

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

**Лабораторна робота №2**

з дисципліни

**“Бази даних та засоби управління ”**

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

Виконав: студент ІII курсу

ФПМ групи КВ-93

Нікішин Є.О.

Перевірив: Павловський В.І.

**Завдання лабораторної роботи**

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL.

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).

**Логічна модель учбової предметної області “Аптека”**

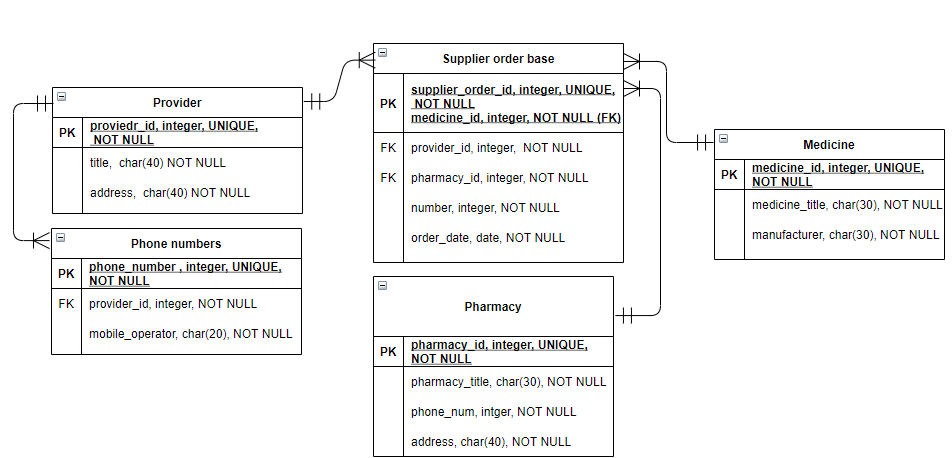


Рисунок 0 - Логічна модель учбової предметної області “Аптека”

**Середовище розробки**

Середовище розробки бази даних - PostgreSQL.

Середовище розробки програми - PyCharm Community Edition.

Мова програмування - Python(консольний застосунок).

Для зв’язку програми Python з базою даних PostgreSQL необхідно було імпортувати бібліотеку PostgreSQL Psycopg2.

**Шаблон проектування**

MVC - шаблон проектування, який використано у програмі.

Model - представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у виконаному програмному застосунку відповідають всі компоненти які записані до файлу model.py.

View - у нашому випадку консольний інтерфейс з яким буде взіємодіяти користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яких користувач вводить необхідні команди та дані, це й представлення даних у вигляді консольного інтерфейсу.

Controller - представляє клас, що забезпечує зв’язок між користувачем та системою, поданням та сховищем даних. Він отримує введені дані користувачем та виконує їх обробку. І в залежності від результату обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання. Згідно компоненту контролер, то йому відповідає файл Controller.py.

**Структура програми та її опис**

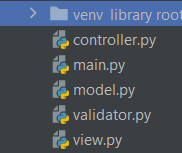
****

Рисунок 1 - структура програми

У файлі model.py (Рисунок 1) - виконуються операції з БД, такі як з’єднання, SELECT, INSERT, DELETE, UPDATE, генерація даних для таблиці та пошук за елементами.

Файл main.py (Рисунок 1) - точка входу у програму, запускає початковий інтерфейс.

У файлі controller.py (Рисунок 1) - обробляє ввід користувача, перевіряючи коректність введених даних відповідно до моделі конкретної таблиці, подає відповідно команду до model.py.

У файлі view.py (Рисунок 1) - відбувається вивід таблиць та запитів після роботи з БД, відповідно до запиту користувача

Файл validator.py (Рисунок 1) - містить функції, які виконують перевірку введених даних на відповідність до типу, який заданий у БД.

**Структура меню програми**

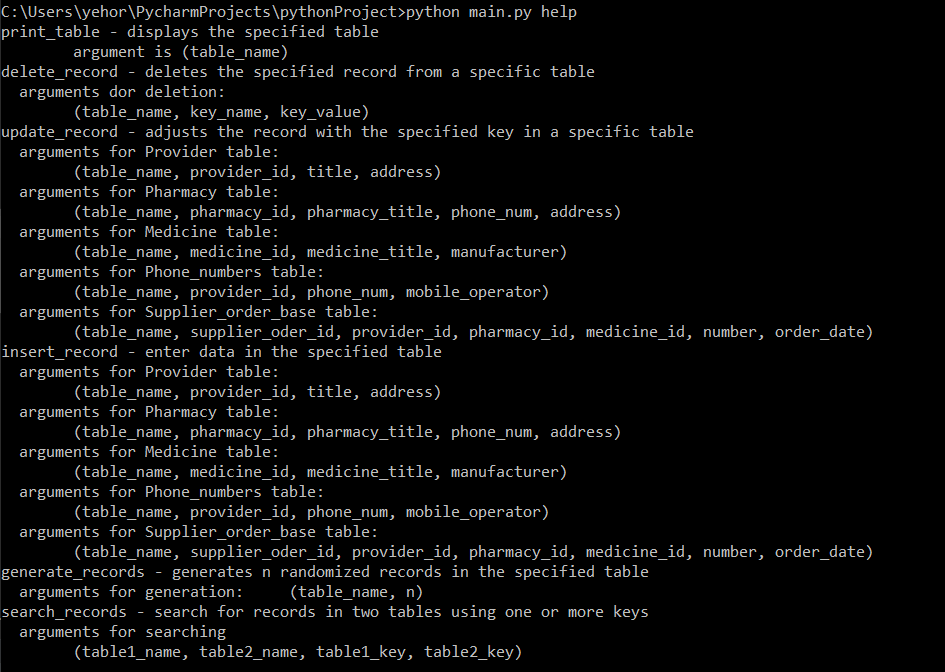
****

Рисунок 2 - ілюстрація виконання команди help.

На знімку екрану(Рисунок 2) показано виконання команди help, яка показує усі можливі запити, які може задавати користувач, коротко описує їх та надає перелік необхідних параметрів.

Кожна команда запускає відповідний метод об’єкту класу Controller, який в свою чергу реалізує передачу аргументів до класу View на їх перевірку і за умови коректного проходження перевірки ці аргументи передаються у клас Model, який і здійснює запит до бази даних.

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

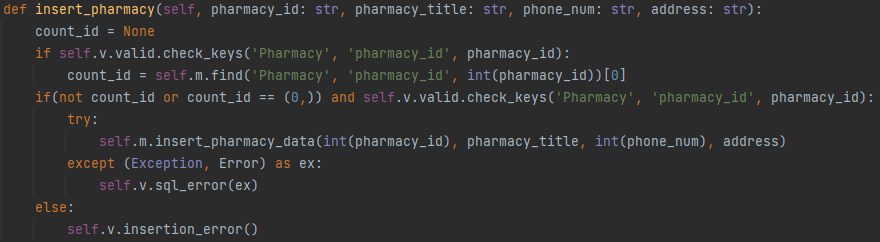


Рисунок 3 - Фрагмент програми для внесення у таблицю “Аптека”

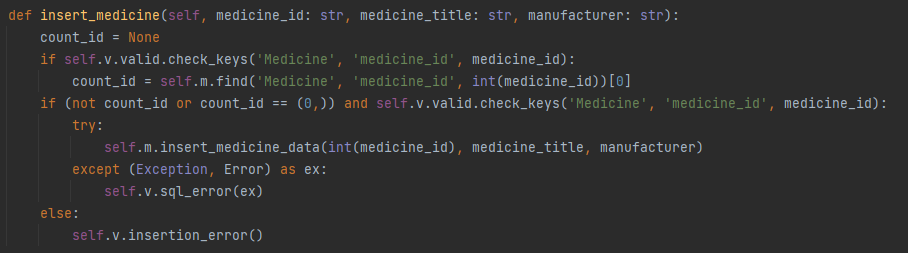


Рисунок 4 - Фрагмент програми для внесення даних у таблицю “Препарат”

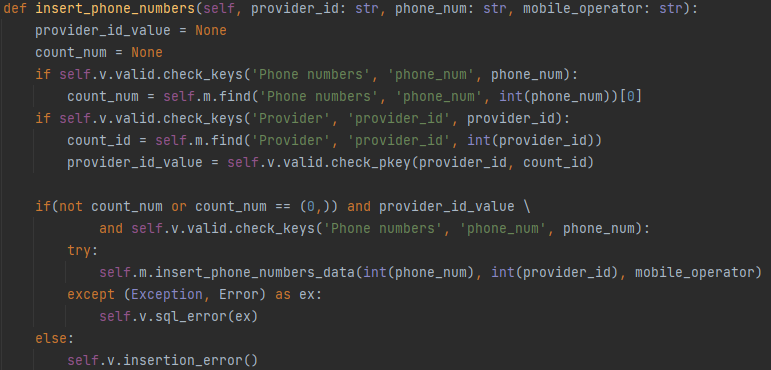
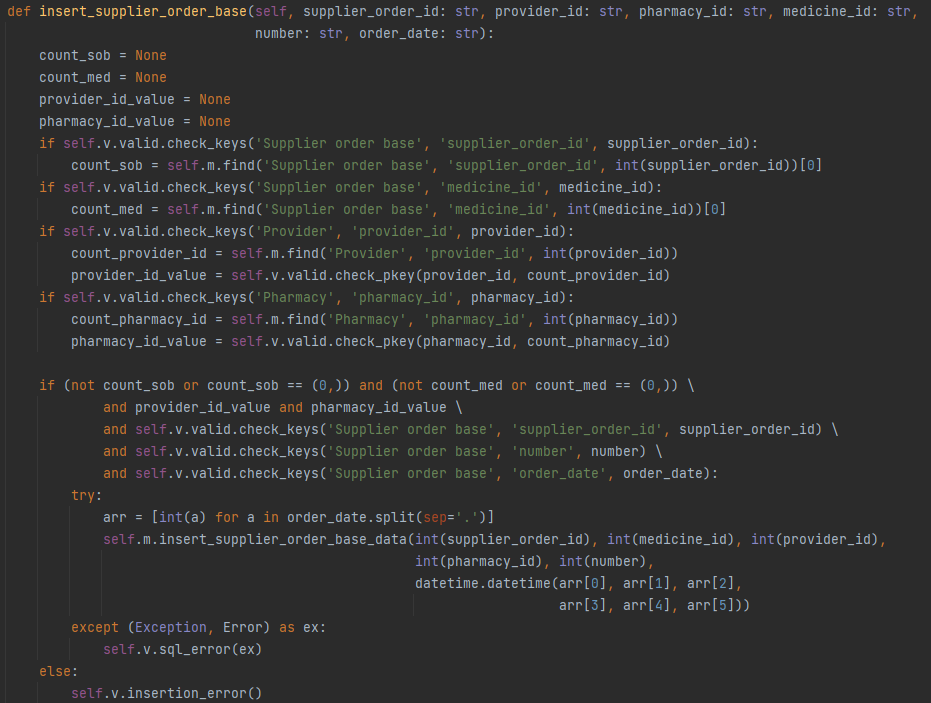


Рисунок 5 - Фрагмент програми для внесення даних у таблицю “Телефонні номери”



Рисунок 6 - Фрагмент програми для внесення даних у таблицю “Постачальник”.

