values**)

**self**.\_\_context.**commit**()

controller.py

def **update**(**self**, **table\_name**, **values**):

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**self**.model.change\_data(**table\_name**, **values**)

**self**.**view**.display\_update(**table\_name**, **values**)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to update data in table", error)

Пояснення до роботи фрагменту програми:

Як і при внесенні даних, спочатку в циклі формується рядок значень, далі виконується SQL-запит UPDATE, який оновлює дані таблиці. Для роботи функції потрібні назва таблиці та кортеж values, в якому назви колонок, значення яких треба змінити та відповідно нові значення для них.

Фрагмент програми для видалення даних:

model.py

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**()

controller.py

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

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

if **table\_name** == 'subjects':

**self**.delete\_subjects(**table\_name**, **column**, value)

elif **table\_name** == 'teachers':

**self**.delete\_teachers(**table\_name**, **column**, value)

elif **table\_name** == 'students':

**self**.delete\_students(**table\_name**, **column**, value)

**else**:

**self**.model.delete\_data(**table\_name**, **column**, value)

**self**.**view**.display\_delete(**table\_name**, value)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to delete data in table", error)

def delete\_subjects(**self**, **table\_name**, **column**, value):

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**self**.model.delete\_data('schedule', 'subjects\_id', value)

**self**.model.delete\_data(**table\_name**, **column**, value)

**self**.**view**.display\_delete(**table\_name**, value)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to delete data from table subjects", error)

def delete\_teachers(**self**, **table\_name**, **column**, value):

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**self**.model.delete\_data('schedule', 'teachers\_id', value)

**self**.model.delete\_data(**table\_name**, **column**, value)

**self**.**view**.display\_delete(**table\_name**, value)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to delete data from table teachers", error)

def delete\_students(**self**, **table\_name**, **column**, value):

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**self**.model.delete\_data('schedule', 'students\_id', value)

**self**.model.delete\_data(**table\_name**, **column**, value)

**self**.**view**.display\_delete(**table\_name**, value)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to delete data from table students", error)

Пояснення до роботи фрагменту програми:

Для роботи функції потрібні назва таблиці, назва PK таблиці (отримується автоматично за назвою таблиці) та його значення. SQL-запит на видалення даних виконується за допомогою запиту DELETE. Також передбачена можливість видалення батьківських даних та відповідним їм дочірнім.

**Лістинг фрагменту програми генерування елементів таблиці БД з випадковими даними**

model.py

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()\*1000)::INT'

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

select\_subquery += """chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)

|| chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*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**()

controller.py

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

try:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**self**.model.generate\_data(**table\_name**, **count**)

**self**.**view**.display\_generate(**table\_name**, **count**)

**except** (Exception, psycopg2.Error) **as** error:

print("Failed to generate data to table", error)

Пояснення до роботи фрагменту програми:

В циклі функція проходить обрану таблицю та перевіряє тип кожної колонки та в залежності від нього за допомогою запиту INSERT генерує та додає випадкове значення цього ж типу в таблицю. В моїх таблицях використовується такі типи: integer та character varying. Для роботи функції потрібні назва таблиці та яку кількість даних генерувати.

**Результати роботи програми**

«Show tables»

Результат роботи «Show tables» для значень таблиці «subjects» наведено на Рисунок 5. Дані таблиці «subjects» у застосунку pgAdmin4 наведено на Рисунок 6. Зображення ситуації виведення сповіщення про помилку при введенні значення номера таблиці, що не існує, наведено на Рисунок 7.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 5 – результат роботи «Show tables» для значень таблиці «subjects»

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 6 – дані таблиці «subjects» у застосунку pgAdmin4

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 7 – сповіщення про помилку

«Insert»

Результат роботи «Insert» для значень таблиці «teachers» наведено на Рисунок 8. Дані таблиці «teachers» у застосунку pgAdmin4 наведено на Рисунок 9. Зображення ситуації виведення сповіщення про помилку при введенні значень не для всіх існуючих колонок таблиці наведено на Рисунок 10. Зображення ситуації виведення сповіщення про помилку при введені значень не відповідного типу до колонку наведено на Рисунок 11.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 8 – результат роботи «Insert» для значень таблиці «teachers»

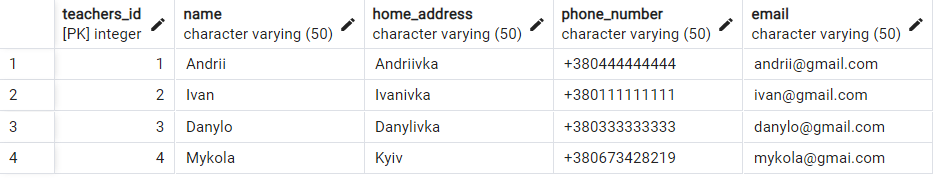


Рисунок 9 – дані таблиці «teachers» у застосунку pgAdmin4

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 10 – сповіщення про помилку

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 11 – сповіщення про помилку

«Delete»

Результат роботи «Delete» для значень таблиці «students» наведено на Рисунок 12. Дані таблиці «students» у застосунку pgAdmin4 до та після видалення наведено на Рисунок 13-14.

