

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота № 2**

з дисципліни «Бази даних і засоби управління»

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

Виконав: Іванов Максим

Студент групи КВ-92

Перевірив: Павловський В.І.

**Київ 2021**

**Лабораторна робота №2**

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 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 магазину «Автосалон»**

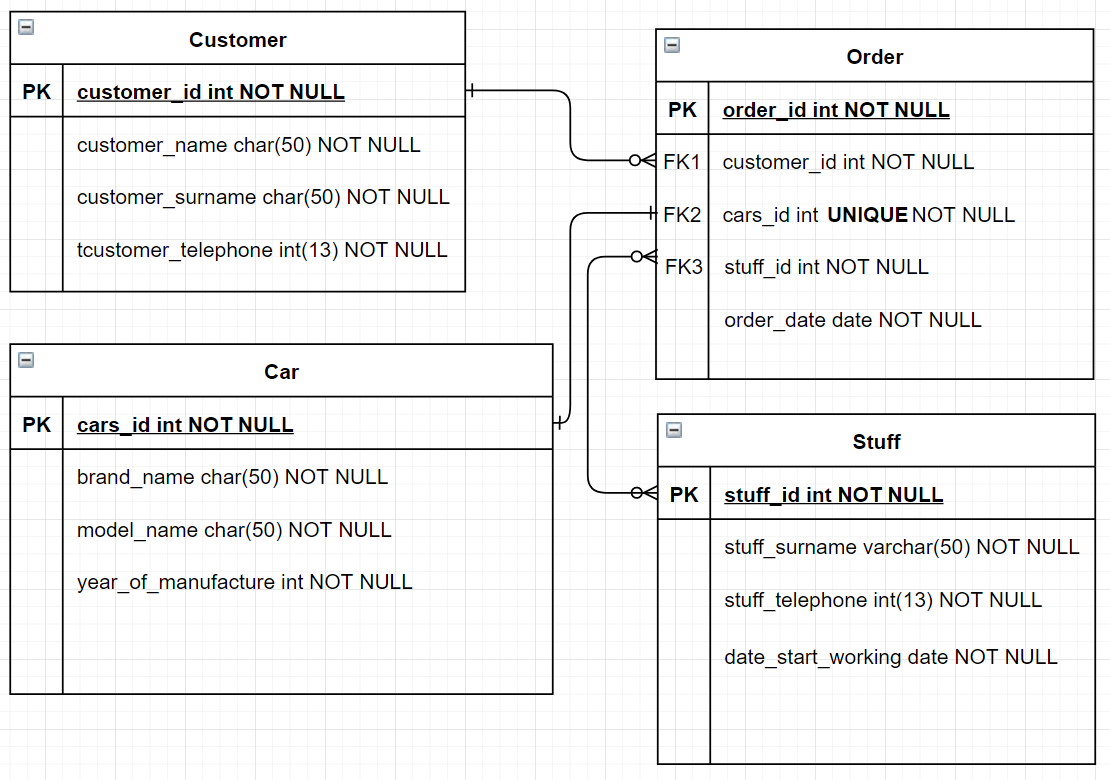


Рисунок0 – логічна модель «Автосалон»

**Середовище розробки**

Середовище розробки бази даних - PostgreSQL

Середовище розробки програми – Visual Studio. Мова програмування Python3.6.

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), datetime (для роботи з датою і передачею її у запити до БД), time (для виміру часу запиту пошуку для завдання 3), sys (для реалізації консольного інтерфейсу).

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити. Це файл model.py. Для цього в ньому використовується бібліотека мови Python – psycopg2. Конструктор класу Model налагоджує зв’язок із сервером і видає повідомлення про помилку, якщо зв’язок не було встановлено.

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

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання. Згідно компоненту контролер, то йому відповідає файл Controller.py.

**Структура програми та її опис**

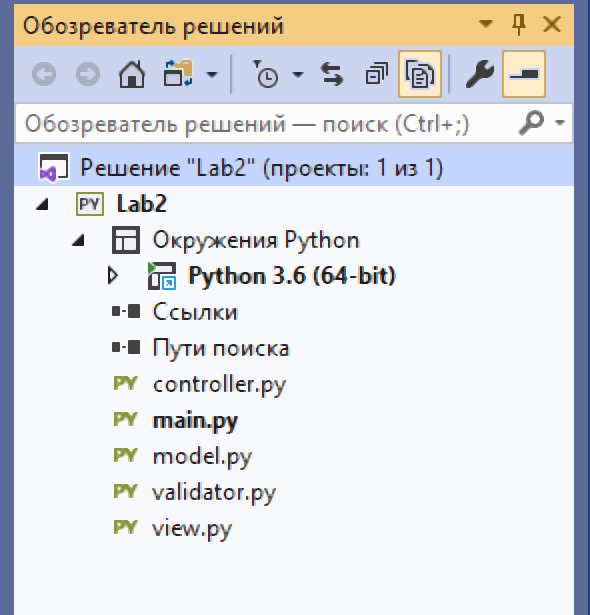
****

Рисунок 1 – структура програми

Файл model.py

Виконує операції з БД.

Методи request, get, get\_el здійснюють запити до бази даних за допомогою cursor вони всі повертають False якщо виникла помилка і запит не відбувся, а якщо вдалося зробити запит, request повертає True, get повертає усі дані що було взято з запитів SELECT (масив кортежів з записами таблиць), get\_el повертає тільки перший запис.

Метод count повертає кількість усіх записів таблиці.

Метод find повертає кількість записів таблиці, що відповідають певній умові (або False, якщо записів не знайдено).

Методи max, min повертають відповідно максимальне і мінімальне значення зазначеного ключа у таблиці.

Методи print\_products, print\_order, print\_catalog, print\_shop здійснюють отримання з БД відповідних таблиць.

Метод delete\_data реалізує запит на видалення відповідного запису таблиці.

Методи update\_data\_(table name) відправляють до БД запит на оновлення даних у певному полі таблиці.

Методи insert\_data\_(table name) відправляють до БД запит на вставку запису в таблицю.

Методи (table name)\_data\_generator реалізують запит до БД на вставку псевдорандомізованих даних (для первинного ключа максимальне вже існуюче значення +1, для зовнішніх ключів — одне із можливих значень для цього ключа).

Методи search\_data\_(number)\_tables реалізують запит на отримання результату пошуку серед number таблиць за рядком пошуку, що генерується в view методами: numeric\_search, string\_search, date\_search.

Файл view.py

Відповідає за ввід/вивід інформації.

Метод cannot\_delete. Виводить на екран повідомлення про неможливість видалення запису, який прив’язаний до іншої таблиці з поясненням можливої причини помилки.

Метод sql\_error. Виводить повідомлення про неможливість встонвлення зв’язку з базою данних.

Метод insertion\_error. Виводить на екран повідомлення про неможливість вставки запису з поясненням можливої причини помилки.

Метод updation\_error. Виводить на екран повідомлення про неможливість оновлення запису з поясненням можливої причини помилки.

Метод deletion\_error. Виводить на екран повідомлення про неможливість видалення запису з поясненням можливої причини помилки.

Метод invalid\_interval. Виводить на екран повідомлення про синтаксичну помилку вводу користувача.

Метод print\_time. Виводить на екран повідомлення про кількість часу з початку роботи програми.

Метод print\_search. Виводить на екран результат пошуку.

Метод print\_cars. Виводить на екран таблицю «Cars».

Метод print\_customers. Виводить на екран таблицю «Customers».

Метод print\_orders. Виводить на екран таблицю «Orders».

Метод print\_stuff. Виводить на екран таблицю «Stuff».

Метод print\_help. Виводить на екран список команд, які можна використовувати в програмі .

Метод proceed\_search. Допомагає користувачу знайти будь-яку інформацію в базі данних.

Метод numeric\_search. Допомагає користувачу знайти будь-яку інформацію в базі данних по ключу.

Метод date\_search. Допомагає користувачу знайти будь-яку інформацію в базі данних по даті.

Метод string\_search. Допомагає користувачу знайти будь-яку інформацію в базі данних по значенню.

Метод get\_search\_num. Допомагає користувачу знайти будь-яку інформацію в базі данних по певним атрибутам.

