

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

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

Виконав: студент 3 курсу

ФПМ групи КВ-82

Гольовський А. А.

Перевірив: Павловський В.І.

Київ – 2020

## **Лабораторна робота №2.**

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

***Мета роботи:*** здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

***Загальне завдання*:**

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).

###### ***Посилання на код в GitHub:***

***https://github.com/Adniry/DB/tree/master/lab2***

***Зміст звіту:***

1. *Логічна модель бази даних;*
2. *Засоби програмування додатку, спосіб доступу до БД;*
3. *Шаблон проектування додатку;*
4. *Функціональні можливості додатку;*
5. *Структура програми додатку;*
6. *Ілюстрація роботи додатку та відповіді на питання.*
7. *Код програми*

**Логічна модель бази даних**

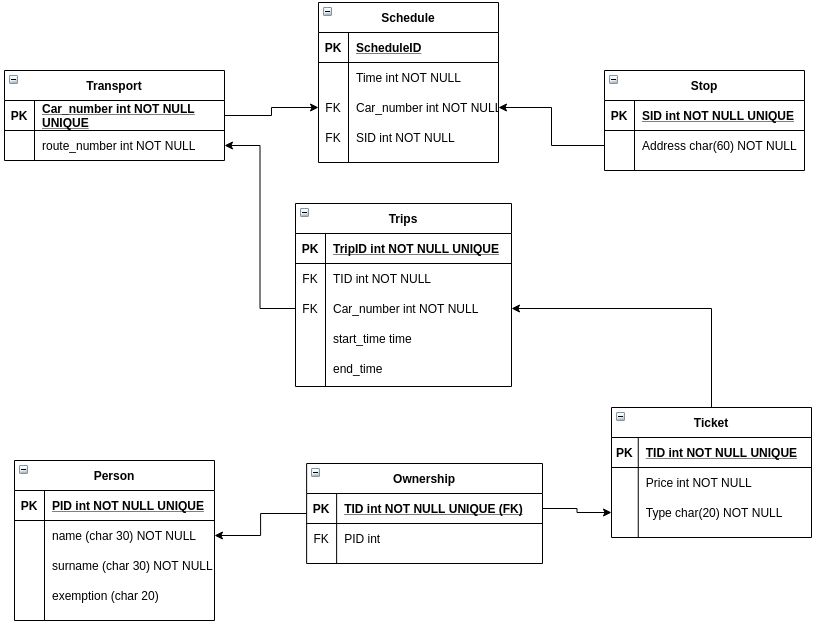
Опис бази даних:

База даних репрезентує предметну область «Міський громадський транспорт». Основні сутності:

* Людина (пасажир)
* Транспорт
* Зупинка
* Квиток

В результаті нормалізації було додано ще 3 таблиці, текстове представлення кінцевого варіанту бази даних наведено в таблиці:

|  |  |  |
| --- | --- | --- |
| **Відношення** | **Атрибут** | **Тип** |
| Відношення “Person” містить інформацію про людину-користувача транспортом. | PID – унікальний номер особи в БД  Name – ім’я особи. Не допускає NULL.  Surname – прізвище особи. Не допускає NULL.  Exemption – тип пільги особи. Не допускає NULL | Числовий, SERIAL PK  Текстовий(30)  Текстовий(30)  Текстовий(20) |
| Відношення ”Ticket” містить інформацію про квитки на транспорт | TicketID – унікальний номер квитка в БД.  Operation\_time – час дії квитка. Не допускає NULL.  Price – ціна квитка. Не допускає NULL. | Числовий, SERIAL PK  Часова мітка  Грошовий |
| Відношення ”Transport” містить інформацію про одиниці транспорту в парку | Car\_number – унікальний номер одиниці транспорту  Route\_number – номер маршруту. Не допускає NULL | Числовий, SERIAL PK  Числовий |
| Відношення ”Stop” містить інформацію про зупинки в місті | SID - унікальний номер зупинки в БД  Address – адреса зупинки. Не допускає NULL | Числовий, SERIAL PK  Текстовий(50) |
| Відношення “Schedule” містить інформацію про час зупинки кожної одиниці транспорту на кожній зупинці її маршруту. | ScheduleID – унікальний номер запису розкладу.  Time – час зупинки. Не допускає NULL  SID – ідентифікатор зупинки  Car\_number – ідентифікатор транспортної одиниці | Числовий SERIAL PK  Часова мітка  Числовий FK  Числовий FK |
| Відношення “Trip” містить інформацію про кожну поїздку | TripID – унікальний номер поїздки  Car\_number – ідентифікатор одиниці транспорту, на якій було здійснено поїздку  TicketID – ідентифікатор квитка, за допомогою якого було здійснено поїздку  Start time – час посадки  End time – час висадки | Числовий SERIAL PK  Числовий FK  Числовий FK  Часова мітка  Часова мітка |
| Відношення “Ownership” містить інформацію про те, кому який квиток належить | TID — ідентифікатор квитка  PID — ідентифікатор особи-власника квитка | Числовий PK FK  Числовий FK |