  
Рисунок 7 - Фрагмент програми для внесення даних у таблицю “Замовлення для постачальника”

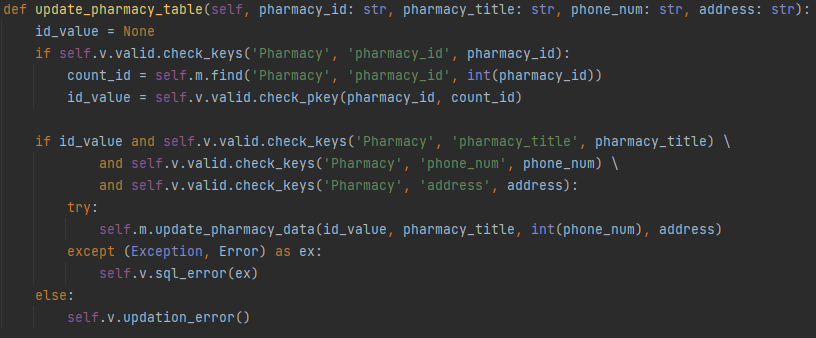


Рисунок 8 - Фрагмент програми для редагування даних у таблиці “Аптека”



Рисунок 9 - Фрагмент програми для редагування даних у таблиці “Постачальник”

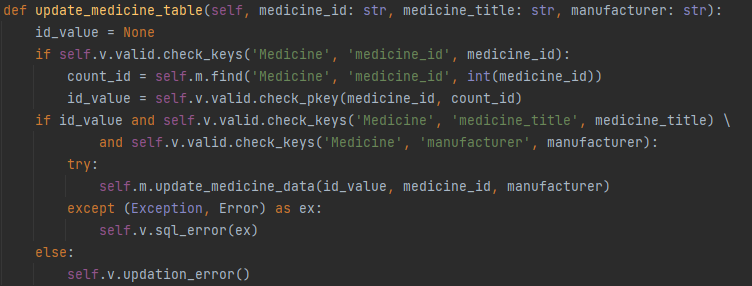


Рисунок 10 - Фрагмент програми для редагування даних у таблиці “Препарат”

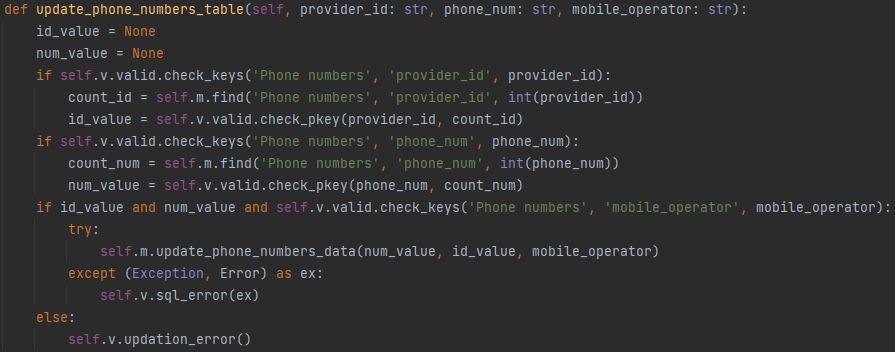


Рисунок 11 - Фрагмент програми для редагування даних у таблиці “Телефонні номери”

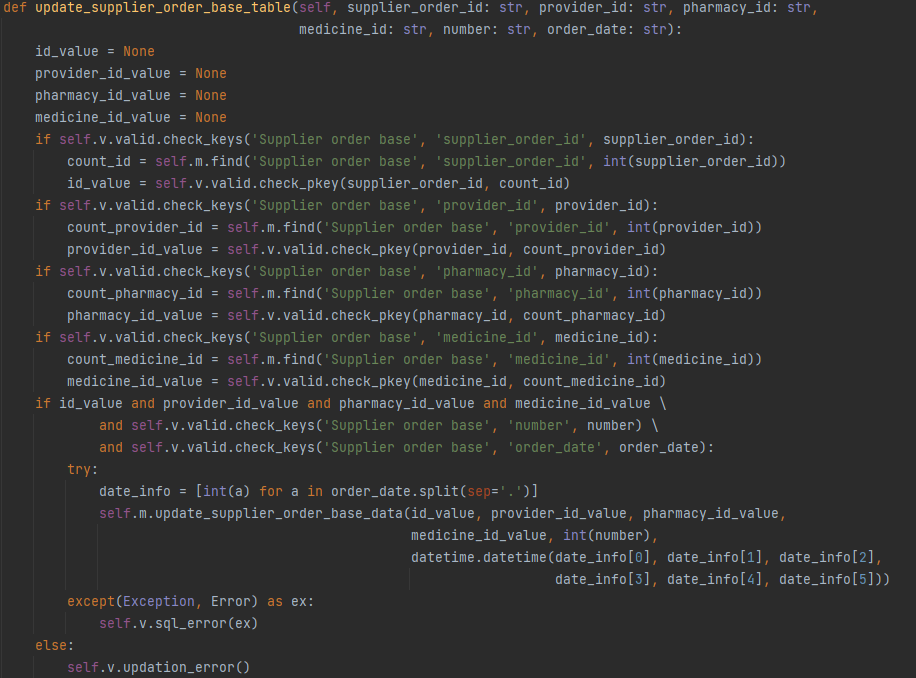
  
Рисунок 12 - Фрагмент програми для редагування даних таблиці “Замовлення для постачальника”

  
Рисунок 13 - Фрагмент програми для вилучення даних у таблиці залежно від параметрів

Функції типу insert\_[*table\_name*] - за умови коректності введених аргументів, відсутності відповідного запису з відповідним первинним ключем та наявності записів інших рядків на які хочемо посилатися зі створеного запису, вставляє новий рядок з відповідними вказаними значеннями полів у таблицю. Список аргументів для кожної з таблиць:

1. Provider - insert\_provider(provider\_id(int), title(str), address(str))
2. Pharmacy - insert\_pharmacy(pharmacy\_id(int), pharmacy\_title(str), phone\_num(int), address(str))
3. Medicine - insert\_medicine(medicine\_id(int), medicine\_title(str), manufacturer(str))
4. Phone numbers - insert\_phone\_numbers(provider\_id(int), phone\_num(int), mobile\_operator(str))
5. Supplier order base - insert\_supplier\_order\_base(suplier\_order\_id(int), provider\_id(int), pharmacy\_id(int), medicine\_id(int), number(int), order\_date(date))

Функція типу update\_[*table\_name*]\_table - аналогічно до попереднього типу функцій виконує перевірку коректності отриманих даних та існування данного первинного ключа у таблиці, й за умови валідних даних виконує заміну усіх полів рядка, окрім первинного ключа. Список аргументів для кожної з таблиць:

1. Provider - update\_provider\_table(provider\_id(int), title(str), address(str))
2. Pharmacy - update\_pharmacy\_table(pharmacy\_id(int), pharmacy\_title(str), phone\_num(int), address(str))
3. Medicine - update\_medicine\_table(medicine\_id(int), medicine\_title(str), manufacturer(str))
4. Phone numbers - update\_phone\_numbers\_table(provider\_id(int), phone\_num(int), mobile\_operator(str))
5. Supplier order base - update\_supplier\_order\_base\_table(suplier\_order\_id(int), provider\_id(int), pharmacy\_id(int), medicine\_id(int), number(int), order\_date(date))

Функція delete - виконує видалення певного рядка таблиці, за первинним ключем та його значенням, дана операція виконується за умови коректного введення аргументів, тобто існування даного запису. Аргументами для даної функції є delete(table\_name, key\_name, value)

Результати роботи даних функцій наведені у розділі “Результати програми”, які знаходяться нижче.

**Лістинги фрагментів програм з запитами пошуку**

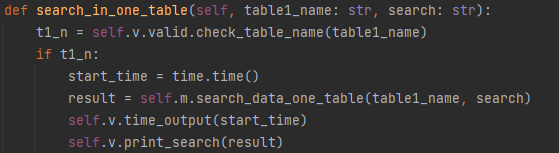
****

Рисунок 14 - Функція для пошуку в одній таблиці, за заданою кількістю атрибутів.

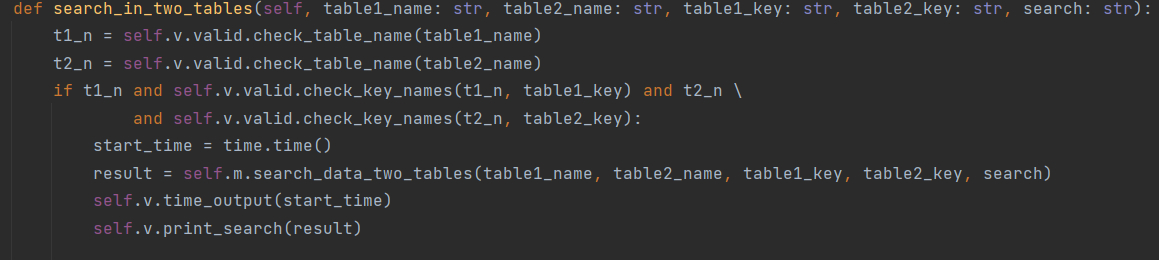
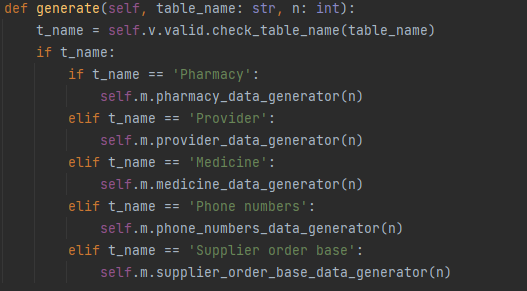


Рисунок 15 - Функція для пошуку в двох таблицях, за заданою кількістю атрибутів.

Вище наведені знімки екрану (рис. 14 та рис. 15), які виконують пошук в одній або у двох таблицях, з заданою кількістю атрибутів.

Наприклад, нам потрібно у таблиці “Телефонні номери” знайти номери певного постачальника, то слід буде вказати що пошук відбувається за атрибутом provider\_id, а якщо слід знайти номер постачальника, але вказати мобільного оператора, то для цього слід задати два атрибути пошуку, а саме provider\_id та mobile\_operator.

**Лістинги фрагментів програм генерування випадкових даних в таблицях баз даних**

****Рисунок 15 - Функція, яка викликає іншу функцію, що в свою чергу виконує генерування випадкових даних.

Функція, яка зображена на рис. 14 обробляє введені дані користувачем, та в залежності від значення параметру table\_name викликає конкретну функцію, яка в свою чергу генерує n випадкових значень для даної таблиці.

Результати роботи даних функцій наведені у розділі “Результати програми”, які знаходяться нижче.

**Лістинг модуля “Model”**

import datetime

import psycopg2 as ps

class Model:

def \_\_init\_\_(self):

self.conn = None

try:

self.conn = ps.connect(

database="Pharmacy",

user="postgres",

password="1234",

host="127.0.0.1",

port="5432",

)

except(Exception, ps.DatabaseError) as error:

print("[INFO] Error when working with PostgreSQL", error)

def request(self, req: str):

try:

cursor = self.conn.cursor()

print(req)

cursor.execute(req)

self.conn.commit()

return True

except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:

print(error)

self.conn.rollback()

return False

def get(self, req: str):

try:

cursor = self.conn.cursor()

print(req)

cursor.execute(req)

self.conn.commit()

return cursor.fetchall()

except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:

print(error)

self.conn.rollback()

return False

def get\_el(self, req: str):

try:

cursor = self.conn.cursor()

print(req)

cursor.execute(req)

self.conn.commit()

return cursor.fetchone()

except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:

print(error)

self.conn.rollback()

return False

def count(self, table\_name: str):

return self.get\_el(f"select count(\*) from public.\"{table\_name}\"")

def find(self, table\_name: str, key\_name: str, key\_value: int):

return self.get\_el(f"select count(\*) from public.\"{table\_name}\" where {key\_name}={key\_value}")

def min(self, table\_name: str, key\_name: str):

return self.get\_el(f"select min({key\_name}) from public.\"{table\_name}\"")

def max(self, table\_name: str, key\_name: str):

return self.get\_el(f"select max({key\_name}) from public.\"{table\_name}\"")

def print\_provider(self) -> None:

return self.get(f"SELECT \* FROM public.\"Provider\"")

def print\_phone\_numbers(self) -> None:

return self.get(f"SELECT \* FROM public.\"Phone numbers\"")

def print\_supplier\_order\_base(self) -> None:

return self.get(f"SELECT \* FROM public.\"Supplier order base\"")

def print\_pharmacy(self) -> None:

return self.get(f"SELECT \* FROM public.\"Pharmacy\"")

def print\_medicine(self) -> None:

return self.get(f"SELECT \* FROM public.\"Medicine\"")

def delete\_data(self, table\_name: str, key\_name: str, key\_value: int) -> None:

self.request(f"DELETE FROM public.\"{table\_name}\" WHERE {key\_name}={key\_value};")