Метод invalid\_search\_num. Виводить на екран повідомлення про те, що запис повинен не дорівнювати нулю.

Метод argument\_error. Виводить на екран повідомлення про неправильний аргумент у введених данних користувача.

Метод wrong\_table. Виводить на екран повідомлення про неправильно надане ім’я таблиці.

Метод no\_command. Виводить на екран повідомлення про неіснування наданої команди, або про те що команду взагалі не було надано.

Метод wrong\_command. Виводить на екран повідомлення про надання неправильної команди.

Файл validator.py

Перевіряє введені та надані користувачем данні.

Метод check\_table\_name. Перевіряє існування введеної таблиці.

Метод check\_pkey\_value. Перевіряє існування значення введеного первинного ключа.

Метод check\_pk\_name. Перевіряє приналежність введеного первинного ключа до таблиці.

Метод check\_key\_names. Перевіряє корректність імені введеного ключа.

Метод check\_possible\_keys. Перевіряє усі можливі ключі і до якої таблиці вони належать.

Файл main.py

Точка входу в програму, запускає початковий інтерфейс.

Файл controller.py

Обробляє ввід користувача, подає відповідну команду до model.py і очікує результат, який потім виводить користувачу.

Метод printf. Описує логіку обробки запиту на виведення таблиці.

Метод delete. Описує логіку обробки запиту на видалення запису таблиці.

Метод update\_<ім’я таблиці>. Описує логіку обробки запиту на оновлення даних у певній таблиці.

Метод insert\_<ім’я таблиці>. Описує логіку обробки запиту на вставку нових данних у таблицю.

Метод generate. Описує логіку обробки запиту на генерацію випадкових данних у певній таблиці.

Метод search\_two. Описує логіку обробки запиту на пошук у двух таблицях.

Метод search\_three. Описує логіку обробки запиту на пошук у трьох таблицях.

Метод search\_four. Описує логіку обробки запиту на пошук у чотирьох таблицях.

**Структура меню програми**

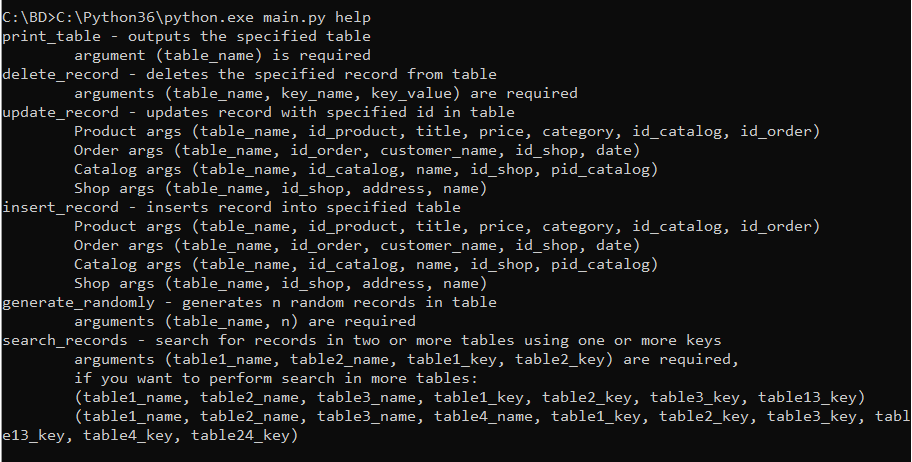


Рисунок 2 – виконання команди help

На знімку екрану терміналу продемонстовано виконання команди help, що показує усі доступні користувачу команди, коротко описує їх та надає список обов’язкових аргументів. Кожна команда запускає відповідний метод об’єкту класу Controller, який реалізує передачу аргументів у клас View на перевірку і за умови їх коректності, Controller далі передає ці аргументи у клас Model, що здійснює запит до бази даних.

Методи реалізовані до пункту 1 завдання лабораторної роботи:

print\_table – за умови коректності імені таблиці виводить вміст цієї таблиці у вікно терміналу. Аргументом може бути одне із імен: Product, Order, Catalog, Shop

delete\_record – за умови правильності введених аргументів, наявності відповідного запису (з вказаним значенням первинного ключа) та незалежності інших таблиць від цього запису (до цього запису немає зовнішнього ключа з іншої таблиці), видаляє запис з вказаним первинним ключем. Аргументами є table\_name, key\_name, key\_value.

update\_record – за умови правильності введених аргументів, наявності відповідного запису (з вказаним значенням первинного ключа) та записів інших таблиць (на які хочемо змінити поточні), змінює усі поля, окрім первинного ключа у обраному записі. Аргументи різні для кожної таблиці.

insert\_record – за умови правильності введених аргументів, відсутності відповідного запису (з вказаним значенням первинного ключа) та наявності записів інших таблиць (на які хочемо посилатись зі створеного запису), вставляє новий рядок у таблицю з обраними значеннями полів. Аргументи різні для кожної таблиці.

Метод реалізований до пункту 2 завдання лабораторної роботи:

generate\_randomly – за умови введення правильного імені таблиці та числа n відмінного від нуля, здійснює генерування n псевдорандомізованих записів у обраній таблиці. Аргементами є ім’я таблиці та часло записів, що мають бути створені.

Метод реалізований до пункту 3 завдання лабораторної роботи:

search\_records – за умови введення потрібної кількості аргументів та правильного задання умов пошуку, реалізує пошук за 1 та більше атрибутами з вказаних таблиць (від двох до чотирьох) і виводить у вікно терміналу результат пошуку (або нічого, якщо пошук не дав результатів) та час, за який було проведено запит. Початково потрібно вказати агрументи: table1\_name table2\_name table1\_key table2\_key aбо table1\_name table2\_name table3\_name table1\_key table2\_key table3\_key table13\_key або table1\_name table2\_name table3\_name table4\_name table1\_key table2\_key table3\_key table13\_key table4\_key table24\_key де table13\_key, table24\_key – це зовнішні ключі, що зв’язують 1 та 3 таблицю, або 2 та 4.

Після вказання цієї інформації потрібно буде вказати кількість атрибутів для пошуку, а тип пошуку, ім’я атрибуту (обов’язково з вказанням до якої таблиці з перелічених аргументів він відноситься: one.key\_name, two.key\_name, three.key\_name або four.key\_name), та значення (спочатку лівий кінець інтервалу, потім правий для числового пошуку та пошуку за датою, або рядок для пошуку за ключовим словом). Спочатку вказуються всі дані для першого атрибуту, потім для другого і т.д. до введеної кількості атрибутів.

\*Важливо: дата з часовою міткою у програмі вказується через крапку у такому форматі: year.month.day.hour.minute.second

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

def insert\_data\_cars(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.request(f"insert into public.\"Cars\" (car\_id, brand\_name , model\_name , year\_of\_manufacture) "

f"VALUES ({car\_id}, \'{brand\_name}\', \'{model\_name}\', \'{year\_of\_manufacture}\');")

def insert\_data\_customers(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.request(f"insert into public.\"Customers\" (customer\_id, customer\_name, customer\_surname, customer\_telephone) "

f"VALUES ({customer\_id}, \'{customer\_name}\', \'{customer\_surname}\', \'{customer\_telephone}\');")

def insert\_data\_orders(self, order\_id: int, customer\_id: str, car\_id: int, stuff\_id: int,order\_date:datetime.datetime) -> None:

self.request(f"insert into public.\"Orders\" (order\_id, customer\_id, car\_id, stuff\_id,order\_date) "

f"VALUES ({order\_id}, \'{customer\_id}\', \'{car\_id}\', \'{stuff\_id}\',\'{order\_date}\');")

def insert\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, stuff\_telephone: str,date\_start\_working:datetime.datetime) -> None:

self.request(f"insert into public.\"Stuff\" (stuff\_id, stuff\_surname, stuff\_telephone,date\_start\_working) "

f"VALUES ({stuff\_id}, \'{stuff\_surname}\', \'{stuff\_telephone}\',\'{date\_start\_working}\');")

Фрагменти програми для внесення даних у таблицях

def update\_data\_customers(self, key\_value: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.request(f"UPDATE public.\"Customers\" SET customer\_name=\'{customer\_name}\', customer\_surname =\'{customer\_surname}\', customer\_telephone=\'{customer\_telephone }\', "