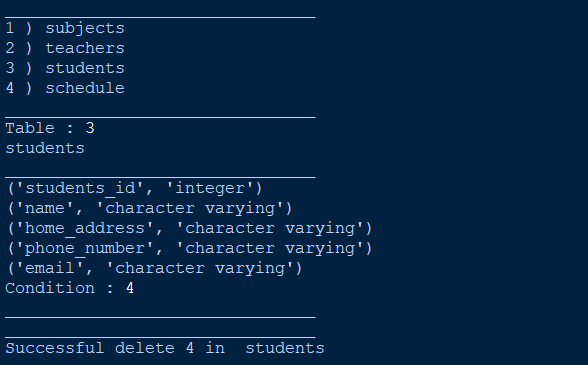


Рисунок 12 – результат роботи «Delete» для значень таблиці «students»

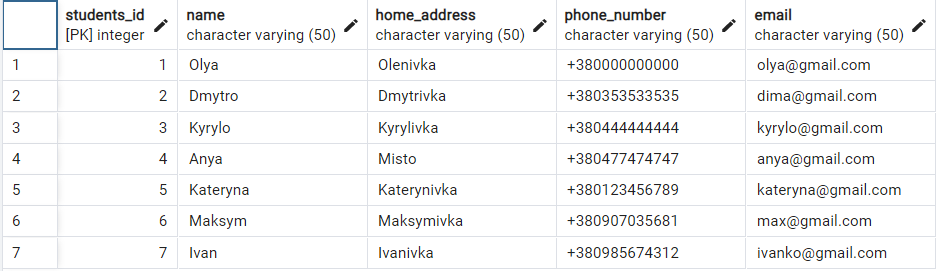


Рисунок 13 – дані таблиці «students» у застосунку pgAdmin4 до видалення

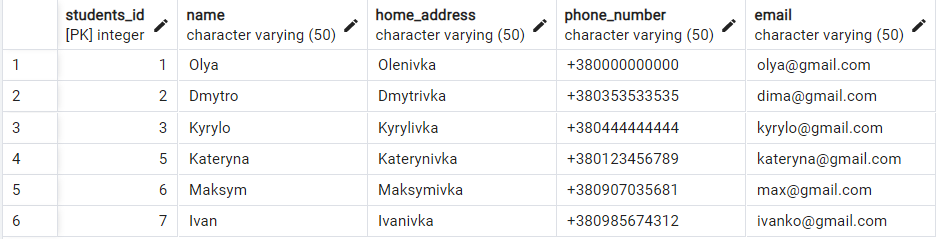


Рисунок 14 – дані таблиці «students» у застосунку pgAdmin4 після видалення

«Update»

Результат роботи «Update» для значень таблиці «students» наведено на Рисунок 15. Дані таблиці «students» у застосунку pgAdmin4 до та після редагування наведено на Рисунок 16-17.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 15 – результат роботи «Update» для значень таблиці «students»

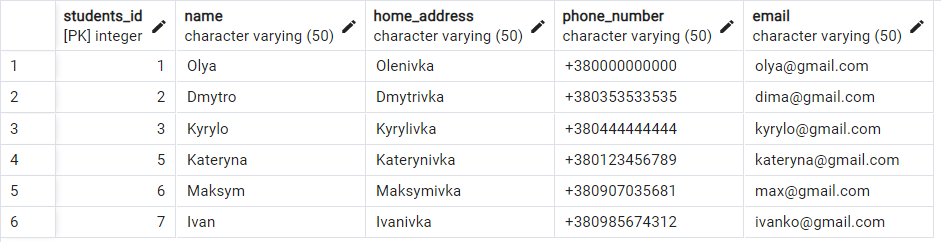


Рисунок 16 – дані таблиці «students» у застосунку pgAdmin4 до редагування



Рисунок 17 – дані таблиці «students» у застосунку pgAdmin4 після редагування

«Generate»

Результат роботи «Generate» при створені 100000 елементів для значень таблиці «schedule» наведено на Рисунок 18. Дані таблиці «schedule» у застосунку pgAdmin4 наведено на Рисунок 19.