def update\_provider\_data(self, provider\_id: int, title: str, address: str) -> None:

self.request(f"UPDATE public.\"Provider\" SET title=\'{title}\', address=\'{address}\' "

f"WHERE provider\_id={provider\_id};")

def update\_phone\_numbers\_data(self, phone\_number: int, provider\_id: int, mobile\_operator: str) -> None:

self.request(f"UPDATE public.\"Phone numbers\" SET provider\_id={provider\_id}, "

f"mobile\_operator=\'{mobile\_operator}\' WHERE phone\_number={phone\_number};")

def update\_supplier\_order\_base\_data(self, key\_value: int, medicine\_id: int, provider\_id: int, pharmacy\_id: int,

number: int, order\_date: datetime.datetime) -> None:

self.request(f"UPDATE public.\"Supplier order base\" SET provider\_id={provider\_id},"

f"pharmacy\_id={pharmacy\_id}, number={number}, order\_date=\'{order\_date}\',"

f"WHERE supplier\_order\_id={key\_value}, medicine\_id={medicine\_id};")

def update\_pharmacy\_data(self, pharmacy\_id: int, title: str, phone\_num: int, address: str) -> None:

self.request(f"UPDATE public.\"Pharmacy\" SET pharmacy\_title=\'{title}\', phone\_num={phone\_num},"

f"address=\'{address}\' WHERE pharmacy\_id={pharmacy\_id};")

def update\_medicine\_data(self, medicine\_id: int, title: str, manufacturer: str) -> None:

self.request(f"UPDATE public.\"Medicine\" SET medicine\_title=\'{title}\', manufacturer=\'{manufacturer}\'"

f"WHERE medicine\_id={medicine\_id};")

def insert\_provider\_data(self, provider\_id: int, title: str, address: str) -> None:

self.request(f"insert into public.\"Provider\" (provider\_id, title, address) "

f"VALUES ({provider\_id}, \'{title}\', {address});")

def insert\_phone\_numbers\_data(self, phone\_number: int, provider\_id: int, mobile\_operator: str) -> None:

self.request(f"insert into public.\"Phone numbers\" (phone\_num, provider\_id, mobile\_operator) "

f"VALUES ({phone\_number}, {provider\_id}, \'{mobile\_operator}\');")

def insert\_supplier\_order\_base\_data(self, key\_value: int, medicine\_id: int, provider\_id: int, pharmacy\_id: int,

number: int, order\_date: datetime.datetime) -> None:

self.request(f"insert into public.\"Supplier order base\" (supplier\_order\_id, provider\_id, pharmacy\_id, "

f"medicine\_id, number, order\_date) VALUES ({key\_value}, {provider\_id}, {pharmacy\_id}, "

f"{medicine\_id}, {number}, \'{order\_date}\');")

def insert\_pharmacy\_data(self, pharmacy\_id: int, title: str, phone\_num: int, address: str) -> None:

self.request(f"insert into public.\"Pharmacy\" (pharmacy\_id, pharmacy\_title, phone\_num, address) "

f"VALUES ({pharmacy\_id}, \'{title}\', {phone\_num}, \'{address}\');")

def insert\_medicine\_data(self, medicine\_id: int, title: str, manufacturer: str) -> None:

self.request(f"insert into public.\"Medicine\" (medicine\_id, medicine\_title, manufacturer) "

f"VALUES ({medicine\_id}, \'{title}\', \'{manufacturer}\');")

def provider\_data\_generator(self, value: int) -> None:

for a in range(value):

self.request("insert into public.\"Provider\" select (SELECT MAX(provider\_id)+1 FROM public.\"Provider\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''); ")

def medicine\_data\_generator(self, value: int) -> None:

for a in range(value):

self.request("insert into public.\"Medicine\" select (SELECT MAX(medicine\_id)+1 FROM public.\"Medicine\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''); ")

def pharmacy\_data\_generator(self, value: int) -> None:

for a in range(value):

self.request("insert into public.\"Pharmacy\" select (SELECT MAX(pharmacy\_id)+1 FROM public.\"Pharmacy\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(12-4)+4):: integer)), ''), "

"FLOOR(RANDOM()\*(999999990-1)+1), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''); ")

def phone\_numbers\_data\_generator(self, value: int) -> None:

for a in range(value):

self.request("(SELECT provider\_id FROM public.\"Provider\" LIMIT 1 OFFSET (round(random() \* "

"((SELECT COUNT(provider\_id) FROM public.\"Provider\")-1)))), "

"insert into public.\"Phone numbers\" "

"select (SELECT MAX(phone\_num)+1 FROM public.\"Phone numbers\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''); ")

def supplier\_order\_base\_data\_generator(self, value: int) -> None:

for a in range(value):

self.request("insert into public.\"Supplier order base\" "

"select (SELECT MAX(supplier\_order\_id)+1 FROM "

"public.\"Supplier order base\"), "

"(SELECT provider\_id FROM public.\"Provider\" LIMIT 1 OFFSET (round(random() \* "

"((SELECT COUNT(provider\_id) FROM public.\"Provider\")-1)))), "

"(SELECT pharmacy\_id FROM public.\"Pharmacy\" LIMIT 1 OFFSET (round(random() \* "

"((SELECT COUNT(pharmacy\_id) FROM public.\"Pharmacy\")-1)))), "

"(SELECT medicine\_id FROM public.\"Medicine\" LIMIT 1 OFFSET (round(random() \* "

"((SELECT COUNT(medicine\_id) FROM public.\"Medicine\")-1)))), "

"FLOOR(RANDOM()\*(1000-1)+1), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def search\_data\_one\_table(self, table1\_name: str, search: str):

return self.get(f"select \* from public.\"{table1\_name}\" "

f"where {search}")

def search\_data\_two\_tables(self, table1\_name: str, table2\_name: str, table1\_key, table2\_key, search: str):

table2\_name = "Phone numbers"

return self.get(f"select \* from public.\"{table1\_name}\" as one inner join public.\"{table2\_name}\" as two "

f"on one.\"{table1\_key}\"=two.\"{table2\_key}\" "

f"where {search}")

Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити.

Конструктор класу Model налагоджує зв’язок із сервером для подальшої роботи, коли зв’язок не буде встановлено, то буде видаватися помилка.

Методи request, get, get\_el виконують запити до БД за допомогою cursor вони повертають False, якщо буде помилка і запит не спрацював, якщо ж все пройшло добре, то метод request повертає значення True, get повертає дані, які було взято із запиту SELECT, get\_el повертає лише перший запис.

Метод count повертає кількість записів у таблиці.

Метод find повертає кількість записів, які відповідають заданим умовам, якщо такий запис існує у таблиці.

Метод min, max повертають відповідно мінімальне та максимальне значення вказаного ключа.

Методи print\_provider, print\_phone\_numbers, print\_supplier\_order\_base, print\_pharmacy, print\_medicine повертають відповідні таблиці, які були отримані з бази даних.

Методи update\_provider\_data, update\_phone\_numbers\_data, update\_supplier\_order\_base\_data, update\_pharmacy\_data, update\_medicine\_data відправляють запит до бази даних на заміну даних у певних полях запису.

Методи insert\_provider\_data, insert\_phone\_numbers\_data, insert\_supplier\_order\_base\_data, insert\_pharmacy\_data, insert\_medicine\_data відправляють запит до бази даних на вставку нового запису у таблицю.

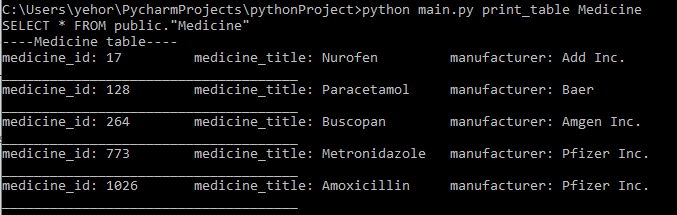
Метод delete\_data виконує запит на видалення запису з певної таблиці.

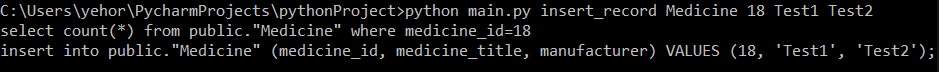
Методи provider\_data\_generator, phone\_numbers\_data\_generator, supplier\_order\_base\_data\_generator, pharmacy\_data\_generator, medicine\_data\_generator виконують запити на вставку псевдорандомізованих даних.

Методи search\_data\_one\_table, search\_data\_two\_tables виконують запит на пошук певних значень атрибутів у вказаних таблицях.

## **Результати роботи**

**Результати виконання директиви внесення**

  
Рисунок 16 - Таблиця “Препарати” до виконання директиви внесення.

Рисунок 17 - Запит на виконання директиви внесення у таблицю “Препарати”.

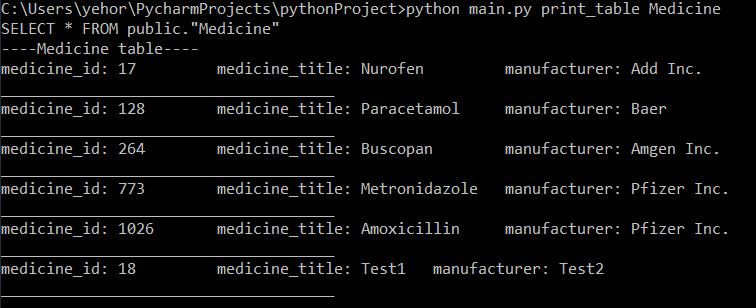
  
Рисунок 18 - Таблиця “Препарати” після виконання директиви внесення.

  
Рисунок 19 - Виведення помилки при повторному внесенні даних з однаковим первинним ключем.

**Результати виконання директиви заміни**

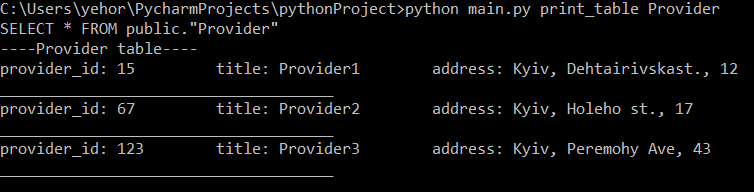
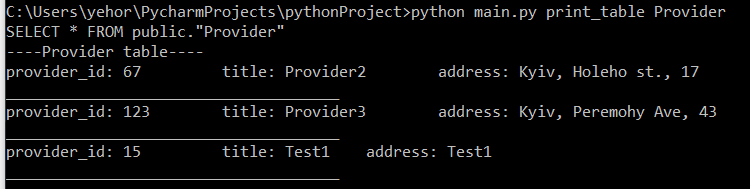
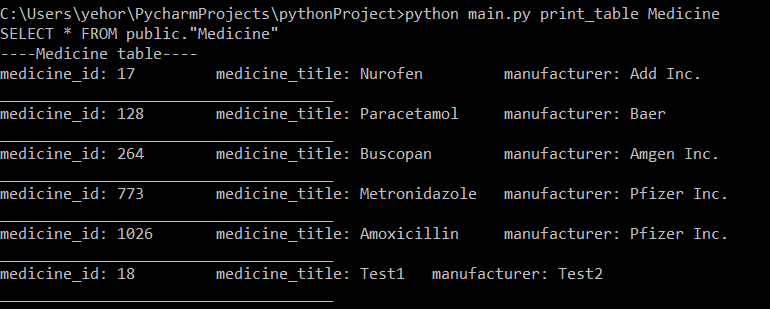
****Рисунок 20- Таблиця “Постачальник” до виконання директиви заміни.

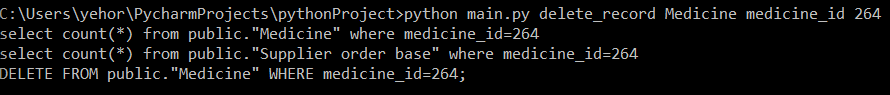


Рисунок 21 - Запит на виконання директиви заміни у таблиці “Постачальник”.

  
Рисунок 22 - Таблиця “Постачальник” після виконання директиви заміни.