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

def update\_data\_orders(self, key\_value: int, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: datetime.datetime) -> None:

self.request(f"UPDATE public.\"Orders\" SET customer\_id=\'{customer\_id}\', car\_id=\'{car\_id}\', stuff\_id=\'{stuff\_id}\',"

f"order\_date=\'{order\_date}\' WHERE order\_id={key\_value};")

def update\_data\_cars(self, key\_value: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.request(f"UPDATE public.\"Cars\" SET brand\_name=\'{brand\_name}\', model\_name=\'{model\_name}\',"

f"year\_of\_manufacture=\'{year\_of\_manufacture}\' WHERE car\_id={key\_value};")

def update\_data\_stuff(self, key\_value: int, stuff\_surname: str, date\_start\_working:datetime.datetime ,stuff\_telephone:str) -> None:

self.request(f"UPDATE public.\"Stuff\" SET stuff\_surname=\'{stuff\_surname}\', date\_start\_working=\'{date\_start\_working}\',"

f"stuff\_telephone=\'{stuff\_telephone}\' WHERE stuff\_id={key\_value};")

Фрагмент програми для редагування даних у таблицях

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

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

Фрагмент програми для видалення даних у таблицях

Дані фрагменти програми, які наведені вижче, відповідають за функціонал додвання даних, редагування та вилучення даних у базі даних. Результати фрагментів роботи програми наведені в розділі «Результати роботи програми», які знаходяться нижче.

**Лістинги фрагментів програм з запитами пошуку**

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}")

def search\_data\_three\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str,

table1\_key, table2\_key, table3\_key, table13\_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}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\""

f"where {search}")

def search\_data\_all\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str, table4\_name: str,

table1\_key, table2\_key, table3\_key, table13\_key,

table4\_key, table24\_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}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\" inner join public.\"{table4\_name}\" as four "

f"on four.\"{table4\_key}\"=two.\"{table24\_key}\""

f"where {search}")

Лістинг програми для пошуку інформації з двох, трьох та чотирьох таблиць з використанням «inner join».

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

def cars\_data\_generator(self, times: int) -> None:

for i in range(times):

self.request("insert into public.\"Cars\""

"select (SELECT MAX(car\_id)+1 FROM public.\"Cars\"), "

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

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

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

FROM generate\_series(1, FLOOR(RANDOM()\*(10-5)+7):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1),")

def customers\_data\_generator(self, times: int) -> None:

for i in range(times):

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

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

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

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(11-2)+6):: integer)), ''), "

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(5-3)+1):: integer)), ''), "

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

def orders\_data\_generator(self, times: int) -> None:

for i in range(times):

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

"(SELECT customer\_id FROM public.\"Customers\" LIMIT 1 OFFSET "

"(round(random() \*((SELECT COUNT(customer\_id) FROM public.\"Customers\")-1)))), "

"(SELECT car\_id FROM public.\"Cars\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(car\_id) FROM public.\"Cars\")-1)))), "

"(SELECT stuff\_id FROM public.\"Stuff\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(stuff\_id) FROM public.\"Stuff\")-1)))), "

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

def stuff\_data\_generator(self, times: int) -> None:

for i in range(times):

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

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+2):: integer)), ''), "

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(7-4)+9):: integer)), ''), "

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

Лістинг програми для генерування випадкових даних для різних таблиць. В кожній таблиці обирається певне поле і заповнюється випадковими символами, які можна використовувати в данній комірці. На вхід подається параметр, який відповідає за довжину таблиці, тобто кількість записів.

**Лістинг модуля "Model"**

Моделі, які присутні у таблиці бази даних. Назви моделей і полів відповідають назвам полів у базі даних.

import datetime

import psycopg2 as ps

class Model:

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

self.conn = None

try:

self.conn = ps.connect(

database="car\_showroom\_nonrelative",

user='postgres',

password="qwerty",

host='127.0.0.1',

port="5432",

)

except(Exception, ps.DatabaseError) as error:

print("[INFO] Error while 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 max(self, table\_name: str, key\_name: str):

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

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

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

def print\_cars(self) -> None:

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

def print\_customers(self) -> None:

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

def print\_orders(self) -> None:

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

def print\_stuff(self) -> None:

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

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

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

def update\_data\_customers(self, key\_value: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.request(f"UPDATE public.\"Customers\" SET customer\_name=\'{customer\_name}\', customer\_surname =\'{customer\_surname}\', customer\_telephone=\'{customer\_telephone }\', "

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

def update\_data\_orders(self, key\_value: int, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: datetime.datetime) -> None:

self.request(f"UPDATE public.\"Orders\" SET customer\_id=\'{customer\_id}\', car\_id=\'{car\_id}\', stuff\_id=\'{stuff\_id}\',"

f"order\_date=\'{order\_date}\' WHERE order\_id={key\_value};")

def update\_data\_cars(self, key\_value: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.request(f"UPDATE public.\"Cars\" SET brand\_name=\'{brand\_name}\', model\_name=\'{model\_name}\',"

f"year\_of\_manufacture=\'{year\_of\_manufacture}\' WHERE car\_id={key\_value};")

def update\_data\_stuff(self, key\_value: int, stuff\_surname: str, date\_start\_working:datetime.datetime ,stuff\_telephone:str) -> None:

self.request(f"UPDATE public.\"Stuff\" SET stuff\_surname=\'{stuff\_surname}\', date\_start\_working=\'{date\_start\_working}\',"

f"stuff\_telephone=\'{stuff\_telephone}\' WHERE stuff\_id={key\_value};")

def insert\_data\_cars(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.request(f"insert into public.\"Cars\" (car\_id, brand\_name , model\_name , year\_of\_manufacture) "

f"VALUES ({car\_id}, \'{brand\_name}\', \'{model\_name}\', \'{year\_of\_manufacture}\');")

def insert\_data\_customers(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.request(f"insert into public.\"Customers\" (customer\_id, customer\_name, customer\_surname, customer\_telephone) "

f"VALUES ({customer\_id}, \'{customer\_name}\', \'{customer\_surname}\', \'{customer\_telephone}\');")

def insert\_data\_orders(self, order\_id: int, customer\_id: str, car\_id: int, stuff\_id: int,order\_date:datetime.datetime) -> None:

self.request(f"insert into public.\"Orders\" (order\_id, customer\_id, car\_id, stuff\_id,order\_date) "

f"VALUES ({order\_id}, \'{customer\_id}\', \'{car\_id}\', \'{stuff\_id}\',\'{order\_date}\');")

def insert\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, stuff\_telephone: str,date\_start\_working:datetime.datetime) -> None:

self.request(f"insert into public.\"Stuff\" (stuff\_id, stuff\_surname, stuff\_telephone,date\_start\_working) "

f"VALUES ({stuff\_id}, \'{stuff\_surname}\', \'{stuff\_telephone}\',\'{date\_start\_working}\');")

def cars\_data\_generator(self, times: int) -> None:

for i in range(times):

self.request("insert into public.\"Cars\""

"select (SELECT MAX(car\_id)+1 FROM public.\"Cars\"), "

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

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

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

FROM generate\_series(1, FLOOR(RANDOM()\*(10-5)+7):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1),")

def customers\_data\_generator(self, times: int) -> None:

for i in range(times):

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

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

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

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(11-2)+6):: integer)), ''), "

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(5-3)+1):: integer)), ''), "

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

def orders\_data\_generator(self, times: int) -> None:

for i in range(times):

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

"(SELECT customer\_id FROM public.\"Customers\" LIMIT 1 OFFSET "

"(round(random() \*((SELECT COUNT(customer\_id) FROM public.\"Customers\")-1)))), "

"(SELECT car\_id FROM public.\"Cars\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(car\_id) FROM public.\"Cars\")-1)))), "

"(SELECT stuff\_id FROM public.\"Stuff\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(stuff\_id) FROM public.\"Stuff\")-1)))), "

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

def stuff\_data\_generator(self, times: int) -> None:

for i in range(times):

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

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+2):: integer)), ''), "

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(7-4)+9):: integer)), ''), "