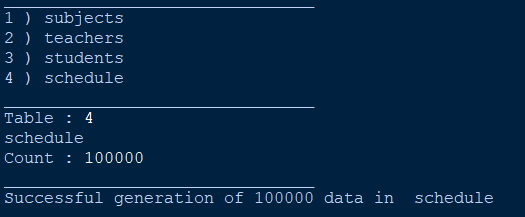


Рисунок 18 – результат роботи «Generate» для значень таблиці «schedule»

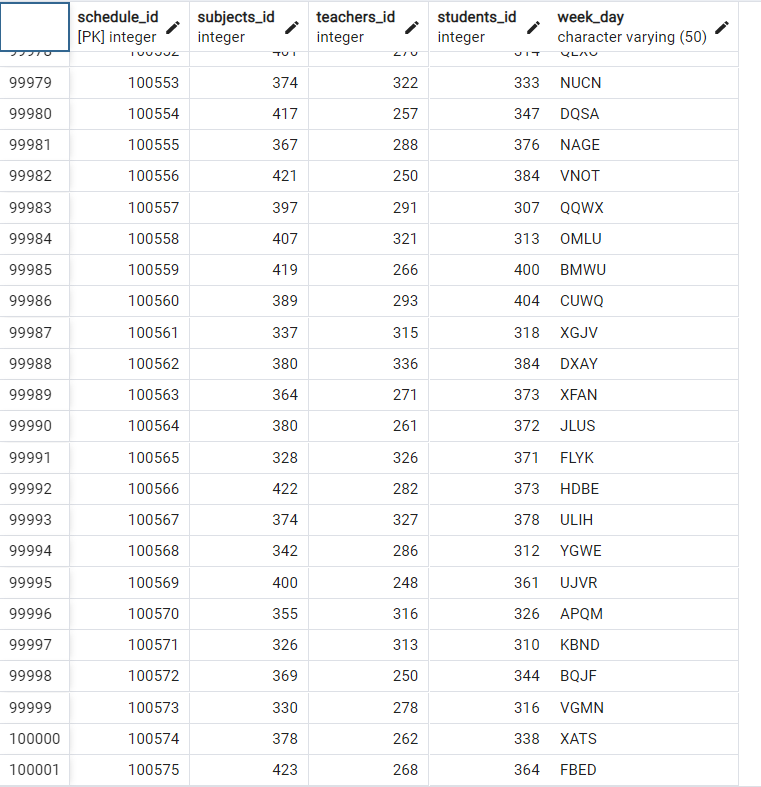


Рисунок 19 – дані таблиці «schedule» у застосунку pgAdmin4

**Результати виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці після цього вилучення**

Оскільки таблиці «subjects», «teachers» та «students» є батьківськими, а таблиця «schedule» є дочірньою до них, то нехай розглянемо вилучення елементу з таблиці «teachers». Стан таблиць наведено в Рисунок 20-23.

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 20 – дані таблиці «subjects» у застосунку pgAdmin4

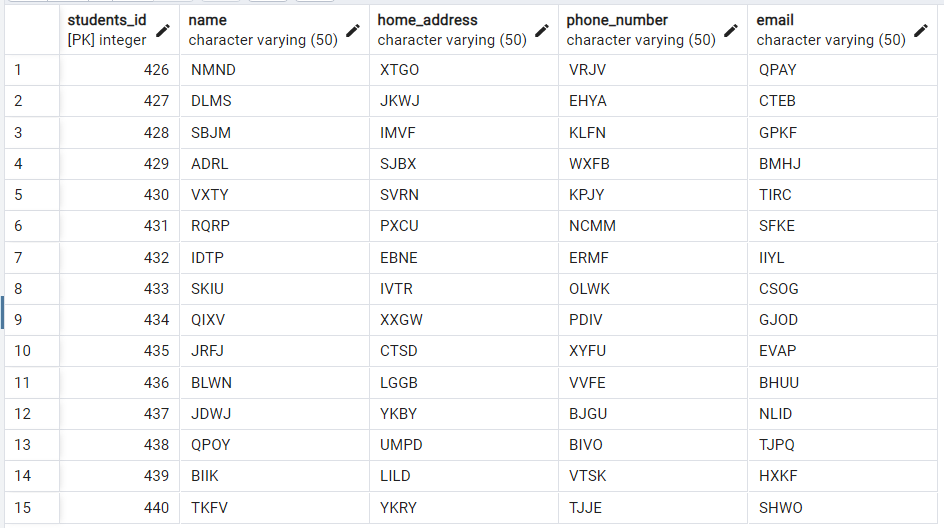


Рисунок 21– дані таблиці «students» у застосунку pgAdmin4

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 22 – дані таблиці «teachers» у застосунку pgAdmin4