Рисунок 1 — Логічна модель предметної області “Міський транспорт”.

**Засоби програмування і спосіб доступу до БД**

Мова програмування: python;

Середовище програмування (IDE): PyCharm;

Засіб для розробки графічного інтерфейсу: QtDesigner, pyuic5;

Спосіб доступу до БД: бібліотека psycopg.

**Шаблон проектування додатку:**

Додаток спроектовано згідно шаблону модель-подання-контроллер. Основні модулі:

* Модель (файл model.py) — Запити до бази даних та бізнес-логіка;
* Подання (файли trip\_d.py та view.py) — Графічний інтерфейс (згенеровано за допомогою pyuic)
* Контроллер (файл controller.py) — Зв’язок графічного інтерфейсу і моделі.

**Функціональні можливості додатку:**

Додаток забезпечує такі можливості:

* Видалення записів у зазначеній таблиці за заданим параметром;
* Запис/зміна записів у зазначеній таблиці за заданими парметрами;
* Пошук записів у зазначеній таблиці за пошуковою фразою;
* Генерація рандомізованих даних у всі таблиці;
* Пошук конкретної поїздки за часом початку цієї поїздки (результуючі рядки формуються з трьох таблиць);

При здійсненні запису/зміни/видаленні даних виконується валідація даних і перевірка на безпечність видалення шляхом перехоплення помилок SQL і виведення їх у вікні програми червоним шрифтом.

При генерації рандомізованих даних забезпечується відповідність даних обмеженням Foreign Key.

**Структура програми додатку**

Модуль model.py

1. Клас Database — клас, об’єкт якого відображає базу даних, з якою встановлено з’єднання (для різних баз даних окреме з’єднання в окремому екземплярі класу). Методи:
   * \_\_init\_\_ - конструктор
   * config — встановлення з’єднання з БД з параметрами, зчитаними з файлу config.ini
   * get\_request — формує запит на отримання даних
   * delete\_request — формує запит на видалення даних
   * insert\_request — формує запит на внесення даних
   * update\_request — формує запит на оновлення даних
   * request — відправка запиту, сформованого котроюсь з попередніх чотирьох функцій до БД
   * getValues — відправка запиту для отримання детальної інформації про всі поїздки в заданому часовому інтервалі
   * generate\_values — формування і відправка запиту на генерацію рандомізованих даних
   * gen\_values — обгортка для попередньої функції (для виловлення помилок)

Модуль Controller:

1. Клас TripController — клас, об’єкт якого виконує виклики методів з модуля model.py на основі взаємодії користувача з графічним інтерфейсом діалогового вікна пошуку поїздок в заданому часовому інтервалі. Єдиний метод класу виконує власне запит до бази даних.
2. Клас Controller — клас, об’єкт якого виконує виклики методів з модуля model.py на основі взаємодії користувача з графічним інтерфейсом взаємодії користувача з графічним інтерфейсом основного вікна додатку.