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

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}")

def search\_data\_three\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str,

table1\_key, table2\_key, table3\_key, table13\_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}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\""

f"where {search}")

def search\_data\_all\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str, table4\_name: str,

table1\_key, table2\_key, table3\_key, table13\_key,

table4\_key, table24\_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}\" inner join public.\"{table3\_name}\" as three "

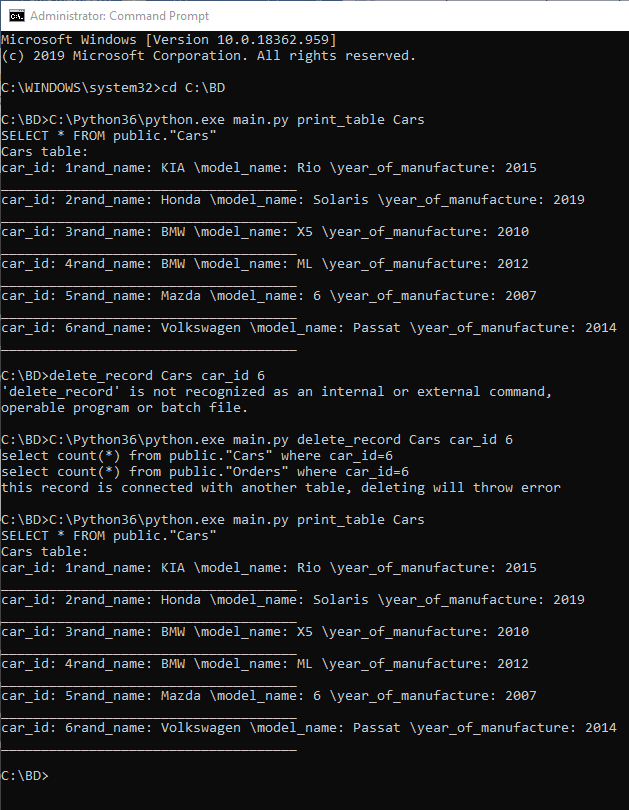
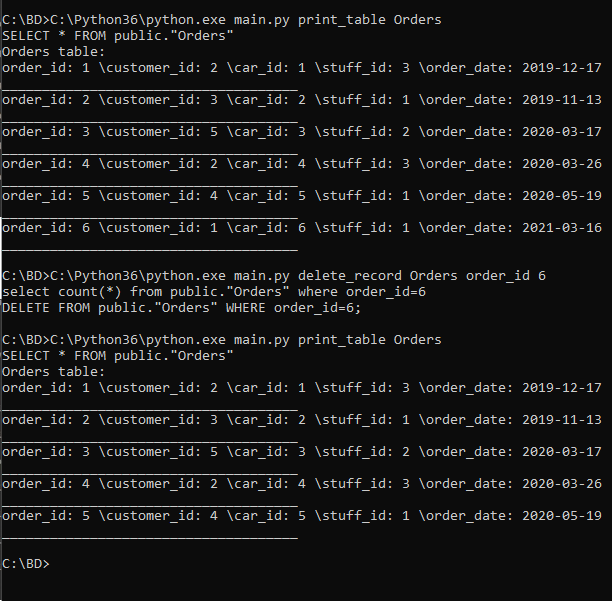
f"on three.\"{table3\_key}\"=one.\"{table13\_key}\" inner join public.\"{table4\_name}\" as four "

f"on four.\"{table4\_key}\"=two.\"{table24\_key}\""

f"where {search}")

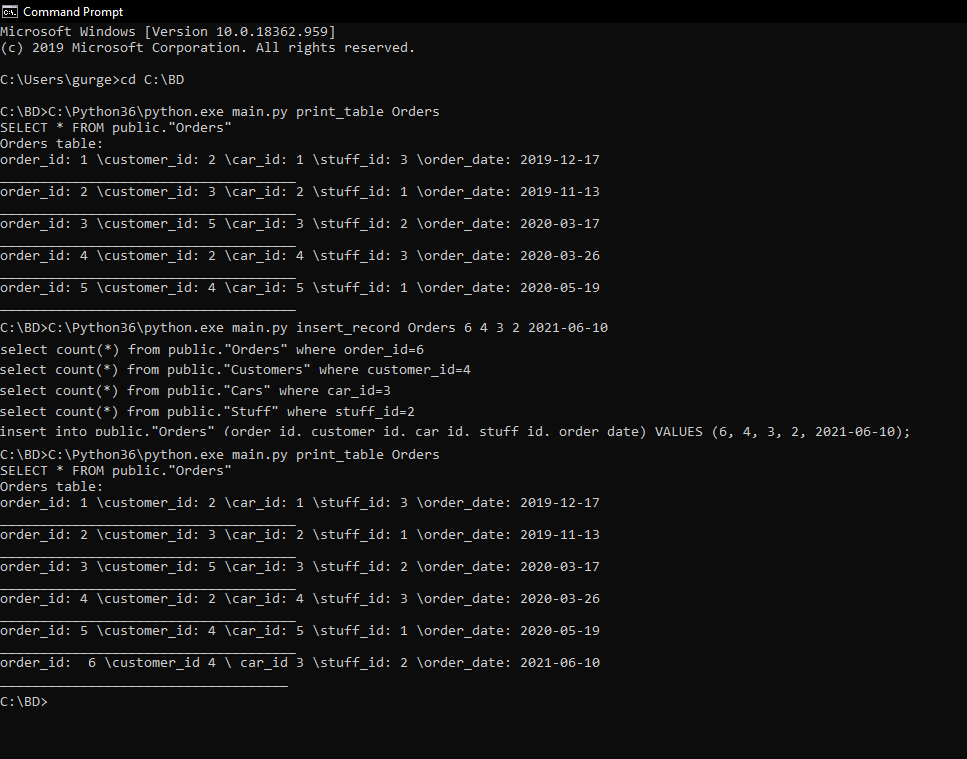
**Завдання 1**

**Запит на видалення**

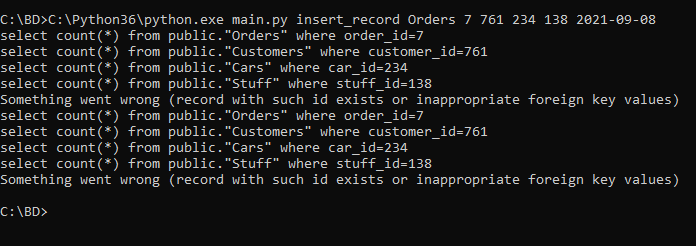
****

**Запит на вставку поля**

Для перевірки роботи розглянемо запити на вставки в дочірню таблицю Orders. Спочатку коректний, потім з неіснуючим значенням зовнішнього ключа.

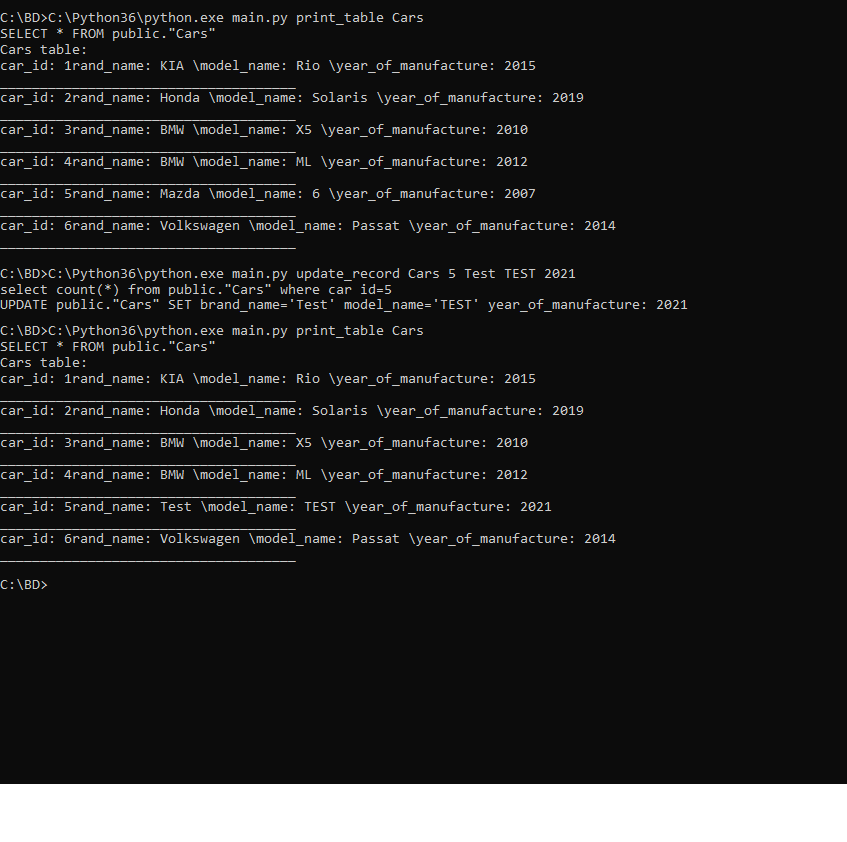


Спроба вставки некоректного запису:

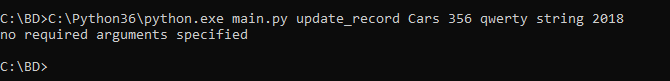


**Запит на зміну полів**

Для перевірки роботи розглянемо запити на зміну значенння в дочірній таблиці Cars. Спочатку коректний, потім з неіснуючим значенням ключа.

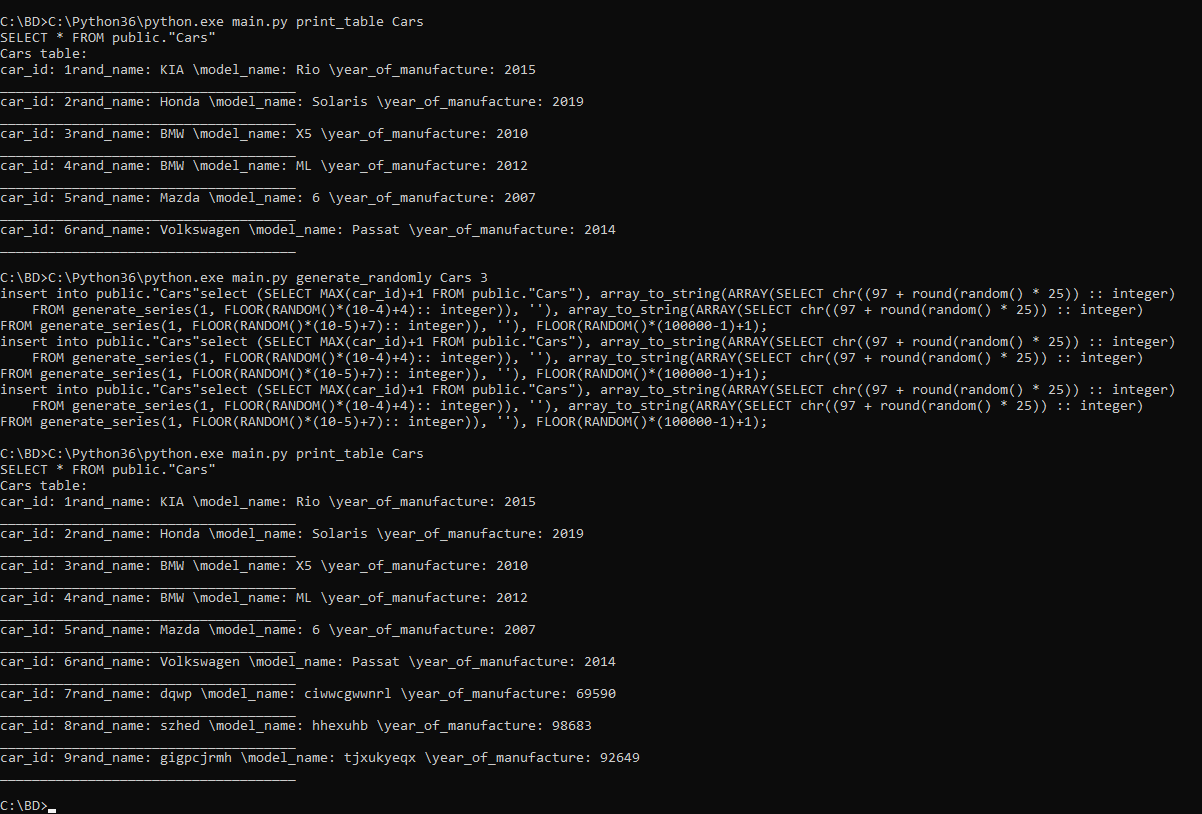


Спроба некоректного оновлення запису:

****

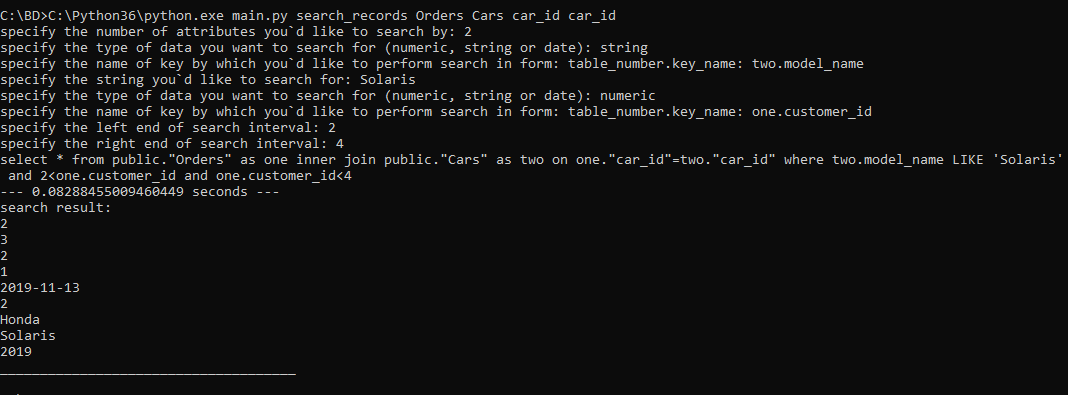
# Завдання 2

Вставка псевдорандомізованих записів



# Завдання 3

Пошук за декількома атрибутами з декількох таблиць.



**Ілюстрації програмного коду з репозиторію Git**

Код програмного модулю “model.py”

import datetime

import psycopg2 as ps

class Model:

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

self.conn = None

try:

self.conn = ps.connect(

database="car\_showroom\_nonrelative",

user='postgres',

password="qwerty",

host='127.0.0.1',

port="5432",

)

except(Exception, ps.DatabaseError) as error:

print("[INFO] Error while 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 max(self, table\_name: str, key\_name: str):

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

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

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

def print\_cars(self) -> None:

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

def print\_customers(self) -> None:

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

def print\_orders(self) -> None:

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

def print\_stuff(self) -> None:

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

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

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

def update\_data\_customers(self, key\_value: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.request(f"UPDATE public.\"Customers\" SET customer\_name=\'{customer\_name}\', customer\_surname =\'{customer\_surname}\', customer\_telephone=\'{customer\_telephone }\', "