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 23– дані таблиці «schedule» у застосунку pgAdmin4

Видаляємо викладача з teachers\_id = 338. Результат вилучення наведено в Рисунок 24. Стан таблиці «teachers» та«schedule» наведено на Рисунок 25-26

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 24 – результат вилучення викладача з teachers\_id = 338

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 25– дані таблиці «teachers» після вилучення

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 26 – дані таблиці «schedule» після вилучення

**Результати виконання операції вставки запису в дочірню таблицю та виведення повідомлення про її неможливість, якщо батьківській таблиці нема відповідного запису**

Оскільки ми тільки що вилучили викладача з teachers\_id = 338, то давайте створимо елемент таблиці «schedule» з цим викладачем. Результат наведено на Рисунок 27.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 27 – результат спроби внесення розкладу з викладачем з teachers\_id = 338

Тепер створимо елемент таблиці «schedule» з існуючим викладачем. Результат наведено на Рисунок 28.

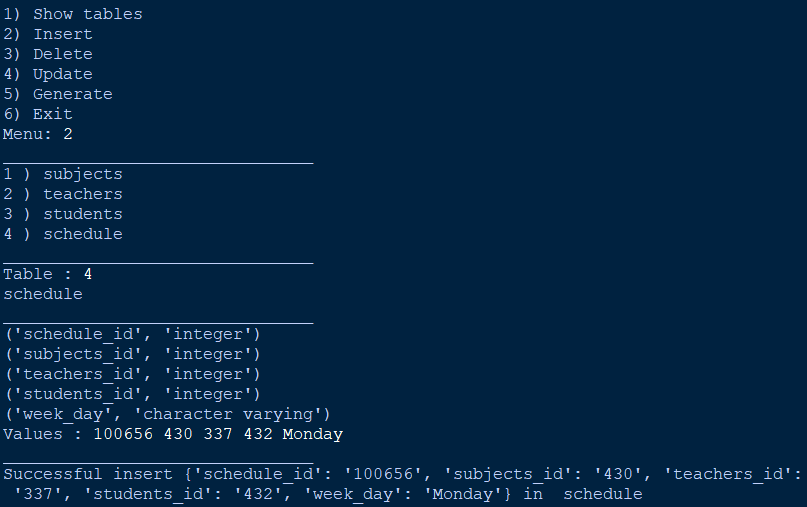


Рисунок 28 – результат внесення в таблицю «schedule» нового елемента

**Текст програми**

main.py

**from** **model** **import** Model

**from** **view** **import** View

**from** **controller** **import** Controller

prog = Controller(Model("localhost", "5432", "School", "postgres", "Pr0gr3s"), View())

prog.menu()

model.py

**import** **psycopg2**

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

**import** **time**

**class** Model:

**def** \_\_init\_\_(self, host, port, database, user, password):

self.\_\_context = psycopg2.connect(host = host, port = port,

database = database, user = user,

password = password)

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

self.\_\_table\_names = **None**

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

self.\_\_cursor.close()

self.\_\_context.close()

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

self.\_\_context.rollback()

**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

**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()

**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()]

**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()

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

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

cursor = self.\_\_cursor

**try**:

cursor.execute(

sql.SQL('SELECT \* FROM {} ORDER BY {} ASC'

).format(sql.Identifier(table\_name), sql.SQL(id\_column)))

**except** **Exception** **as** error:

**return** str(error)

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

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

line = ''

columns = '('

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

**if** values[key]:

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

columns += key + ','

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

self.\_\_cursor.execute(

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

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

values)

self.\_\_context.commit()

**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()\*1000)::INT'

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

select\_subquery += """chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)

|| chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*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()

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

line = ''

condition = values.pop('condition')

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

**if** values[key]:

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

self.\_\_cursor.execute(

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

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

values)

self.\_\_context.commit()

**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()

view.py

**class** View:

@staticmethod

**def** show\_table(db):

i = 1

**for** element **in** db:

print(i, ')', element)

i += 1

@staticmethod

**def** show(db):

**for** element **in** db:

print(element)

@staticmethod

**def** display\_menu():

print('''