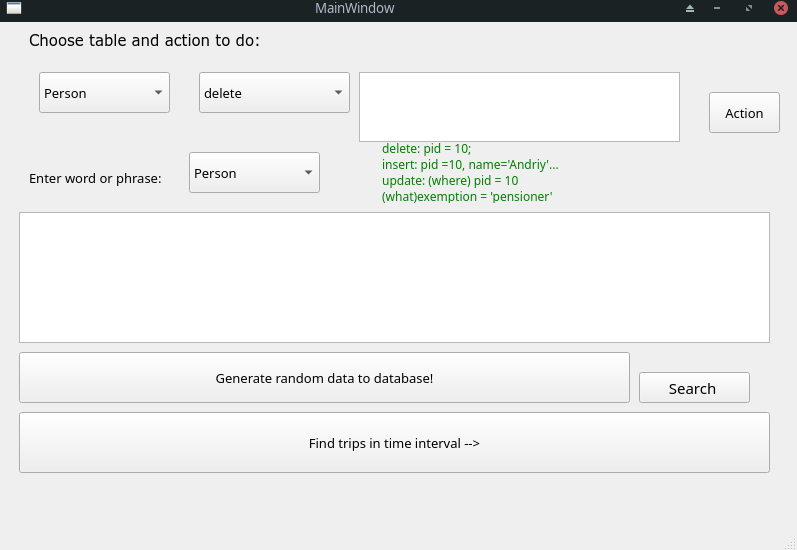
Методи:

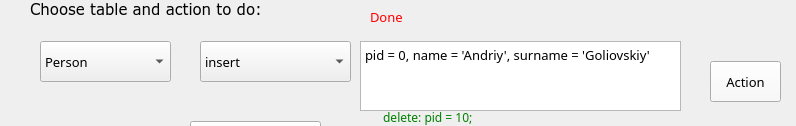
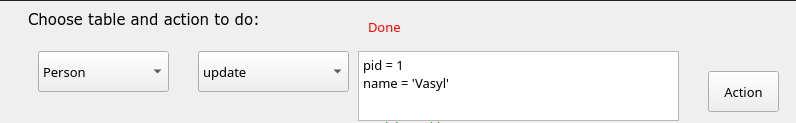
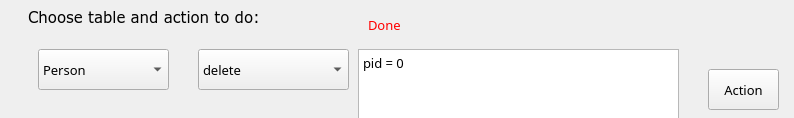
* + \_\_init\_\_ - конструктор, в якому виконуються виклики методів ініціалізації графічного інтерфейсу та з’єднання з БД
  + gen\_values\_call — метод який викликає метод генерації рандомізованих даних класу Database модуля model.py
  + saveInfo — метод який викликає методи виклику SQL-запитів модуля model.py на основі даних, введених у відповідні поля графічного інтерфейсу
  + showDialog — ініціалізація діалогового вікна
  + full\_str\_call — метод, що викликає відповідний метод класу Database, що виконує пошук записів за заданою таблицею та критеріями.

Модуль View та Trip\_dial — модулі, що містять програмну репрезентацію графічного інтерфейсу, створеного в QtDesigner, згенеровану за допомогою pyuic5.

**Ілюстрація роботи додатку та відповіді на питання**

Головне меню додатку:

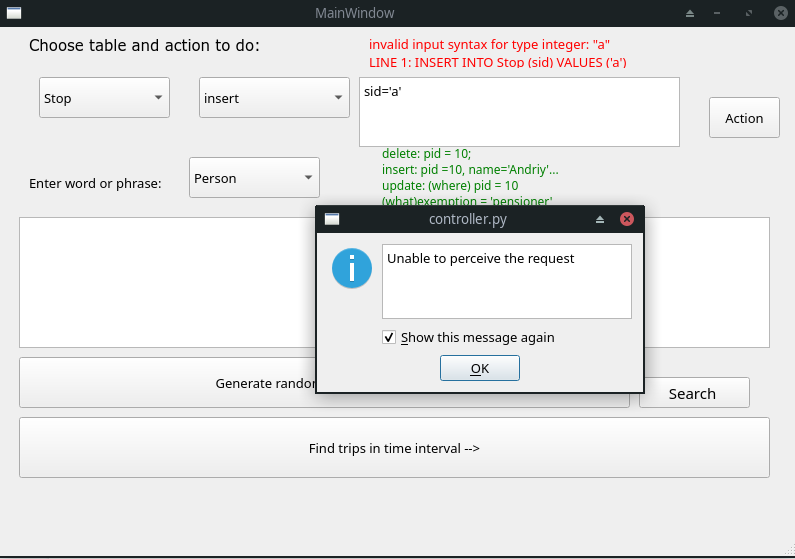
 Формати запитів:

* На введення:
* На видалення:
* На оновлення:

Фрагмент коду, відповідального за перехоплення помилок:

try:  
 cursor = self.conn.cursor()  
 print(req)  
 cursor.execute(req)  
 cursor.execute("SELECT \* FROM person LIMIT 1;")  
 self.colnames = [desc[0] for desc in cursor.description]  
 self.conn.commit()  
 return True  
except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:  
 print(error)  
 self.error = error # поле класу, яке містить повідомлення про помилку  
 self.conn.rollback()  
 return False

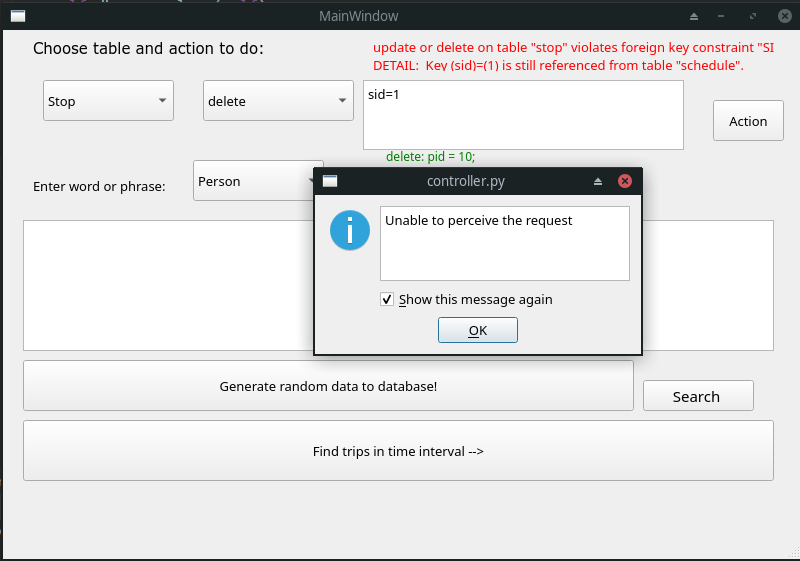
Приклад обробки помилок, які можуть виникнути на стороні сереверу (у тому числі синтаксичних):



Контроль вилучення/оновлення даних здійснюється за замовчуванням — заборона виконувати дію, якщо є обмеження:

ON UPDATE NO ACTION

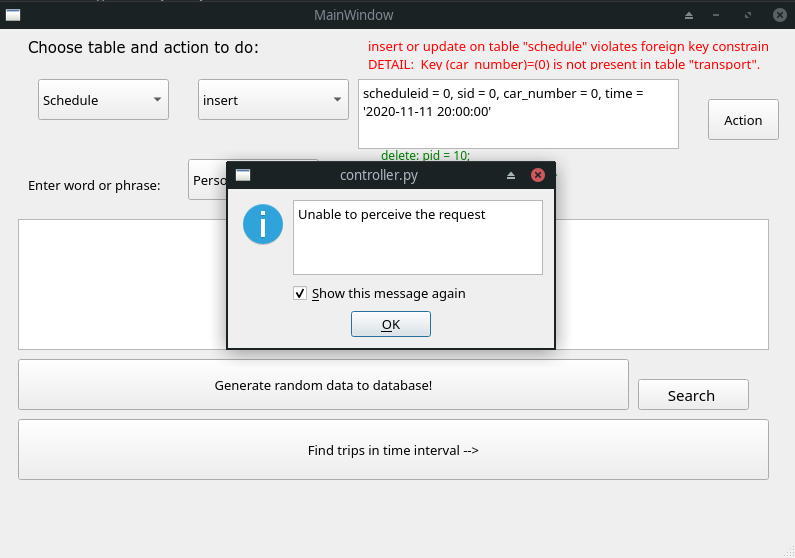
ON DELETE NO ACTION

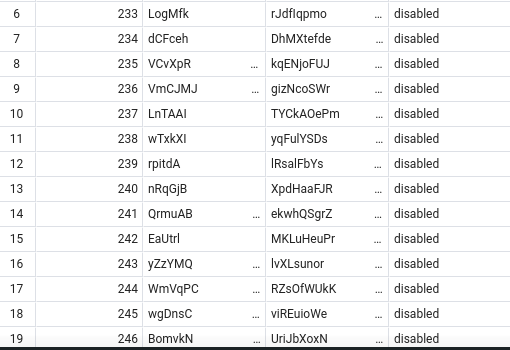
****

Код, відповідальний за наявність відповідного рядка при внесенні даних у підлеглу:

|  |  |
| --- | --- |
| try: |  |
|  | cursor = self.conn.cursor() |
|  | print(req) |
|  | cursor.execute(req) |
|  | cursor.execute("SELECT \* FROM person LIMIT 1;") |
|  | self.colnames = [desc[0] for desc in cursor.description] |
|  | self.conn.commit() |
|  | return True |
|  | except(Exception, ps.DatabaseError, ps.ProgrammingError) as error: |
|  | print(error) |
|  | self.error = error |
|  | self.conn.rollback() |
|  | return False |

Приклад перехоплення помилки при внесенні даних у підлеглу таблицю при відсутності відповідного рядка у батьківській:

****

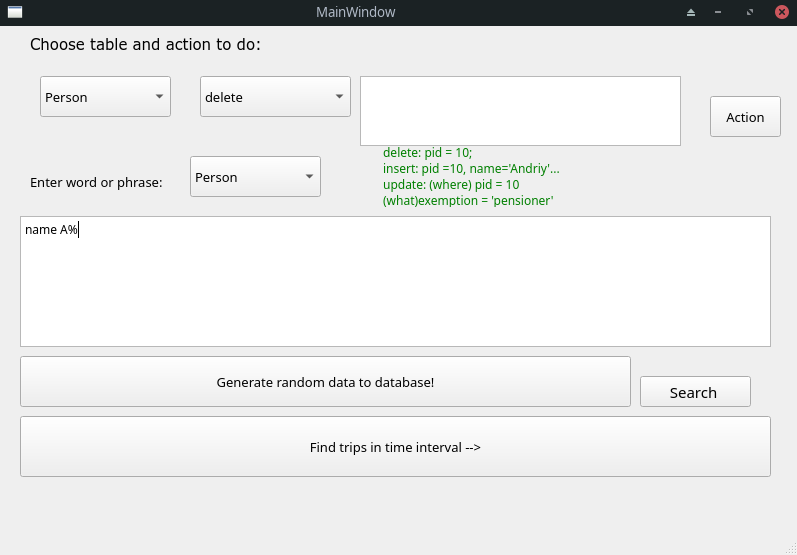
Згенеровані рандомізовані дані:

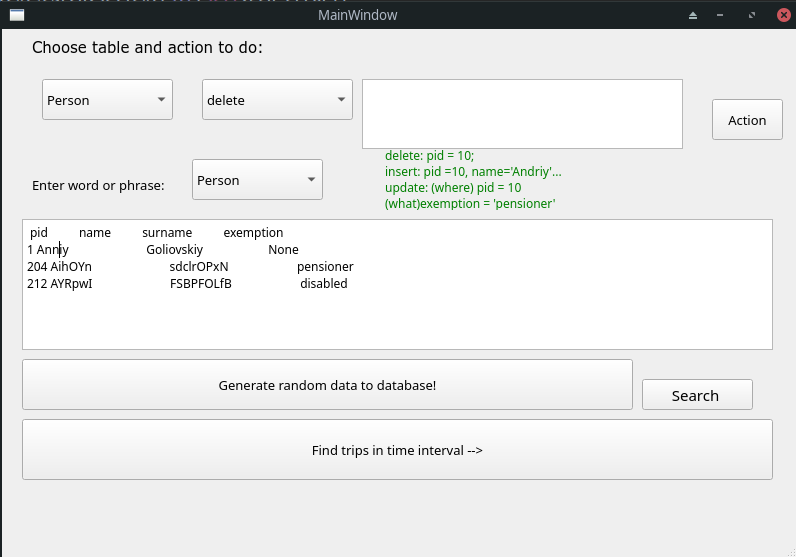
Копії SQL-запитів, які генерують рандомізовані дані (в якості аргументів функцій замість {0}, {1} і так далі передаються числа, які задають інтервал випадкових значень):

"INSERT INTO person (pid, name, surname, exemption) VALUES (generate\_series({0}, {1}), random\_string(6), random\_string(9), 'pensioner');"  
"INSERT INTO Person (pid, name, surname, exemption) VALUES (generate\_series({0}, {1}), random\_string (6), random\_string (9), 'disabled');"  
  
"INSERT INTO stop (sid, Address) VALUES (generate\_series({0}, {1}), random\_string(15));"  
  
"INSERT INTO ticket (TID, Price, Operation\_time) VALUES (generate\_series({0},{1}), generate\_series({2}, {3}), generate\_series('{4}'::timestamp, '{5}','24 hours'));"  
  
"INSERT INTO Transport (Car\_number, Route\_number) VALUES (generate\_series({0},{1}), generate\_series({0}, {1}));"  
  
"INSERT INTO schedule (scheduleid, car\_number, sid, time) VALUES (generate\_series({0}, {1}), generate\_series({0},{1}), generate\_series({0}, {1}), generate\_series('{2}'::timestamp, '{3}','24 hours'));"

"INSERT INTO ownership (tid, pid) VALUES (generate\_series({0}, {1}), generate\_series({0},{1}));"  
  
"INSERT INTO trip (tripid, car\_number, tid, start\_time, end\_time) VALUES (generate\_series({0}, {1}),generate\_series({0}, {1}),generate\_series({0}, {1}),generate\_series('{2}'::timestamp, '{3}','10 minutes'), generate\_series('{4}'::timestamp, '{5}','10 minutes'));"

Приклад пошукового запиту:

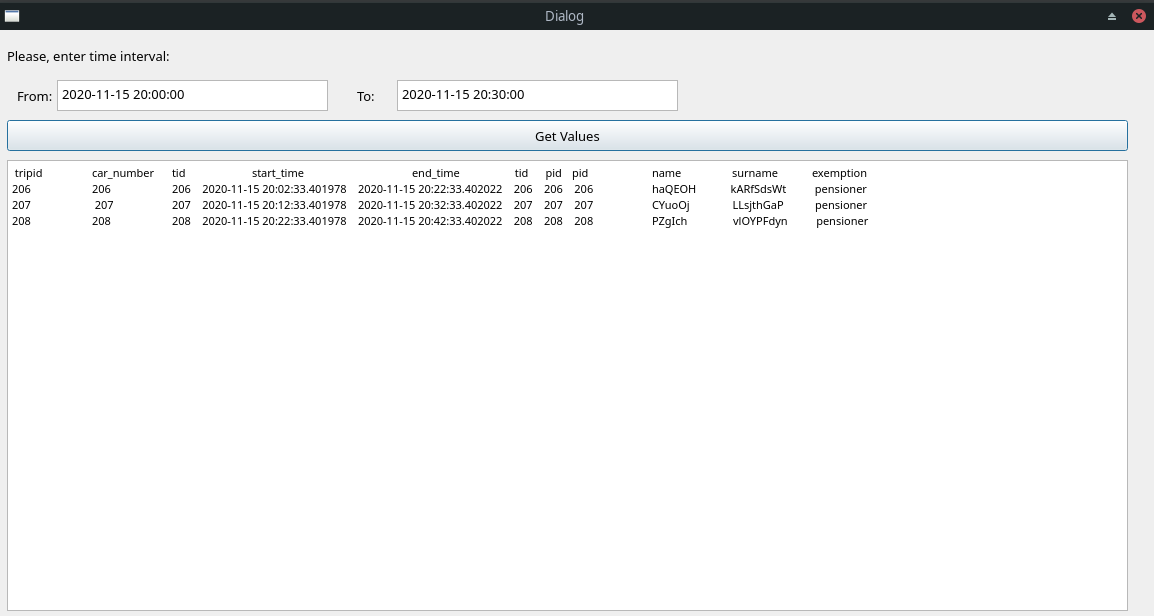


 Результат пошукового запиту:

SQL-запит для пошуку:

"SELECT \* FROM {1} WHERE {2} LIKE '{3}';"

Приклад пошуку поїздок за часом:



**Код програми**

Файл model.py:

import psycopg2 as ps  
from configparser import ConfigParser  
import datetime  
from datetime import timedelta  
import json  
  
class Database:  
  
 def config(self, filename='config.ini', section='postgresql'):  
 parser = ConfigParser()  
 parser.read(filename)  
 db = {}  
 if parser.has\_section(section):  
 params = parser.items(section)  
 for param in params:  
 db[param[0]] = param[1]  
 else:  
 raise Exception('Section {0} not found in the {1} file'.format(section, filename))  
 return db  
  
 def get\_request(self, req):  
 try:  
 cursor = self.conn.cursor()  
 cursor.execute(req)  
 self.conn.commit()  
 self.colnames = [desc[0] for desc in cursor.description]  
 return cursor.fetchall()  
 except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:  
 self.conn.rollback()  
 self.gen\_error = error  
 self.erFlag = True  
 print(error)  
 return False  
  
 def \_\_init\_\_(self):  
 self.conn = None  
 self.error = ''  
 self.gen\_error = ''  
 self.erFlag = False  
 self.Gen = True  
 self.colnames = list()  
 try:  
 params = self.config('config.ini')  
 self.conn = ps.connect(\*\*params)  
  
 except(Exception, ps.DatabaseError) as error:  
 print(error)  
  
 def delete\_request(self, action, text):  
 return self.request("DELETE FROM {0} WHERE {1};".format(action, text))  
  
 def insert\_request(self, table, text):  
 enter = [list.split('=') for list in text.split(',')] #devided values  
 values = arguments = str()  
 for word in enter:  
 arguments += word[0] + ','  
 values += word[1] + ','  
 arguments = arguments[:-1]  
 values = values[:-1]  
 return self.request("INSERT INTO {0} ({1}) VALUES ({2}) ".format(table, arguments, values))  
  
 def request(self, req):  
 try:  
 cursor = self.conn.cursor()  
 print(req)  
 cursor.execute(req)  
 cursor.execute("SELECT \* FROM person LIMIT 1;")  
 self.colnames = [desc[0] for desc in cursor.description]  
 self.conn.commit()  
 return True  
 except(Exception, ps.DatabaseError, ps.ProgrammingError) as error:  
 print(error)  
 self.error = error # поле класу, яке містить повідомлення про помилку  
 self.conn.rollback()  
 return False  
  
 def getValues(self, Trip):  
 from\_range = Trip.textEdit.toPlainText()  
 to\_range = Trip.textEdit\_2.toPlainText()  
 req = "SELECT \* FROM (SELECT \* FROM (SELECT \* FROM trip RIGHT JOIN ownership ON trip.tid = ownership.tid) AS result WHERE (start\_time BETWEEN '{0}' AND '{1}')) AS result INNER JOIN person ON result.pid = person.pid;".format(from\_range, to\_range)  
 result = self.get\_request(req)  
 r\_str = ""  
 for i in result:  
 for st in i:  
 r\_str += str(st) + " "  
 r\_str += "\n"  
 for word in self.colnames:  
 Trip.columns += word + "\t"  
 Trip.columns += '\n'  
 Trip.plainTextEdit.setPlainText(Trip.columns + r\_str)  
  