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

def update\_data\_orders(self, key\_value: int, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: datetime.datetime) -> None:

self.request(f"UPDATE public.\"Orders\" SET customer\_id=\'{customer\_id}\', car\_id=\'{car\_id}\', stuff\_id=\'{stuff\_id}\',"

f"order\_date=\'{order\_date}\' WHERE order\_id={key\_value};")

def update\_data\_cars(self, key\_value: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.request(f"UPDATE public.\"Cars\" SET brand\_name=\'{brand\_name}\', model\_name=\'{model\_name}\',"

f"year\_of\_manufacture=\'{year\_of\_manufacture}\' WHERE car\_id={key\_value};")

def update\_data\_stuff(self, key\_value: int, stuff\_surname: str, date\_start\_working:datetime.datetime ,stuff\_telephone:str) -> None:

self.request(f"UPDATE public.\"Stuff\" SET stuff\_surname=\'{stuff\_surname}\', date\_start\_working=\'{date\_start\_working}\',"

f"stuff\_telephone=\'{stuff\_telephone}\' WHERE stuff\_id={key\_value};")

def insert\_data\_cars(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.request(f"insert into public.\"Cars\" (car\_id, brand\_name , model\_name , year\_of\_manufacture) "

f"VALUES ({car\_id}, \'{brand\_name}\', \'{model\_name}\', \'{year\_of\_manufacture}\');")

def insert\_data\_customers(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.request(f"insert into public.\"Customers\" (customer\_id, customer\_name, customer\_surname, customer\_telephone) "

f"VALUES ({customer\_id}, \'{customer\_name}\', \'{customer\_surname}\', \'{customer\_telephone}\');")

def insert\_data\_orders(self, order\_id: int, customer\_id: str, car\_id: int, stuff\_id: int,order\_date:datetime.datetime) -> None:

self.request(f"insert into public.\"Orders\" (order\_id, customer\_id, car\_id, stuff\_id,order\_date) "

f"VALUES ({order\_id}, \'{customer\_id}\', \'{car\_id}\', \'{stuff\_id}\',\'{order\_date}\');")

def insert\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, stuff\_telephone: str,date\_start\_working:datetime.datetime) -> None:

self.request(f"insert into public.\"Stuff\" (stuff\_id, stuff\_surname, stuff\_telephone,date\_start\_working) "

f"VALUES ({stuff\_id}, \'{stuff\_surname}\', \'{stuff\_telephone}\',\'{date\_start\_working}\');")

def cars\_data\_generator(self, times: int) -> None:

for i in range(times):

self.request("insert into public.\"Cars\""

"select (SELECT MAX(car\_id)+1 FROM public.\"Cars\"), "

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

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

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

FROM generate\_series(1, FLOOR(RANDOM()\*(10-5)+7):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1);")

def customers\_data\_generator(self, times: int) -> None:

for i in range(times):

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

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

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

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(11-2)+6):: integer)), ''), "

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(5-3)+1):: integer)), ''), "

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

def orders\_data\_generator(self, times: int) -> None:

for i in range(times):

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

"(SELECT customer\_id FROM public.\"Customers\" LIMIT 1 OFFSET "

"(round(random() \*((SELECT COUNT(customer\_id) FROM public.\"Customers\")-1)))), "

"(SELECT car\_id FROM public.\"Cars\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(car\_id) FROM public.\"Cars\")-1)))), "

"(SELECT stuff\_id FROM public.\"Stuff\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(stuff\_id) FROM public.\"Stuff\")-1)))), "

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

def stuff\_data\_generator(self, times: int) -> None:

for i in range(times):

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

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+2):: integer)), ''), "

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

"FROM generate\_series(1, FLOOR(RANDOM()\*(7-4)+9):: integer)), ''), "

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

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}")

def search\_data\_three\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str,

table1\_key, table2\_key, table3\_key, table13\_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}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\""

f"where {search}")

def search\_data\_all\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str, table4\_name: str,

table1\_key, table2\_key, table3\_key, table13\_key,

table4\_key, table24\_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}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\" inner join public.\"{table4\_name}\" as four "

f"on four.\"{table4\_key}\"=two.\"{table24\_key}\""

f"where {search}")

Код програмного модулю “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.printf(name)

elif command == 'delete\_record':

try:

args = {"name": sys.argv[2], "key": sys.argv[3], "val": sys.argv[4]}

except IndexError:

c.v.argument\_error()

else:

c.delete(args["name"], args["key"], args["val"])

elif command == 'update\_record':

try:

args = {"name": sys.argv[2], "key": sys.argv[3]}

if args["name"] == 'Cars':

args["brand\_name"], args["model\_name"], args["year\_of\_manufacture "] = \

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

elif args["name"] == 'Customers':

args["customer\_name"], args["customer\_surname"], args["customer\_telephone"] = \

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

elif args["name"] == 'Orders':

args["customer\_id"], args["car\_id"], args["order\_date"] = \

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

elif args["name"] == 'Stuff':

args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"]= \

sys.argv[4], sys.argv[5]

else:

c.v.wrong\_table()

except IndexError:

c.v.argument\_error()

else:

if args["name"] == 'Cars':

c.update\_product(args["key"], args["brand\_name"], args["model\_name"],

args["year\_of\_manufacture"])

elif args["name"] == 'Customers':

c.update\_order(args["key"], args["customer\_name"], args["customer\_surname"], args["customer\_telephone"])

elif args["name"] == 'Orders':

c.update\_catalog(args["key"], args["customer\_id"], args["car\_id"], args["order\_date"])

elif args["name"] == 'Stuff':

c.update\_shop(args["key"], args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"])

elif command == 'insert\_record':

try:

args = {"name": sys.argv[2], "key": sys.argv[3]}

if args["name"] == 'Cars':

args["brand\_name"], args["model\_name"], args["category"], args["year\_of\_manufacture"] = \

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

elif args["name"] == 'Customers':

args["customer\_name"], args["customer\_surname"], args["customer\_telephone"] = \

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

elif args["name"] == 'Orders':

args["customer\_id"], args["car\_id"], args["stuff\_id"], args["order\_date"] = \

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

elif args["name"] == 'Stuff':

args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"] = \

sys.argv[4], sys.argv[5]

else:

c.v.wrong\_table()

except IndexError:

c.v.argument\_error()

else:

if args["name"] == 'Cars':

c.insert\_cars(args["key"], args["brand\_name"], args["model\_name"],

args["year\_of\_manufacture"])

elif args["name"] == 'Customers':

c.insert\_customers(args["key"], args["customer\_name"], args["customer\_surname"], args["customer\_telephone"])

elif args["name"] == 'Orders':

c.insert\_orders(args["key"], args["customer\_id"], args["car\_id"], args["stuff\_id"], args["order\_date"])

c.insert\_orders(args["key"], args["customer\_id"], args["car\_id"], args["stuff\_id"], args["order\_date"])

elif args["name"] == 'Stuff':

c.insert\_shop(args["key"], args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"])

elif command == 'generate\_randomly':

try:

args = {"name": sys.argv[2], "n": int(sys.argv[3])}

except (IndexError, Exception):

print(Exception, IndexError)

else:

c.generate(args["name"], args["n"])

elif command == 'search\_records':

if len(sys.argv) in [6, 9, 12]:

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:

if len(sys.argv) == 6:

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

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

c.search\_two(args["table1\_name"], args["table2\_name"], args["key1\_name"], args["key2\_name"],

c.v.proceed\_search(search\_num))

elif len(sys.argv) == 9:

args = {"table1\_name": sys.argv[2], "table2\_name": sys.argv[3], "table3\_name": sys.argv[4],

"key1\_name": sys.argv[5], "key2\_name": sys.argv[6], "key3\_name": sys.argv[7],

"key13\_name": sys.argv[8]}

c.search\_three(args["table1\_name"], args["table2\_name"], args["table3\_name"],

args["key1\_name"], args["key2\_name"], args["key3\_name"], args["key13\_name"],

c.v.proceed\_search(search\_num))

elif len(sys.argv) == 12:

args = {"table1\_name": sys.argv[2], "table2\_name": sys.argv[3], "table3\_name": sys.argv[4],

"table4\_name": sys.argv[5],

"key1\_name": sys.argv[6], "key2\_name": sys.argv[7], "key3\_name": sys.argv[8],

"key13\_name": sys.argv[9], "key4\_name": sys.argv[10], "key24\_name": sys.argv[11]}

c.search\_four(args["table1\_name"], args["table2\_name"], args["table3\_name"],

args["table4\_name"],

args["key1\_name"], args["key2\_name"], args["key3\_name"], args["key13\_name"],

args["key4\_name"], args["key24\_name"], c.v.proceed\_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()

Код програмного модулю “validator.py”

import datetime

class Validator:

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

self.error = ''

self.er\_flag = False

def check\_table\_name(self, arg: str):

if arg in ['Cars', 'Customers', 'Orders', 'Stuff']:

return arg

else:

self.er\_flag = True

self.error = f'table {arg} does not exist in the database'

print(self.error)

return False

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

try:

value = int(arg)

except ValueError:

self.er\_flag = True

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

print(self.error)

return 0

else:

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

return value

else:

self.er\_flag = True

self.error = f'{arg} is not existing primary key value'

print(self.error)

return 0

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

if table\_name == 'Cars' and key\_name == 'car\_id' \

or table\_name == 'Customers' and key\_name == 'customer\_id' \

or table\_name == 'Orders' and key\_name == 'order\_id' \

or table\_name == 'Stuff' and key\_name == 'stuff\_id':

return key\_name

else:

self.er\_flag = True

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

print(self.error)

return False

def check\_pk(self, val, count):

try:

value = int(val)

except ValueError:

self.er\_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 == 'Cars' and key in ['car\_id', 'brand\_name', 'model\_name', 'year\_of\_manufacture']:

return True

elif table\_name == 'Customers' and key in ['customer\_id', 'customer\_name', 'customer\_surname', 'customer\_telephone']:

return True

elif table\_name == 'Orders' and key in ['order\_id', 'customer\_id', 'car\_id', 'stuff\_id','order\_date']:

return True

elif table\_name == 'Stuff' and key in ['stuff\_id', 'stuff\_surname', 'stuff\_telephone','date\_start\_working']:

return True

else:

self.er\_flag = True

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

print(self.error)

return False

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

if table\_name == 'Cars':

if key in ['car\_id']:

try:

value = int(val)

except ValueError:

self.er\_flag = True

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

print(self.error)

return False

else:

return True

elif key in ['brand\_name', 'model\_name']:

return True

elif key == 'year\_of\_manufacture':

try:

value = float(val)

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct year\_of\_manufacture value'

print(self.error)

return False

else:

return True

else:

self.er\_flag = True

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

print(self.error)

return False

elif table\_name == 'Customers':

if key in ['customer\_id']:

try:

value = int(val)

except ValueError:

self.er\_flag = True

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

print(self.error)

return False

else:

return True

elif key in ['customer\_name','customer\_surname']:

return True

elif key in ['customer\_telephone']:

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.er\_flag = True

self.error = f'{val} is not correct customer\_telephone value'

print(self.error)

return False

else:

return True

else:

self.er\_flag = True

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

print(self.error)

return False

elif table\_name == 'Orders':

if key in ['order\_id', 'customer\_id', 'car\_id']:

try:

value = int(val)

except ValueError:

self.er\_flag = True

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

print(self.error)

return False

else:

return True

elif key == 'order\_date':

return True

else:

self.er\_flag = True

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

print(self.error)

return False

elif table\_name == 'Stuff':

if key == 'stuff\_id':

try:

value = int(val)

except ValueError:

self.er\_flag = True

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

print(self.error)

return False

else:

return True

elif key in ['stuff\_surname', 'stuff\_telephone','date\_start\_working']:

return True

else:

self.er\_flag = True

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

print(self.error)

return False

Код програмного модулю “view.py”

import datetime

import time

import validator

class View:

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

self.valid = validator.Validator()

def cannot\_delete(self) -> None:

print('this record is connected with another table, deleting will '

'throw error')

def sql\_error(self, e) -> None:

print("[INFO] Error while working with Postgresql", e)

def insertion\_error(self) -> None:

print('Something went wrong (record with such id exists or inappropriate foreign key values)')

def updation\_error(self) -> None:

print('Something went wrong (record with such id does not exist or inappropriate foreign key value)')

def deletion\_error(self) -> None:

print('record with such id does not exist')

def invalid\_interval(self) -> None:

print('invalid interval input')

def print\_time(self, start) -> None:

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

def print\_search(self, result):

print('search result:')

for row in result:

for i in range(0, len(row)):

print(row[i])

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

def print\_cars(self, table):

print('Cars table:')

for row in table:

print('car\_id:', row[0], '\brand\_name:', row[1], '\model\_name:', row[2], '\year\_of\_manufacture:', row[3])

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

def print\_customers(self, table):

print('Customers table:')

for row in table:

print('customer\_id:', row[0], '\customer\_name:', row[1], '\customer\_surname:', row[2], '\customer\_telephone:', row[3])

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

def print\_orders(self, table):

print('Orders table:')

for row in table:

print('order\_id:', row[0], '\customer\_id:', row[1], '\car\_id:', row[2], '\stuff\_id:', row[3], '\order\_date:', row[4])

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

def print\_stuff(self, table):

print('Stuff table:')

for row in table:

print('stuff\_id:', row[0], '\stuff\_surname:', row[1], '\stuff\_surname:', row[2],'\date\_start\_working:', row[3] )

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

def print\_help(self):

print('print\_table - outputs the specified table \n\targument (table\_name) is required')

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

'\targuments (table\_name, key\_name, key\_value) are required')

print('update\_record - updates record with specified id in table\n'

'\tProduct args (table\_name, id\_product, title, price, category, id\_catalog, id\_order)\n'

'\tOrder args (table\_name, id\_order, customer\_name, id\_shop, date)\n'

'\tCatalog args (table\_name, id\_catalog, name, id\_shop, pid\_catalog)\n'

'\tShop args (table\_name, id\_shop, address, name)')

print('insert\_record - inserts record into specified table \n'

'\tProduct args (table\_name, id\_product, title, price, category, id\_catalog, id\_order)\n'

'\tOrder args (table\_name, id\_order, customer\_name, id\_shop, date)\n'

'\tCatalog args (table\_name, id\_catalog, name, id\_shop, pid\_catalog)\n'

'\tShop args (table\_name, id\_shop, address, name)')

print('generate\_randomly - generates n random records in table\n'

'\targuments (table\_name, n) are required')

print('search\_records - search for records in two or more tables using one or more keys \n'

'\targuments (table1\_name, table2\_name, table1\_key, table2\_key) are required, \n'

'\tif you want to perform search in more tables: \n'

'\t(table1\_name, table2\_name, table3\_name, table1\_key, table2\_key, table3\_key, table13\_key) \n'

'\t(table1\_name, table2\_name, table3\_name, table4\_name, table1\_key, table2\_key, table3\_key, table13\_key, '

'table4\_key, table24\_key)')

def proceed\_search(self, search\_num):

search = ''

for i in range(0, search\_num):

while True:

search\_type = input('specify the type of data you want to search for '

'(numeric, string or date): ')

if search\_type == 'numeric' or search\_type == 'string' or search\_type == 'date':

break

key = input('specify the name of key by which you`d like to perform search '

'in form: table\_number.key\_name: ')

if search\_type == 'numeric':

a = input('specify the left end of search interval: ')

b = input('specify the right end of search interval: ')

if search == '':

search = self.numeric\_search(a, b, key)

else:

search += ' and ' + self.numeric\_search(a, b, key)

elif search\_type == 'date':

data = input('specify the left end of search interval '

'in form: year.month.day.hour.minute.second: ')

datb = input('specify the right end of search interval '

'in form: year.month.day.hour.minute.second: ')

if search == '':

search = self.date\_search(data, datb, key)

else:

search += ' and ' + self.date\_search(data, datb, key)

elif search\_type == 'string':

string = input('specify the string you`d like to search for: ')

if search == '':

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

else:

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

return search

def numeric\_search(self, a: str, b: str, key: str):

try:

a, b = int(a), int(b)

except ValueError:

self.invalid\_interval()

else:

return f"{a}<{key} and {key}<{b}"

def date\_search(self, a: str, b: str, key: str):

try:

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

brr = [int(x) for x in b.split(sep='.')]

except Exception:

print(Exception)

self.invalid\_interval()

else:

return f"{key} BETWEEN \'{datetime.datetime(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5])}\' " \

f"AND \'{datetime.datetime(brr[0], brr[1], brr[2], brr[3], brr[4], brr[5])}\'"

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

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

def get\_search\_num(self):

return input('specify the number of attributes you`d like to search by: ')

def invalid\_search\_num(self):

print('should be number different from 0')

def argument\_error(self):

print('no required arguments specified')

def wrong\_table(self):

print('wrong table name')

def no\_command(self):

print('no command name specified, type help to see possible commands')

def wrong\_command(self):

print('unknown command name, type help to see possible commands')

Код програмного модулю “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 printf(self, table\_name):

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

if t\_name:

if t\_name == 'Cars':

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

elif t\_name == 'Customers':

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

elif t\_name == 'Orders':

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

elif t\_name == 'Stuff':