**Результати виконання директиви вилучення**

****Рисунок 23 - Таблиця “Препарат” до виконання директиви вилучення.

  
Рисунок 24 - Запит на вилучення рядка за конкретним ключем у таблиці “Препарат”.

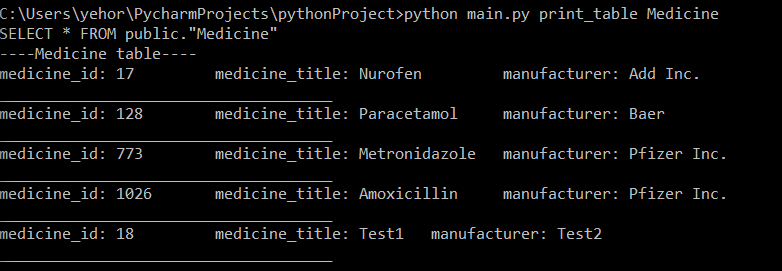


Рисунок 25 - Таблиця “Препарат” після виконання директиви вилучення.

**Результати виконання директиви пошуку даних**

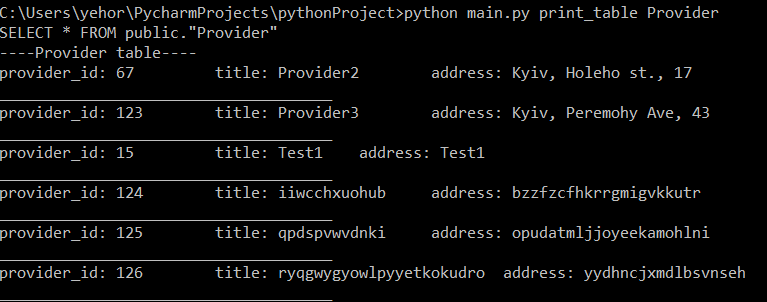
****

Рисунок 26 - Таблиця “Постачальник” для наочної перевірки роботи директиви пошуку в одній таблиці за одним атрибутом.

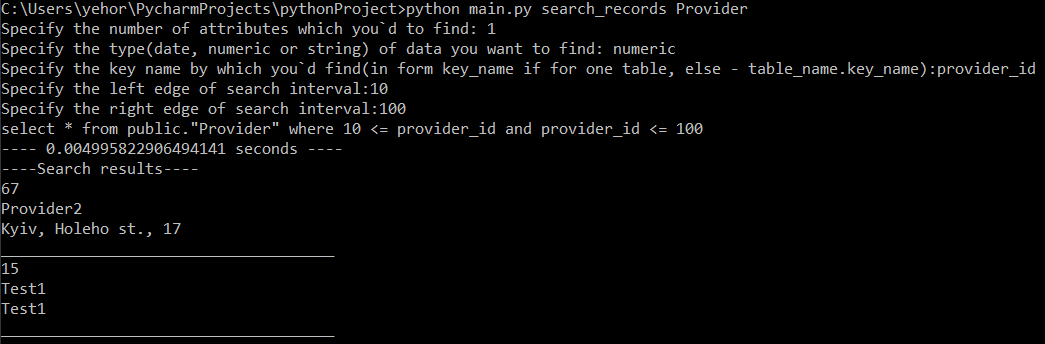


Рисунок 27 - Запит та результат роботи директиви пошуку в таблиці “Постачальник” за одним атрибутом.

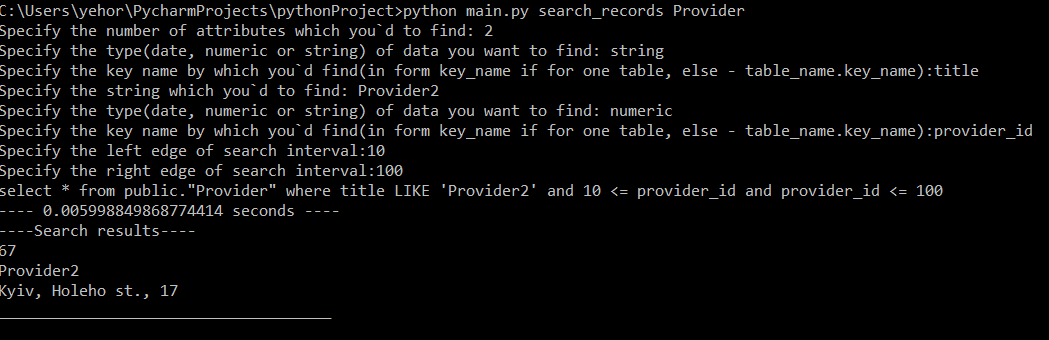


Рисунок 27 - Запит та результат роботи директиви пошуку в таблиці “Постачальник” за кількома атрибутами.

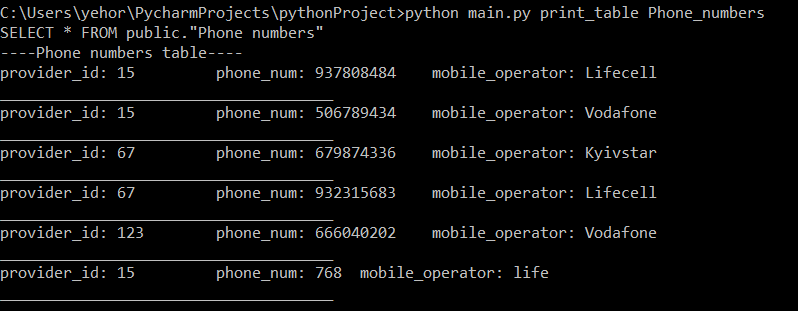


Рисунок 26 - Таблиця “Телефонні номери” для наочної перевірки роботи директиви пошуку в двох таблицях за одним атрибутом.

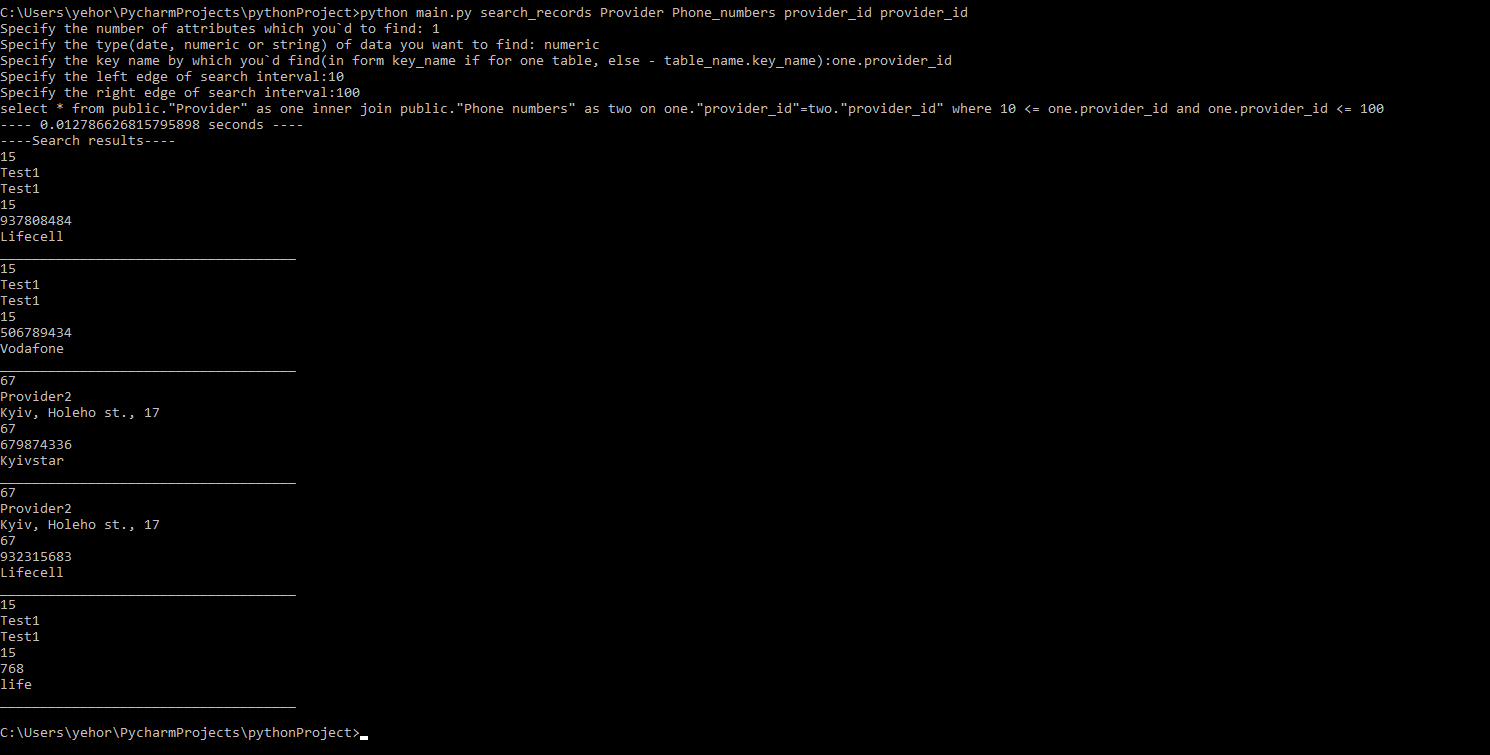


Рисунок 27 - Запит та результат роботи директиви пошуку в таблицях “Постачальник” та “Телефонні номери” за одним атрибутом.

**Результати виконання директиви генерування даних**

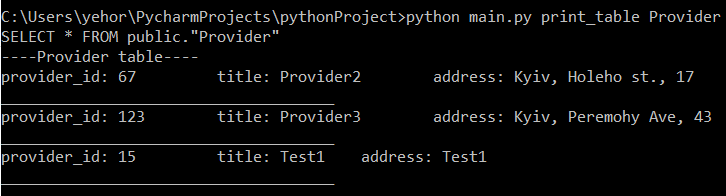
****

Рисунок 28 - Таблиця “Постачальник” до виконання директиви генерування даних.

Текст з командного рядка для виконання команди генерування трьох записів у таблиці “Постачальник”. (Вставляю текст, оскільки знімок екрану не є читабельним).

C:\Users\yehor\PycharmProjects\pythonProject>python main.py generate\_records Provider 3

insert into public."Provider" select (SELECT MAX(provider\_id)+1 FROM public."Provider"), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), '');

insert into public."Provider" select (SELECT MAX(provider\_id)+1 FROM public."Provider"), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), '');

insert into public."Provider" select (SELECT MAX(provider\_id)+1 FROM public."Provider"), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), '');

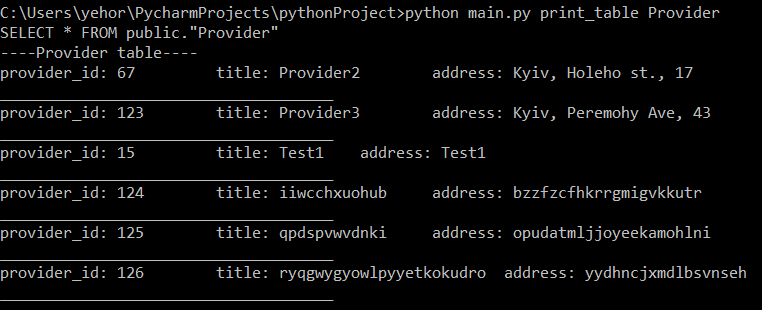


Рисунок 29 - Таблиця “Постачальник” після виконання директиви генерування даних.

**Скріншоти результатів виконання операції вилучення запису батьківської таблиці та виведення вмісту дочірньої.**

Дана маніпуляція, а саме видалення запису з батьківської таблиці, який зв’язаний з дочірньою, буде неможливим та буде видана помилка.

Для перевірки візьмемо, такі таблиці, а саме дочірньою буде таблиця “Замовлення для постачальника”, а батьківською - “Постачальник”.