1) Show tables

2) Insert

3) Delete

4) Update

5) Generate

6) Exit''')

@staticmethod

**def** display\_insert(table\_name, insert):

print('Successful insert', insert, 'in ', table\_name)

@staticmethod

**def** display\_delete(table\_name, delete):

print('Successful delete', delete, 'in ', table\_name)

@staticmethod

**def** display\_update(table\_name, update):

print('Successful update', update, 'in ', table\_name)

@staticmethod

**def** display\_generate(table\_name, count):

print('Successful generation of', count, 'data in ', table\_name)

controller.py

**import** **psycopg2**

**class** Controller:

**def** \_\_init\_\_(self, model, view):

self.model = model

self.view = view

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

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

names = self.model.get\_table\_names()

self.view.show\_table(names)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to get table names", error)

**def** get\_columns(self, table\_name):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

columns = self.model.get\_column\_types(table\_name)

self.view.show(columns)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to get table columns", error)

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

**try**:

columnNames = self.model.get\_column\_names(table\_name)

**return** columnNames

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to get table column names", error)

**def** show\_table(self, table\_name):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

table = self.model.get\_table\_data(table\_name)

self.view.show(table)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to get table data", error)

**def** digit\_to\_table\_name(self):

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

number = input('Table : ')

**if** str(number).isdigit():

**if** number == '1':

**return** 'subjects'

**elif** number == '2':

**return** 'teachers'

**elif** number == '3':

**return** 'students'

**elif** number == '4':

**return** 'schedule'

**else**:

print('Incorrect input')

self.digit\_to\_table\_name()

@staticmethod

**def** table\_name\_to\_pk(table\_name):

**if** table\_name == 'subjects':

**return** 'subjects\_id'

**elif** table\_name == 'teachers':

**return** 'teachers\_id'

**elif** table\_name == 'students':

**return** 'students\_id'

**elif** table\_name == 'schedule':

**return** 'schedule\_id'

**else**:

print('Incorrect input')

**return** ' '

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

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

self.model.insert\_data(table\_name, values)

self.view.display\_insert(table\_name, values)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to insert data to table", error)

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

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**if** table\_name == 'subjects':

self.delete\_subjects(table\_name, column, value)

**elif** table\_name == 'teachers':

self.delete\_teachers(table\_name, column, value)

**elif** table\_name == 'students':

self.delete\_students(table\_name, column, value)

**else**:

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to delete data in table", error)

**def** delete\_subjects(self, table\_name, column, value):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

self.model.delete\_data('schedule', 'subjects\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to delete data from table subjects", error)

**def** delete\_teachers(self, table\_name, column, value):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

self.model.delete\_data('schedule', 'teachers\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to delete data from table teachers", error)

**def** delete\_students(self, table\_name, column, value):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

self.model.delete\_data('schedule', 'students\_id', value)

self.model.delete\_data(table\_name, column, value)

self.view.display\_delete(table\_name, value)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to delete data from table students", error)

**def** update(self, table\_name, values):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

self.model.change\_data(table\_name, values)

self.view.display\_update(table\_name, values)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to update data in table", error)

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

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

self.model.generate\_data(table\_name, count)

self.view.display\_generate(table\_name, count)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to generate data to table", error)

**def** menu(self):

prog = **True**

**while** prog:

self.view.display\_menu()

choice = input('Menu: ')

**if** choice == '1':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

print(tn)

self.show\_table(tn)

self.model.clear\_transaction()

**elif** choice == '2':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

print(tn)

self.get\_columns(tn)

columns = self.get\_names(tn)

val = input("Values : ").split(' ')

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

self.insert(tn, values)

self.model.clear\_transaction()

**elif** choice == '3':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

print(tn)

self.get\_columns(tn)

t\_id = self.table\_name\_to\_pk(tn)

condition = input("Condition : ")

self.delete(tn, t\_id, condition)

self.model.clear\_transaction()

**elif** choice == '4':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

print(tn)

self.get\_columns(tn)

columns = (input("Column : ") + ' condition').split(' ')

val1 = input("Change : ").split(' ')

print('pk\_id : ')

val2 = (self.table\_name\_to\_pk(tn) + '=' + input('')).split(' ')

val = (val1 + val2)

print(val)

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

self.update(tn, values)

self.model.clear\_transaction()

**elif** choice == '5':

self.get\_table\_list()

tn = self.digit\_to\_table\_name()

print(tn)

count = input('Count : ')

self.random(tn, count)

self.model.clear\_transaction()

**elif** choice == '6':

prog = **False**

**else**:

print('Incorrect input')