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

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

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

k\_name = self.v.valid.check\_pk\_name(table\_name, key\_name)

if t\_name and k\_name:

count = self.m.find(t\_name, k\_name, value)

k\_val = self.v.valid.check\_pk(value, count)

if k\_val:

if t\_name == 'Cars' or t\_name == 'Customers' or t\_name == 'Stuff':

count\_p = self.m.find('Orders', k\_name, value)[0]

if count\_p:

self.v.cannot\_delete()

else:

try:

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

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

# elif t\_name == 'Shop':

# count\_c = self.m.find('Catalog', k\_name, value)[0]

# count\_o = self.m.find('Order', k\_name, value)[0]

# if count\_c or count\_o:

# self.v.cannot\_delete()

# else:

# try:

# self.m.delete\_data(table\_name, key\_name, k\_val)

# except (Exception, Error) as \_ex:

# self.v.sql\_error(\_ex)

else:

try:

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

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.deletion\_error()

def update\_customer(self, key: str, customer\_name: str, customer\_surname: str, customer\_telephone: str):

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', key):

count\_p = self.m.find('Customers', 'customer\_id', int(key))

# p\_val = self.v.valid.check\_pk(key, count\_p)

try:

self.m.update\_data\_product(p\_val, customer\_surname, customer\_name,

customer\_telephone)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def update\_cars(self, key: str, brand\_name: str, model\_name: str, year\_of\_manufacture: int):

if self.v.valid.check\_possible\_keys('Cars', 'car\_id', key):

count\_p = self.m.find('Cars', 'car\_id', int(key))

# p\_val = self.v.valid.check\_pk(key, count\_p)

try:

# arr = [int(x) for x in date.split(sep='.')]

self.m.update\_data\_order(brand\_name,model\_name, year\_of\_manufacture)

# 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.updation\_error()

def update\_orders(self, key: str, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str):

if self.v.valid.check\_possible\_keys('Orders', 'order\_id', key):

count\_s = self.m.find('Orders', 'order\_id', int(key))

# s\_val = self.v.valid.check\_pk(key, count\_s)

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', customer\_id):

count\_c = self.m.find('Customers', 'customer\_id', int(customer\_id))

c\_val = self.v.valid.check\_pk(customer\_id, count\_c)

if self.v.valid.check\_possible\_keys('Cars', 'car\_id', car\_id):

count\_pc = self.m.find('Cars', 'car\_id', int(car\_id))

pc\_val = self.v.valid.check\_pk(car\_id, count\_pc)

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', stuff\_id):

count\_p = self.m.find('Stuff', 'stuff\_id', int(stuff\_id))

p\_val = self.v.valid.check\_pk(stuff\_id, count\_p)

if s\_val and c\_val and pc\_val and p\_val:

try:

self.m.update\_data\_catalog(s\_val,c\_val, pc\_val, p\_val, order\_date)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def update\_stuff(self, key: str, stuff\_surname: str, date\_start\_working:str ,stuff\_telephone:str):

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', key):

count\_s = self.m.find('Stuff', 'stuff\_id', int(key))

# s\_val = self.v.valid.check\_pk(key, count\_s)

if s\_val:

try:

self.m.update\_data\_shop(s\_val, stuff\_surname , stuff\_telephone, date\_start\_working)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def insert\_customer(self, key: str, customer\_name: str, customer\_surname: str, customer\_telephone: str):

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', key):

count\_p = self.m.find('Customers', 'customer\_id', int(key))[0]

if (not count\_p or count\_p == (0,)) \

and self.v.valid.check\_possible\_keys('Customers', 'customer\_id', key):

try:

self.m.insert\_data\_product(int(key), customer\_name , customer\_surname ,

customer\_telephone)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.insertion\_error()

def insert\_cars(self, key: str, brand\_name: str, model\_name: str, year\_of\_manufacture: int):

if self.v.valid.check\_possible\_keys('Cars ', 'car\_id', key):

count\_s = self.m.find('Cars', 'car\_id', int(key))

# s\_val = self.v.valid.check\_pk(key, count\_s)

# if s\_val and self.v.valid.check\_possible\_keys('Cars', 'car\_id', key) \

# and self.v.valid.check\_possible\_keys('Cars', 'date', date):

try:

# arr = [int(x) for x in date.split(sep='.')]

self.m.insert\_data\_order(int(key),brand\_name ,model\_name, year\_of\_manufacture)

# 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 insert\_orders(self, key: str, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str):

if self.v.valid.check\_possible\_keys('Orders', 'order\_id', key):

count\_s = self.m.find('Orders', 'order\_id', int(key))

s\_val = self.v.valid.check\_pk(key, count\_s)

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', customer\_id):

count\_c = self.m.find('Customers', 'customer\_id', int(customer\_id))[0]

c\_val = self.v.valid.check\_pk(customer\_id, count\_c)

if self.v.valid.check\_possible\_keys('Cars', 'car\_id', car\_id):

count\_pc = self.m.find('Cars', 'car\_id', int(car\_id))

pc\_val = self.v.valid.check\_pk(car\_id, count\_pc)

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', stuff\_id):

count\_p = self.m.find('Stuff', 'stuff\_id', int(stuff\_id))

p\_val = self.v.valid.check\_pk(stuff\_id, count\_p)

if (not count\_c or count\_c == (0,)) and s\_val and c\_val and pc\_val and p\_val \

and self.v.valid.check\_possible\_keys('Orders', 'order\_id', key):

# try:

# self.m.insert\_data\_catalog(int(key) , s\_val, pc\_val,p\_val,order\_date)

# except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.insertion\_error()

def insert\_shop(self, key: str, stuff\_surname: str, date\_start\_working:str ,stuff\_telephone:str):

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', key):

count\_s = self.m.find('Stuff', 'stuff\_id', int(key))[0]

if (not count\_s or count\_s == (0,)) and self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', key):

try:

self.m.insert\_data\_shop(int(key), stuff\_surname,date\_start\_working, stuff\_telephone )

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 == 'Cars':

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

elif t\_name == 'Customers':

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

elif t\_name == 'Orders':

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

elif t\_name == 'Stuff':

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

def search\_two(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.print\_time(start\_time)

self.v.print\_search(result)

def search\_three(self, table1\_name: str, table2\_name: str, table3\_name: str,

table1\_key: str, table2\_key: str, table3\_key: str, table13\_key: str,

search: str):

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

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

t3\_n = self.v.valid.check\_table\_name(table3\_name)

if t1\_n and self.v.valid.check\_key\_names(t1\_n, table1\_key) and self.v.valid.check\_key\_names(t1\_n, table13\_key) \

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

and t3\_n and self.v.valid.check\_key\_names(t3\_n, table3\_key) \

and self.v.valid.check\_key\_names(t3\_n, table13\_key):

start\_time = time.time()

result = self.m.search\_data\_three\_tables(table1\_name, table2\_name, table3\_name,

table1\_key, table2\_key, table3\_key, table13\_key,

search)

self.v.print\_time(start\_time)

self.v.print\_search(result)

def search\_four(self, table1\_name: str, table2\_name: str, table3\_name: str, table4\_name: str,

table1\_key: str, table2\_key: str, table3\_key: str, table13\_key: str,

table4\_key: str, table24\_key: str,

search: str):

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

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

t3\_n = self.v.valid.check\_table\_name(table3\_name)

t4\_n = self.v.valid.check\_table\_name(table2\_name)

if t1\_n and self.v.valid.check\_key\_names(t1\_n, table1\_key) and self.v.valid.check\_key\_names(t1\_n, table13\_key) \

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

and self.v.valid.check\_key\_names(t2\_n, table24\_key) \

and t3\_n and self.v.valid.check\_key\_names(t3\_n, table3\_key) \

and self.v.valid.check\_key\_names(t3\_n, table13\_key) \

and t4\_n and self.v.valid.check\_key\_names(t4\_n, table4\_key) \

and self.v.valid.check\_key\_names(t4\_n, table24\_key):

start\_time = time.time()

result = self.m.search\_data\_all\_tables(table1\_name, table2\_name, table3\_name, table4\_name,

table1\_key, table2\_key, table3\_key, table13\_key,

table4\_key, table24\_key,

search)

self.v.print\_time(start\_time)

self.v.print\_search(result)