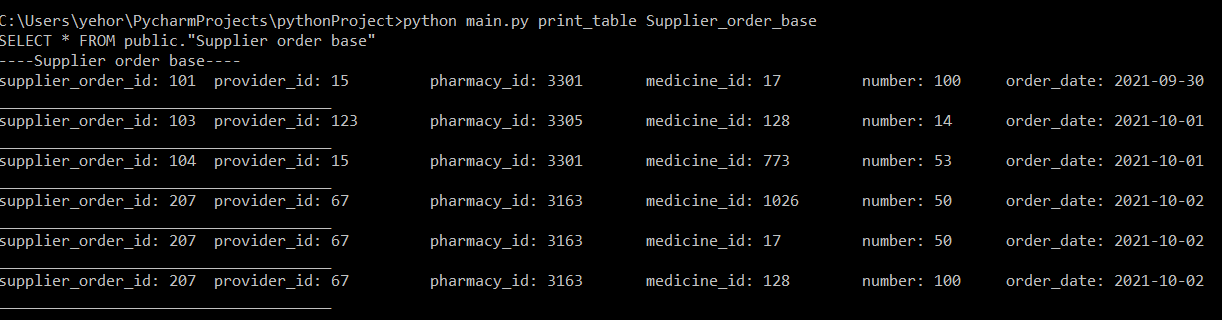
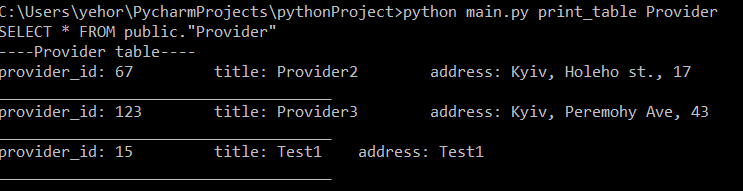
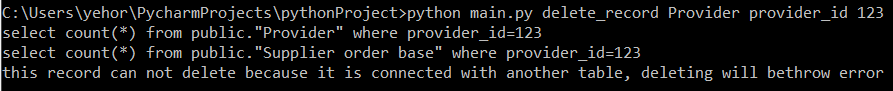


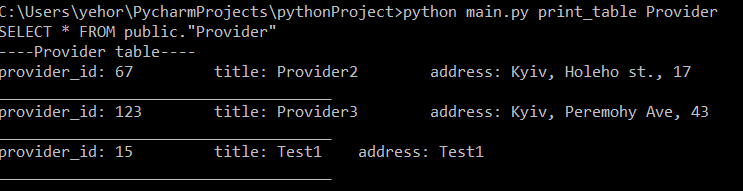
Рисунок 30 - Скріншот дочірньої таблиці “Замовлення для постачальника”.

   
Рисунок 31 - Скріншот батьківської таблиці “Постачальник”

  
Рисунок 32 - Спроба видалення запису за батьківської таблиці “Постачальник”.

**Скріншоти результатів виконання операції вставки у дочірню таблицю та виведення повідомлення про її неможливість, якщо у батьківській таблиці немає відповідного запису .**

Для перевірки візьмемо, такі таблиці, а саме дочірньою буде таблиця “Телефонні номери”, а батьківською - “Постачальник”.

   
Рисунок 33 - Скріншот батьківської таблиці “Постачальник”

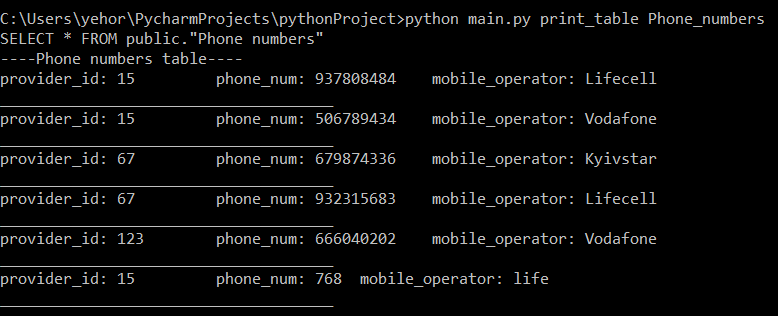
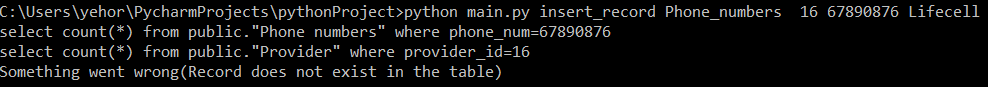


Рисунок 35 - Скріншот дочірньої таблиці “Телефонні номери”.

  
Рисунок \* - Спроба створити запис у дочірній таблиці з неіснуючим записом у батьківській таблиці.

**Текст програми**

*main.py*

**import** controller **as** con

**from** psycopg2 **import** Error

**import** sys

c = con.Controller()

**try**:

command = sys.argv[1]

**except** IndexError:

c.v.no\_command()

**else**:

**if** command == **'print\_table'**:

**try**:

name = sys.argv[2]

**except** IndexError:

c.v.argument\_error()

**else**:

c.print(name)

**elif** command == **'delete\_record'**:

**try**:

args = {**"table\_name"**: sys.argv[2], **"key"**: sys.argv[3], **"value"**: sys.argv[4]}

**except** IndexError:

c.v.arg\_error()

**else**:

c.delete(args[**"table\_name"**], args[**"key"**], args[**"value"**])

**elif** command == **'insert\_record'**:

**try**:

args = {**"table\_name"**: sys.argv[2], **"key"**: sys.argv[3]}

**if** args[**"table\_name"**] == **'Medicine'**:

args[**"title"**], args[**"manufacturer"**] = sys.argv[4], sys.argv[5]

**elif** args[**"table\_name"**] == **'Provider'**:

args[**"title"**], args[**"address"**] = sys.argv[4], sys.argv[5]

**elif** args[**"table\_name"**] == **'Phone\_numbers'**:

args[**"provider\_id"**], args[**"mobile\_operator"**] = sys.argv[4], sys.argv[5]

**elif** args[**"table\_name"**] == **'Pharmacy'**:

args[**"title"**], args[**"phone\_num"**], args[**"address"**] = sys.argv[4], sys.argv[5], sys.argv[6]

**elif** args[**"table\_name"**] == **'Supplier\_order\_base'**:

args[**"provider\_id"**], args[**"pharmacy\_id"**], args[**"medicine\_id"**], args[**"number"**], args[**"order\_date"**] =\

sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7], sys.argv[8]

**except** IndexError:

c.v.wrong\_table()

**else**:

**if** args[**"table\_name"**] == **'Medicine'**:

c.insert\_medicine(args[**"key"**], args[**"title"**], args[**"manufacturer"**])

**elif** args[**"table\_name"**] == **'Provider'**:

c.insert\_provider(args[**"key"**], args[**"title"**], args[**"address"**])

**elif** args[**"table\_name"**] == **'Pharmacy'**:

c.insert\_pharmacy(args[**"key"**], args[**"title"**], args[**"phone\_num"**], args[**"address"**])

**elif** args[**"table\_name"**] == **'Phone\_numbers'**:

c.insert\_phone\_numbers(args[**"key"**], args[**"provider\_id"**], args[**"mobile\_operator"**])

**elif** args[**"table\_name"**] == **'Supplier\_order\_base'**:

c.insert\_supplier\_order\_base(args[**"key"**], args[**"provider\_id"**], args[**"pharmacy\_id"**], args[**"medicine\_id"**],

args[**"number"**], args[**"order\_date"**])

**elif** command == **'update\_record'**:

**try**:

args = {**"table\_name"**: sys.argv[2], **"key"**: sys.argv[3]}

**if** args[**"table\_name"**] == **'Medicine'**:

args[**"title"**], args[**"manufacturer"**] = sys.argv[4], sys.argv[5]

**elif** args[**"table\_name"**] == **'Provider'**:

args[**"title"**], args[**"address"**] = sys.argv[4], sys.argv[5]

**elif** args[**"table\_name"**] == **'Phone\_numbers'**:

args[**"provider\_id"**], args[**"mobile\_operator"**] = sys.argv[4], sys.argv[5]

**elif** args[**"table\_name"**] == **'Pharmacy'**:

args[**"title"**], args[**"phone\_num"**], args[**"address"**] = sys.argv[4], sys.argv[5], sys.argv[6]

**elif** args[**"table\_name"**] == **'Supplier\_order\_base'**:

args[**"provider\_id"**], args[**"pharmacy\_id"**], args[**"medicine\_id"**], args[**"number"**], args[**"order\_date"**] = \

sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7], sys.argv[8]

**except** IndexError:

c.v.wrong\_table()

**else**:

**if** args[**"table\_name"**] == **'Medicine'**:

c.update\_medicine\_table(args[**"key"**], args[**"title"**], args[**"manufacturer"**])

**elif** args[**"table\_name"**] == **'Provider'**:

c.update\_provider\_table(args[**"key"**], args[**"title"**], args[**"address"**])

**elif** args[**"table\_name"**] == **'Pharmacy'**:

c.update\_pharmacy\_table(args[**"key"**], args[**"title"**], args[**"phone\_num"**], args[**"address"**])

**elif** args[**"table\_name"**] == **'Phone\_numbers'**:

c.update\_phone\_numbers\_table(args[**"key"**], args[**"provider\_id"**], args[**"mobile\_operator"**])

**elif** args[**"table\_name"**] == **'Supplier\_order\_base'**:

c.update\_supplier\_order\_base\_table(args[**"key"**], args[**"provider\_id"**], args[**"pharmacy\_id"**],

args[**"medicine\_id"**], args[**"number"**], args[**"order\_date"**])

**elif** command == **'generate\_records'**:

**try**:

args = {**"table\_name"**: sys.argv[2], **"value"**: int(sys.argv[3])}

**except** (Exception, IndexError):

print(Exception, IndexError)

**else**:

c.generate(args[**"table\_name"**], args[**"value"**])

**elif** command == **"search\_records"**:

**if** len(sys.argv) == 3:

search\_num = c.v.get\_search\_num()

**try**:

search\_num = int(search\_num)

**except** ValueError:

c.v.invalid\_search\_num()

**else**:

**if** search\_num > 0:

args = {**"table1\_name"**: sys.argv[2]}

c.search\_in\_one\_table(args[**"table1\_name"**], c.v.initiate\_search(search\_num))

**else**:

c.v.invalid\_search\_num()

**elif** len(sys.argv) == 6:

search\_num = c.v.get\_search\_num()

**try**:

search\_num = int(search\_num)

**except** ValueError:

c.v.invalid\_search\_num()

**else**:

**if** search\_num > 0:

args = {**"table1\_name"**: sys.argv[2], **"table2\_name"**: sys.argv[3],

**"key1\_name"**: sys.argv[4], **"key2\_name"**: sys.argv[5]}

c.search\_in\_two\_tables(args[**"table1\_name"**], args[**"table2\_name"**], args[**"key1\_name"**],

args[**"key2\_name"**], c.v.initiate\_search(search\_num))

**else**:

c.v.invalid\_search\_num()

**else**:

c.v.argument\_error()

**elif** command == **'help'**:

c.v.print\_help()

**else**:

c.v.wrong\_command()

*controller.py*

**import** psycopg2

**from** psycopg2 **import** Error

**import** model

**import** view

**import** datetime

**import** time

**class** Controller:

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

self.v = view.View()

self.m = model.Model()

**def** print(self, table\_name):

name = self.v.valid.check\_table\_name(table\_name)

**if** name:

**if** name == **'Pharmacy'**:

self.v.print\_pharmacy(self.m.print\_pharmacy())

**elif** name == **'Medicine'**:

self.v.print\_medicine(self.m.print\_medicine())

**elif** name == **'Provider'**:

self.v.print\_provider(self.m.print\_provider())

**elif** name == **"Phone\_numbers"**:

self.v.print\_phone\_numbers(self.m.print\_phone\_numbers())

**elif** name == **'Supplier\_order\_base'**:

self.v.print\_supplier\_order\_base(self.m.print\_supplier\_order\_base())

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

table\_name\_ = self.v.valid.check\_table\_name(table\_name)

key\_name\_ = self.v.valid.check\_key\_names(table\_name\_, key\_name)

**if** table\_name\_ **and** key\_name\_:

count = self.m.find(table\_name\_, key\_name, value)

key\_value = self.v.valid.check\_pkey(value, count)

**if** key\_value:

**if** table\_name\_ == **'Provider' or** table\_name\_ == **'Pharmacy' or** table\_name\_ == **'Medicine'**:

count\_sob = self.m.find(**'Supplier order base'**, key\_name, value)[0]

**if** count\_sob:

self.v.not\_delete()

**else**:

**try**:

self.m.delete\_data(table\_name, key\_name, key\_value)

**except**(Exception, Error) **as** ex:

self.v.sql\_error(ex)

**elif** table\_name\_ == **'Provider'**:

count\_p = self.m.find(**'Phone numbers'**, key\_name, key\_value)[0]

**if** count\_p:

self.v.not\_delete()

**else**:

**try**:

self.m.delete\_data(table\_name, key\_name, key\_value)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

**try**:

self.m.delete\_data(table\_name, key\_name, key\_value)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.deletion\_error()

**def** update\_pharmacy\_table(self, pharmacy\_id: str, pharmacy\_title: str, phone\_num: str, address: str):

id\_value = **None**

**if** self.v.valid.check\_keys(**'Pharmacy'**, **'pharmacy\_id'**, pharmacy\_id):

count\_id = self.m.find(**'Pharmacy'**, **'pharmacy\_id'**, int(pharmacy\_id))

id\_value = self.v.valid.check\_pkey(pharmacy\_id, count\_id)

**if** id\_value **and** self.v.valid.check\_keys(**'Pharmacy'**, **'pharmacy\_title'**, pharmacy\_title) \

**and** self.v.valid.check\_keys(**'Pharmacy'**, **'phone\_num'**, phone\_num) \

**and** self.v.valid.check\_keys(**'Pharmacy'**, **'address'**, address):

**try**:

self.m.update\_pharmacy\_data(id\_value, pharmacy\_title, int(phone\_num), address)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.updation\_error()

**def** update\_provider\_table(self, provider\_id: str, title: str, address: str):

id\_value = **None**

**if** self.v.valid.check\_keys(**'Provider'**, **'provider\_id'**, provider\_id):

count\_id = self.m.find(**'Provider'**, **'provider\_id'**, int(provider\_id))

id\_value = self.v.valid.check\_pkey(provider\_id, count\_id)

**if** id\_value **and** self.v.valid.check\_keys(**'Provider'**, **'title'**, title) \

**and** self.v.valid.check\_keys(**'Provider'**, **'address'**, address):

**try**:

self.m.update\_provider\_data(id\_value, title, address)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.updation\_error()

**def** update\_medicine\_table(self, medicine\_id: str, medicine\_title: str, manufacturer: str):

id\_value = **None**

**if** self.v.valid.check\_keys(**'Medicine'**, **'medicine\_id'**, medicine\_id):

count\_id = self.m.find(**'Medicine'**, **'medicine\_id'**, int(medicine\_id))

id\_value = self.v.valid.check\_pkey(medicine\_id, count\_id)

**if** id\_value **and** self.v.valid.check\_keys(**'Medicine'**, **'medicine\_title'**, medicine\_title) \

**and** self.v.valid.check\_keys(**'Medicine'**, **'manufacturer'**, manufacturer):

**try**:

self.m.update\_medicine\_data(id\_value, medicine\_id, manufacturer)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.updation\_error()

**def** update\_phone\_numbers\_table(self, provider\_id: str, phone\_num: str, mobile\_operator: str):

id\_value = **None**

num\_value = **None**

**if** self.v.valid.check\_keys(**'Phone numbers'**, **'provider\_id'**, provider\_id):

count\_id = self.m.find(**'Phone numbers'**, **'provider\_id'**, int(provider\_id))

id\_value = self.v.valid.check\_pkey(provider\_id, count\_id)

**if** self.v.valid.check\_keys(**'Phone numbers'**, **'phone\_num'**, phone\_num):

count\_num = self.m.find(**'Phone numbers'**, **'phone\_num'**, int(phone\_num))

num\_value = self.v.valid.check\_pkey(phone\_num, count\_num)

**if** id\_value **and** num\_value **and** self.v.valid.check\_keys(**'Phone numbers'**, **'mobile\_operator'**, mobile\_operator):

**try**:

self.m.update\_phone\_numbers\_data(num\_value, id\_value, mobile\_operator)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.updation\_error()

**def** update\_supplier\_order\_base\_table(self, supplier\_order\_id: str, provider\_id: str, pharmacy\_id: str,

medicine\_id: str, number: str, order\_date: str):

id\_value = **None**

provider\_id\_value = **None**

pharmacy\_id\_value = **None**

medicine\_id\_value = **None**

**if** self.v.valid.check\_keys(**'Supplier order base'**, **'supplier\_order\_id'**, supplier\_order\_id):

count\_id = self.m.find(**'Supplier order base'**, **'supplier\_order\_id'**, int(supplier\_order\_id))

id\_value = self.v.valid.check\_pkey(supplier\_order\_id, count\_id)

**if** self.v.valid.check\_keys(**'Supplier order base'**, **'provider\_id'**, provider\_id):

count\_provider\_id = self.m.find(**'Supplier order base'**, **'provider\_id'**, int(provider\_id))

provider\_id\_value = self.v.valid.check\_pkey(provider\_id, count\_provider\_id)

**if** self.v.valid.check\_keys(**'Supplier order base'**, **'pharmacy\_id'**, pharmacy\_id):

count\_pharmacy\_id = self.m.find(**'Supplier order base'**, **'pharmacy\_id'**, int(pharmacy\_id))

pharmacy\_id\_value = self.v.valid.check\_pkey(pharmacy\_id, count\_pharmacy\_id)

**if** self.v.valid.check\_keys(**'Supplier order base'**, **'medicine\_id'**, medicine\_id):

count\_medicine\_id = self.m.find(**'Supplier order base'**, **'medicine\_id'**, int(medicine\_id))

medicine\_id\_value = self.v.valid.check\_pkey(medicine\_id, count\_medicine\_id)

**if** id\_value **and** provider\_id\_value **and** pharmacy\_id\_value **and** medicine\_id\_value \

**and** self.v.valid.check\_keys(**'Supplier order base'**, **'number'**, number) \

**and** self.v.valid.check\_keys(**'Supplier order base'**, **'order\_date'**, order\_date):

**try**:

date\_info = [int(a) **for** a **in** order\_date.split(sep=**'.'**)]

self.m.update\_supplier\_order\_base\_data(id\_value, provider\_id\_value, pharmacy\_id\_value,

medicine\_id\_value, int(number),

datetime.datetime(date\_info[0], date\_info[1], date\_info[2],

date\_info[3], date\_info[4], date\_info[5]))

**except**(Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.updation\_error()

**def** insert\_pharmacy(self, pharmacy\_id: str, pharmacy\_title: str, phone\_num: str, address: str):

count\_id = **None**

**if** self.v.valid.check\_keys(**'Pharmacy'**, **'pharmacy\_id'**, pharmacy\_id):

count\_id = self.m.find(**'Pharmacy'**, **'pharmacy\_id'**, int(pharmacy\_id))[0]

**if**(**not** count\_id **or** count\_id == (0,)) **and** self.v.valid.check\_keys(**'Pharmacy'**, **'pharmacy\_id'**, pharmacy\_id):

**try**:

self.m.insert\_pharmacy\_data(int(pharmacy\_id), pharmacy\_title, int(phone\_num), address)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.insertion\_error()

**def** insert\_provider(self, provider\_id: str, title: str, address: str):

count\_id = **None**

**if** self.v.valid.check\_keys(**'Provider'**, **'provider\_id'**, provider\_id):

count\_id = self.m.find(**'Provider'**, **'provider\_id'**, int(provider\_id))[0]

**if** (**not** count\_id **or** count\_id == (0,)) **and** self.v.valid.check\_keys(**'Provider'**, **'provider\_id'**, provider\_id):

**try**:

self.m.insert\_provider\_data(int(provider\_id), title, address)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.insertion\_error()

**def** insert\_medicine(self, medicine\_id: str, medicine\_title: str, manufacturer: str):

count\_id = **None**

**if** self.v.valid.check\_keys(**'Medicine'**, **'medicine\_id'**, medicine\_id):

count\_id = self.m.find(**'Medicine'**, **'medicine\_id'**, int(medicine\_id))[0]

**if** (**not** count\_id **or** count\_id == (0,)) **and** self.v.valid.check\_keys(**'Medicine'**, **'medicine\_id'**, medicine\_id):

**try**:

self.m.insert\_medicine\_data(int(medicine\_id), medicine\_title, manufacturer)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.insertion\_error()

**def** insert\_phone\_numbers(self, provider\_id: str, phone\_num: str, mobile\_operator: str):

provider\_id\_value = **None**

count\_num = **None**

**if** self.v.valid.check\_keys(**'Phone numbers'**, **'phone\_num'**, phone\_num):

count\_num = self.m.find(**'Phone numbers'**, **'phone\_num'**, int(phone\_num))[0]

**if** self.v.valid.check\_keys(**'Provider'**, **'provider\_id'**, provider\_id):

count\_id = self.m.find(**'Provider'**, **'provider\_id'**, int(provider\_id))

provider\_id\_value = self.v.valid.check\_pkey(provider\_id, count\_id)

**if**(**not** count\_num **or** count\_num == (0,)) **and** provider\_id\_value \

**and** self.v.valid.check\_keys(**'Phone numbers'**, **'phone\_num'**, phone\_num):

**try**:

self.m.insert\_phone\_numbers\_data(int(phone\_num), int(provider\_id), mobile\_operator)

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.insertion\_error()

**def** insert\_supplier\_order\_base(self, supplier\_order\_id: str, provider\_id: str, pharmacy\_id: str, medicine\_id: str,

number: str, order\_date: str):

count\_sob = **None**

count\_med = **None**

provider\_id\_value = **None**

pharmacy\_id\_value = **None**

**if** self.v.valid.check\_keys(**'Supplier order base'**, **'supplier\_order\_id'**, supplier\_order\_id):

count\_sob = self.m.find(**'Supplier order base'**, **'supplier\_order\_id'**, int(supplier\_order\_id))[0]

**if** self.v.valid.check\_keys(**'Supplier order base'**, **'medicine\_id'**, medicine\_id):

count\_med = self.m.find(**'Supplier order base'**, **'medicine\_id'**, int(medicine\_id))[0]

**if** self.v.valid.check\_keys(**'Provider'**, **'provider\_id'**, provider\_id):

count\_provider\_id = self.m.find(**'Provider'**, **'provider\_id'**, int(provider\_id))

provider\_id\_value = self.v.valid.check\_pkey(provider\_id, count\_provider\_id)

**if** self.v.valid.check\_keys(**'Pharmacy'**, **'pharmacy\_id'**, pharmacy\_id):

count\_pharmacy\_id = self.m.find(**'Pharmacy'**, **'pharmacy\_id'**, int(pharmacy\_id))

pharmacy\_id\_value = self.v.valid.check\_pkey(pharmacy\_id, count\_pharmacy\_id)

**if** (**not** count\_sob **or** count\_sob == (0,)) **and** (**not** count\_med **or** count\_med == (0,)) \

**and** provider\_id\_value **and** pharmacy\_id\_value \

**and** self.v.valid.check\_keys(**'Supplier order base'**, **'supplier\_order\_id'**, supplier\_order\_id) \

**and** self.v.valid.check\_keys(**'Supplier order base'**, **'number'**, number) \

**and** self.v.valid.check\_keys(**'Supplier order base'**, **'order\_date'**, order\_date):

**try**:

arr = [int(a) **for** a **in** order\_date.split(sep=**'.'**)]

self.m.insert\_supplier\_order\_base\_data(int(supplier\_order\_id), int(medicine\_id), int(provider\_id),

int(pharmacy\_id), int(number),

datetime.datetime(arr[0], arr[1], arr[2],

arr[3], arr[4], arr[5]))

**except** (Exception, Error) **as** ex:

self.v.sql\_error(ex)

**else**:

self.v.insertion\_error()

**def** generate(self, table\_name: str, n: int):

t\_name = self.v.valid.check\_table\_name(table\_name)

**if** t\_name:

**if** t\_name == **'Pharmacy'**:

self.m.pharmacy\_data\_generator(n)

**elif** t\_name == **'Provider'**:

self.m.provider\_data\_generator(n)

**elif** t\_name == **'Medicine'**:

self.m.medicine\_data\_generator(n)

**elif** t\_name == **'Phone numbers'**:

self.m.phone\_numbers\_data\_generator(n)

**elif** t\_name == **'Supplier order base'**:

self.m.supplier\_order\_base\_data\_generator(n)

**def** search\_in\_one\_table(self, table1\_name: str, search: str):

t1\_n = self.v.valid.check\_table\_name(table1\_name)

**if** t1\_n:

start\_time = time.time()

result = self.m.search\_data\_one\_table(table1\_name, search)

self.v.time\_output(start\_time)

self.v.print\_search(result)

**def** search\_in\_two\_tables(self, table1\_name: str, table2\_name: str, table1\_key: str, table2\_key: str, search: str):

t1\_n = self.v.valid.check\_table\_name(table1\_name)

t2\_n = self.v.valid.check\_table\_name(table2\_name)

**if** t1\_n **and** self.v.valid.check\_key\_names(t1\_n, table1\_key) **and** t2\_n \

**and** self.v.valid.check\_key\_names(t2\_n, table2\_key):

start\_time = time.time()

result = self.m.search\_data\_two\_tables(table1\_name, table2\_name, table1\_key, table2\_key, search)

self.v.time\_output(start\_time)

self.v.print\_search(result)

*model.py*

**import** datetime

**import** psycopg2 **as** ps

**class** Model:

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

self.conn = **None**

**try**:

self.conn = ps.connect(

database=**"Pharmacy"**,

user=**"postgres"**,

password=**"1234"**,

host=**"127.0.0.1"**,

port=**"5432"**,

)

**except**(Exception, ps.DatabaseError) **as** error:

print(**"[INFO] Error when working with PostgreSQL"**, error)

**def** request(self, req: str):

**try**:

cursor = self.conn.cursor()

print(req)

cursor.execute(req)

self.conn.commit()

**return True**

**except**(Exception, ps.DatabaseError, ps.ProgrammingError) **as** error:

print(error)

self.conn.rollback()

**return False**

**def** get(self, req: str):

**try**:

cursor = self.conn.cursor()

print(req)

cursor.execute(req)

self.conn.commit()

**return** cursor.fetchall()

**except**(Exception, ps.DatabaseError, ps.ProgrammingError) **as** error:

print(error)

self.conn.rollback()

**return False**

**def** get\_el(self, req: str):

**try**:

cursor = self.conn.cursor()

print(req)

cursor.execute(req)

self.conn.commit()

**return** cursor.fetchone()

**except**(Exception, ps.DatabaseError, ps.ProgrammingError) **as** error:

print(error)

self.conn.rollback()

**return False**

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

**return** self.get\_el(**f"select count(\*) from public.\"{**table\_name**}\""**)

**def** find(self, table\_name: str, key\_name: str, key\_value: int):

**return** self.get\_el(**f"select count(\*) from public.\"{**table\_name**}\" where {**key\_name**}={**key\_value**}"**)

**def** min(self, table\_name: str, key\_name: str):

**return** self.get\_el(**f"select min({**key\_name**}) from public.\"{**table\_name**}\""**)

**def** max(self, table\_name: str, key\_name: str):

**return** self.get\_el(**f"select max({**key\_name**}) from public.\"{**table\_name**}\""**)

**def** print\_provider(self) -> **None**:

**return** self.get(**f"SELECT \* FROM public.\"Provider\""**)

**def** print\_phone\_numbers(self) -> **None**:

**return** self.get(**f"SELECT \* FROM public.\"Phone numbers\""**)

**def** print\_supplier\_order\_base(self) -> **None**:

**return** self.get(**f"SELECT \* FROM public.\"Supplier order base\""**)

**def** print\_pharmacy(self) -> **None**:

**return** self.get(**f"SELECT \* FROM public.\"Pharmacy\""**)

**def** print\_medicine(self) -> **None**:

**return** self.get(**f"SELECT \* FROM public.\"Medicine\""**)

**def** update\_provider\_data(self, provider\_id: int, title: str, address: str) -> **None**:

self.request(**f"UPDATE public.\"Provider\" SET title=\'{**title**}\', address=\'{**address**}\' "**

**f"WHERE provider\_id={**provider\_id**};"**)

**def** update\_phone\_numbers\_data(self, phone\_number: int, provider\_id: int, mobile\_operator: str) -> **None**:

self.request(**f"UPDATE public.\"Phone numbers\" SET provider\_id={**provider\_id**}, "**

**f"mobile\_operator=\'{**mobile\_operator**}\' WHERE phone\_number={**phone\_number**};"**)

**def** update\_supplier\_order\_base\_data(self, key\_value: int, medicine\_id: int, provider\_id: int, pharmacy\_id: int,

number: int, order\_date: datetime.datetime) -> **None**:

self.request(**f"UPDATE public.\"Supplier order base\" SET provider\_id={**provider\_id**},"**

**f"pharmacy\_id={**pharmacy\_id**}, number={**number**}, order\_date=\'{**order\_date**}\',"**

**f"WHERE supplier\_order\_id={**key\_value**}, medicine\_id={**medicine\_id**};"**)

**def** update\_pharmacy\_data(self, pharmacy\_id: int, title: str, phone\_num: int, address: str) -> **None**:

self.request(**f"UPDATE public.\"Pharmacy\" SET pharmacy\_title=\'{**title**}\', phone\_num={**phone\_num**},"**

**f"address=\'{**address**}\' WHERE pharmacy\_id={**pharmacy\_id**};"**)

**def** update\_medicine\_data(self, medicine\_id: int, title: str, manufacturer: str) -> **None**:

self.request(**f"UPDATE public.\"Medicine\" SET medicine\_title=\'{**title**}\', manufacturer=\'{**manufacturer**}\'"**

**f"WHERE medicine\_id={**medicine\_id**};"**)

**def** insert\_provider\_data(self, provider\_id: int, title: str, address: str) -> **None**:

self.request(**f"insert into public.\"Provider\" (provider\_id, title, address) "**

**f"VALUES ({**provider\_id**}, \'{**title**}\', {**address**});"**)

**def** insert\_phone\_numbers\_data(self, phone\_number: int, provider\_id: int, mobile\_operator: str) -> **None**:

self.request(**f"insert into public.\"Phone numbers\" (phone\_num, provider\_id, mobile\_operator) "**

**f"VALUES ({**phone\_number**}, {**provider\_id**}, \'{**mobile\_operator**}\');"**)

**def** insert\_supplier\_order\_base\_data(self, key\_value: int, medicine\_id: int, provider\_id: int, pharmacy\_id: int,

number: int, order\_date: datetime.datetime) -> **None**:

self.request(**f"insert into public.\"Supplier order base\" (supplier\_order\_id, provider\_id, pharmacy\_id, "**

**f"medicine\_id, number, order\_date) VALUES ({**key\_value**}, {**provider\_id**}, {**pharmacy\_id**}, "**

**f"{**medicine\_id**}, {**number**}, \'{**order\_date**}\');"**)

**def** insert\_pharmacy\_data(self, pharmacy\_id: int, title: str, phone\_num: int, address: str) -> **None**:

self.request(**f"insert into public.\"Pharmacy\" (pharmacy\_id, pharmacy\_title, phone\_num, address) "**

**f"VALUES ({**pharmacy\_id**}, \'{**title**}\', {**phone\_num**}, \'{**address**}\');"**)

**def** insert\_medicine\_data(self, medicine\_id: int, title: str, manufacturer: str) -> **None**:

self.request(**f"insert into public.\"Medicine\" (medicine\_id, medicine\_title, manufacturer) "**

**f"VALUES ({**medicine\_id**}, \'{**title**}\', \'{**manufacturer**}\');"**)

**def** delete\_data(self, table\_name: str, key\_name: str, key\_value: int) -> **None**:

self.request(**f"DELETE FROM public.\"{**table\_name**}\" WHERE {**key\_name**}={**key\_value**};"**)

**def** provider\_data\_generator(self, value: int) -> **None**:

**for** a **in** range(value):

self.request(**"insert into public.\"Provider\" select (SELECT MAX(provider\_id)+1 FROM public.\"Provider\"), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''); "**)

**def** medicine\_data\_generator(self, value: int) -> **None**:

**for** a **in** range(value):

self.request(**"insert into public.\"Medicine\" select (SELECT MAX(medicine\_id)+1 FROM public.\"Medicine\"), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''); "**)

**def** pharmacy\_data\_generator(self, value: int) -> **None**:

**for** a **in** range(value):

self.request(**"insert into public.\"Pharmacy\" select (SELECT MAX(pharmacy\_id)+1 FROM public.\"Pharmacy\"), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(12-4)+4):: integer)), ''), "**

**"FLOOR(RANDOM()\*(999999990-1)+1), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''); "**)

**def** phone\_numbers\_data\_generator(self, value: int) -> **None**:

**for** a **in** range(value):

self.request(**"(SELECT provider\_id FROM public.\"Provider\" LIMIT 1 OFFSET (round(random() \* "**

**"((SELECT COUNT(provider\_id) FROM public.\"Provider\")-1)))), "**

**"insert into public.\"Phone numbers\" "**

**"select (SELECT MAX(phone\_num)+1 FROM public.\"Phone numbers\"), "**

**"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "**

**"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''); "**)

**def** supplier\_order\_base\_data\_generator(self, value: int) -> **None**:

**for** a **in** range(value):

self.request(**"insert into public.\"Supplier order base\" "**

**"select (SELECT MAX(supplier\_order\_id)+1 FROM "**

**"public.\"Supplier order base\"), "**

**"(SELECT provider\_id FROM public.\"Provider\" LIMIT 1 OFFSET (round(random() \* "**

**"((SELECT COUNT(provider\_id) FROM public.\"Provider\")-1)))), "**

**"(SELECT pharmacy\_id FROM public.\"Pharmacy\" LIMIT 1 OFFSET (round(random() \* "**

**"((SELECT COUNT(pharmacy\_id) FROM public.\"Pharmacy\")-1)))), "**

**"(SELECT medicine\_id FROM public.\"Medicine\" LIMIT 1 OFFSET (round(random() \* "**

**"((SELECT COUNT(medicine\_id) FROM public.\"Medicine\")-1)))), "**

**"FLOOR(RANDOM()\*(1000-1)+1), "**

**"(SELECT to\_timestamp(1549634400+random()\*70071999));"**)

**def** search\_data\_one\_table(self, table1\_name: str, search: str):

**return** self.get(**f"select \* from public.\"{**table1\_name**}\" "**

**f"where {**search**}"**)

**def** search\_data\_two\_tables(self, table1\_name: str, table2\_name: str, table1\_key, table2\_key, search: str):

**return** self.get(**f"select \* from public.\"{**table1\_name**}\" as one inner join public.\"{**table2\_name**}\" as two "**

**f"on one.\"{**table1\_key**}\"=two.\"{**table2\_key**}\" "**

**f"where {**search**}"**)

*validator.py*

**import** datetime

**class** Validator:

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

self.error = **''**

self.error\_flag = **False**

**def** check\_table\_name(self, table\_name: str):

**if** table\_name **in** [**'Provider'**, **'Pharmacy'**, **'Medicine'**, **'Phone\_numbers'**, **'Supplier\_order\_base'**]:

**return** table\_name

**else**:

self.error\_flag = **True**

self.error = **f'Table {**table\_name**} does not exist in the database'**

print(self.error)

**return False**

**def** check\_pkey\_value(self, key\_value: str, min\_val: int, max\_val: int):

**try**:

value = int(key\_value)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**key\_value**} is not correct primary key value'**

print(self.error)

**return** 0

**else**:

**if** min\_val <= value <= max\_val:

**return** value

**else**:

self.error\_flag = **True**

self.error = **f'{**key\_value**} is not existing primary key value'**

print(self.error)

**return** 0

**def** check\_pkey\_name(self, table\_name: str, key\_name: str):

**if** table\_name == **'Provider' and** key\_name == **'provider\_id'** \

**or** table\_name == **'Medicine' and** key\_name == **'medicine\_id'** \

**or** table\_name == **'Pharmacy' and** key\_name == **'pharmacy\_id'** \

**or** table\_name == **'Phone numbers' and** key\_name == **'phone\_num'** \

**or** table\_name == **'Supplier order base' and** key\_name == **'supplier\_order\_id and medicine\_id'**:

**return** key\_name

**else**:

self.error\_flag = **True**

self.error = **f'key {**key\_name**} is not a primary key of table {**table\_name**}'**

**return False**

**def** check\_pkey(self, val, count):

**try**:

value = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not correct primary key value'**

print(self.error)

**return** 0

**else**:

**if** count **and not** count == (0,):

**return** value

**else**:

**return** 0

**def** check\_key\_names(self, table\_name: str, key: str):

**if** table\_name == **'Provider' and** key **in** [**'provider\_id'**, **'title'**, **'address'**]:

**return True**

**elif** table\_name == **'Phone\_numbers' and** key **in** [**'phone\_num'**, **'provider\_id'**, **'mobile\_operator'**]:

**return True**

**elif** table\_name == **'Pharmacy' and** key **in** [**'pharmacy\_id'**, **'pharmacy\_title'**, **'phone\_num'**, **'address'**]:

**return True**

**elif** table\_name == **'Medicine' and** key **in** [**'medicine\_id'**, **'medicine\_title'**, **'manufacturer'**]:

**return True**

**elif** table\_name == **'Supplier\_order\_base' and** key **in** [**'supplier\_order\_id'**, **'medicine\_id'**, **'pharmacy\_id'**,

**'provider\_id'**, **'number'**, **'order\_date'**]:

**return True**

**else**:

self.error\_flag = **True**

self.error = **f'{**key**} is not correct name for {**table\_name**} table'**

print(self.error)

**return False**

**def** check\_keys(self, table\_name: str, key: str, val: str):

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

**if** key **in** [**'pharmacy\_id'**]:

**try**:

value = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible key value'**

print(self.error)

**return False**

**else**:

**return True**

**elif** key **in** [**'pharmacy\_title'**, **'address'**]:

**return True**

**elif** key **in** [**'phone\_num'**]:

**try**:

number = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible phone number value'**

print(self.error)

**return False**

**else**:

**return True**

**else**:

self.error\_flag = **True**

self.error = **f'{**key**} is not correct name for Pharmacy table'**

print(self.error)

**return False**

**elif** table\_name == **'Phone numbers'**:

**if** key **in** [**'provider\_id'**, **'phone\_num'**]:

**try**:

value = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible key value'**

print(self.error)

**return False**

**else**:

**return True**

**elif** key **in** [**'mobile\_operator'**]:

**return True**

**else**:

self.error\_flag = **True**

self.error = **f'{**key**} is not correct name for Phone numbers table'**

print(self.error)

**return False**

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

**if** key **in** [**'provider\_id'**]:

**try**:

value = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible key value'**

print(self.error)

**return False**

**else**:

**return True**

**elif** key **in** [**'title'**, **'address'**]:

**return True**

**else**:

self.error\_flag = **True**

self.error = **f'{**key**} is not correct name for Provider table'**

print(self.error)

**return False**

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

**if** key == **'medicine\_id'**:

**try**:

value = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible key value'**

print(self.error)

**return False**

**else**:

**return True**

**elif** key **in** [**'medicine\_title'**, **'manufacturer'**]:

**return True**

**else**:

self.error\_flag = **True**

self.error = **f'{**key**} is not correct name for Medicine table'**

print(self.error)

**return False**

**elif** table\_name == **'Supplier order base'**:

**if** key **in** [**'pharmacy\_id'**, **'medicine\_id'**, **'provider\_id'**, **'supplier\_order\_id'**]:

**try**:

value = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible key value'**

print(self.error)

**return False**

**else**:

**return True**

**elif** key == **'number'**:

**try**:

number = int(val)

**except** ValueError:

self.error\_flag = **True**

self.error = **f'{**val**} is not possible number value'**

print(self.error)

**return False**

**else**:

**return True**

**elif** key == **'order\_date'**:

**try**:

arr = [int(x) **for** x **in** val.split(sep=**'.'**)]

datetime.datetime(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5])

**except** TypeError:

self.error\_flag = **True**

self.error = **f'{**val**} is not correct order date value'**

print(self.error)

**return False**

**else**:

**return True**

**else**:

self.error\_flag = **True**

self.error = **f'{**key**} is not correct name for Supplier order base table'**

print(self.error)

**return False**

*view.py*

**import** validator

**import** datetime

**import** time

**class** View:

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

self.valid = validator.Validator()

@staticmethod

**def** not\_delete() -> **None**:

print(**'this record can not delete because it is connected with another table, deleting will be'**

**'throw error'**)

@staticmethod

**def** sql\_error(err) -> **None**:

print(**"[INFO] Error in the process of working with PostgreSQL"**, err)

@staticmethod

**def** insertion\_error() -> **None**:

print(**'Something went wrong(Record does not exist in the table)'**)

@staticmethod

**def** updation\_error() -> **None**:

print(**'Something went wrong'**)

@staticmethod

**def** deletion\_error() -> **None**:

print(**'Something went wrong, deletion with such does not exist'**)

@staticmethod

**def** invalid\_interval() -> **None**:

print(**'invalid interval input'**)

@staticmethod

**def** time\_output(start) -> **None**:

print(**"---- %s seconds ----"** % (time.time() - start))

@staticmethod

**def** print\_pharmacy(table):

print(**'----Pharmacy table----'**)

**for** row **in** table:

print(**'pharmacy\_id:'**, row[0], **'\tpharmacy\_title:'**, row[1], **'\tphone\_num:'**, row[2], **'\taddress:'**, row[3])

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

@staticmethod

**def** print\_medicine(table):

print(**'----Medicine table----'**)

**for** row **in** table:

print(**'medicine\_id:'**, row[0], **'\tmedicine\_title:'**, row[1], **'\tmanufacturer:'**, row[2])

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

@staticmethod

**def** print\_provider(table):

print(**'----Provider table----'**)

**for** row **in** table:

print(**'provider\_id:'**, row[0], **'\ttitle:'**, row[1], **'\taddress:'**, row[2])

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

@staticmethod

**def** print\_phone\_numbers(table):

print(**'----Phone numbers table----'**)

**for** row **in** table:

print(**'provider\_id:'**, row[0], **'\tphone\_num:'**, row[1], **'\tmobile\_operator:'**, row[2])

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

@staticmethod

**def** print\_supplier\_order\_base(table):

print(**'----Supplier order base----'**)

**for** row **in** table:

print(**'supplier\_order\_id:'**, row[0], **'\tprovider\_id:'**, row[1], **'\tpharmacy\_id:'**, row[2],

**'\tmedicine\_id:'**, row[3], **'\tnumber:'**, row[4], **'\torder\_date:'**, row[5])

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

**def** print\_search(self, res):

print(**'----Search results----'**)

**for** row **in** res:

**for** a **in** range(0, len(row)):

print(row[a])

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

@staticmethod

**def** print\_help():

print(**'print\_table - displays the specified table \n'**

**'\targument is (table\_name)'**)

print(**'delete\_record - deletes the specified record from a specific table \n'**

**' arguments dor deletion:\n'**

**'\t(table\_name, key\_name, key\_value)'**)

print(**'update\_record - adjusts the record with the specified key in a specific table\n'**

**' arguments for Provider table: \n'**

**'\t(table\_name, provider\_id, title, address)\n'**

**' arguments for Pharmacy table: \n'**

**'\t(table\_name, pharmacy\_id, pharmacy\_title, phone\_num, address)\n'**

**' arguments for Medicine table: \n'**

**'\t(table\_name, medicine\_id, medicine\_title, manufacturer)\n'**

**' arguments for Phone\_numbers table: \n'**

**'\t(table\_name, provider\_id, phone\_num, mobile\_operator)\n'**

**' arguments for Supplier\_order\_base table: \n'**

**'\t(table\_name, supplier\_oder\_id, provider\_id, pharmacy\_id, medicine\_id, number, order\_date)'**)

print(**'insert\_record - enter data in the specified table\n'**

**' arguments for Provider table: \n'**

**'\t(table\_name, provider\_id, title, address)\n'**

**' arguments for Pharmacy table: \n'**

**'\t(table\_name, pharmacy\_id, pharmacy\_title, phone\_num, address)\n'**

**' arguments for Medicine table: \n'**

**'\t(table\_name, medicine\_id, medicine\_title, manufacturer)\n'**

**' arguments for Phone\_numbers table: \n'**

**'\t(table\_name, provider\_id, phone\_num, mobile\_operator)\n'**

**' arguments for Supplier\_order\_base table: \n'**

**'\t(table\_name, supplier\_oder\_id, provider\_id, pharmacy\_id, medicine\_id, number, order\_date)'**)

print(**'generate\_records - generates n randomized records in the specified table\n'**

**' arguments for generation:'**

**'\t(table\_name, n)'**)

print(**'search\_records - search for records in one table using one or more attributes \n'**

**' arguments for searching\n'**

**'\t(table1\_name, table1\_key)'**)

**def** initiate\_search(self, search\_number):

search\_value = **''**

**for** param **in** range(0, search\_number):

**while True**:

search\_type = input(**'Specify the type(date, numeric or string) of data you want to find: '**)

**if** search\_type **in** [**'date'**, **'numeric'**, **'string'**]:

**break**

key\_value = input(**'Specify the key name by which you`d find(in form key\_name if for one table, else - '**

**'table\_name.key\_name):'**)

**if** search\_type == **'numeric'**:

left\_edge = input(**'Specify the left edge of search interval:'**)

right\_edge = input(**'Specify the right edge of search interval:'**)

**if** search\_value == **''**:

search\_value = self.numeric\_search(left\_edge, right\_edge, key\_value)

**else**:

search\_value += **' and '** + self.numeric\_search(left\_edge, right\_edge, key\_value)

**elif** search\_type == **'string'**:

string\_value = input(**'Specify the string which you`d to find: '**)

**if** search\_value == **''**:

search\_value = self.string\_search(string\_value, key\_value)

**else**:

search\_value += **' and '** + self.string\_search(string\_value, key\_value)

**elif** search\_type == **'date'**:

date\_left = input(**'Specify the left edge of search interval in form yy.mm.dd: '**)

date\_right = input(**'Specify the left edge of search interval in form yy.mm.dd: '**)

**if** search\_value == **''**:

search\_value = self.date\_search(date\_left, date\_right, key\_value)

**else**:

search\_value += **'and'** + self.date\_search(date\_left, date\_right, key\_value)

**return** search\_value

**def** numeric\_search(self, left: str, right: str, key: str):

**try**:

left, right = int(left), int(right)

**except** ValueError:

self.invalid\_interval()

**else**:

**return f"{**left**} <= {**key**} and {**key**} <= {**right**}"**

**def** string\_search(self, string: str, key: str):

**return f"{**key**} LIKE \'{**string**}\'"**

**def** date\_search(self, left: str, right: str, key: str):

**try**:

l\_edge = [int(a) **for** a **in** left.split(sep=**'.'**)]

r\_edge = [int(a) **for** a **in** right.split(sep=**'.'**)]

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

print(e)

self.invalid\_interval()

**else**:

**return f"{**key**} BETWEEN "** \

**f"\'{**datetime.datetime(l\_edge[0], l\_edge[1], l\_edge[2])**}\' "** \

**f"AND "** \

**f"\'{**datetime.datetime(r\_edge[0], r\_edge[1], r\_edge[2])**}\'"**

@staticmethod

**def** get\_search\_num():

**return** input(**'Specify the number of attributes which you`d to find: '**)

@staticmethod

**def** invalid\_search\_num():

print(**'The number should be different from 0'**)

@staticmethod

**def** arg\_error():

print(**'No required arguments specified'**)

@staticmethod

**def** wrong\_table():

print(**'Wrong table name'**)

@staticmethod

**def** wrong\_command():

print(**'Wrong command, please use Help to see correct commands'**)

@staticmethod

**def** no\_command():

print(**'No command, please use Help to see correct commands'**)