 def update\_request(self, table, text):  
 property = text.split('\n')  
 print(property)  
 print("UPDATE {0} SET {1} WHERE {2}".format(table, property[1], property[0]))  
 return self.request("UPDATE {0} SET {1} WHERE {2}".format(table, property[1], property[0]))  
  
 def generate\_values(self):  
 with open('data.json', 'r+') as f:  
 data = json.load(f)  
 self.Gen = False  
 start\_last\_number = data['last\_number']+1  
 start\_estimated\_value = data['estimated\_value']+1  
 amount = 20  
 self.Gen= self.request("INSERT INTO person (pid, name, surname, exemption) VALUES (generate\_series({0}, {1}), random\_string(6), random\_string(9), 'pensioner');".format(start\_estimated\_value, start\_estimated\_value+amount))  
 self.Gen = self.request("INSERT INTO Person (pid, name, surname, exemption) VALUES (generate\_series({0}, {1}), random\_string (6), random\_string (9), 'disabled');".format(start\_estimated\_value+amount+1, start\_estimated\_value+2\*amount))  
  
 self.Gen = self.request("INSERT INTO stop (sid, Address) VALUES (generate\_series({0}, {1}), random\_string(15));".format(start\_estimated\_value, start\_estimated\_value+amount))  
  
 self.Gen = self.request("INSERT INTO ticket (TID, Price, Operation\_time) VALUES (generate\_series({0},{1}), generate\_series({2}, {3}), generate\_series('{4}'::timestamp, '{5}','24 hours'));".format(start\_estimated\_value, start\_estimated\_value+amount, 0, 20, str(datetime.datetime.now()), str( datetime.datetime.now() + timedelta(days=amount))))  
  
 self.Gen = self.request("INSERT INTO Transport (Car\_number, Route\_number) VALUES (generate\_series({0},{1}), generate\_series({0}, {1}));".format(start\_estimated\_value, start\_estimated\_value+amount))  
  
 self.Gen = self.request("INSERT INTO schedule (scheduleid, car\_number, sid, time) VALUES (generate\_series({0}, {1}), generate\_series({0},{1}), generate\_series({0}, {1}), generate\_series('{2}'::timestamp, '{3}','24 hours'));".format(start\_estimated\_value, start\_estimated\_value+amount, str(datetime.datetime.now()), str( datetime.datetime.now() + timedelta(days=amount))))  
  
 self.Gen = self.request("INSERT INTO ownership (tid, pid) VALUES (generate\_series({0}, {1}), generate\_series({0},{1}));".format(start\_estimated\_value, start\_estimated\_value+amount))  
  
 self.Gen = self.request("INSERT INTO trip (tripid, car\_number, tid, start\_time, end\_time) VALUES (generate\_series({0}, {1}),generate\_series({0}, {1}),generate\_series({0}, {1}),generate\_series('{2}'::timestamp, '{3}','10 minutes'), generate\_series('{4}'::timestamp, '{5}','10 minutes'));".format(start\_estimated\_value,  
 start\_estimated\_value+amount, str(datetime.datetime.now()), str( datetime.datetime.now() + timedelta(minutes=amount\*10)),  
 str( datetime.datetime.now() + timedelta(minutes=20)), str( datetime.datetime.now() + timedelta(minutes=20+amount\*10))))  
 print(start\_estimated\_value)  
  
 data = {'last\_number': start\_last\_number + amount, 'estimated\_value': start\_estimated\_value + amount}  
 with open('data.json', 'w+') as f:  
 json.dump(data, f)  
  
 def gen\_values(self, Controller):  
 print(Controller)  
 self.generate\_values()  
 if self.Gen:  
 Controller.gen\_label.setText('Done!')  
 else:  
 Controller.gen\_label.setText('Error while generating!')  
  
 def requestFormat(self, comboTable, comboAction, textAction, Controller):  
 Controller.gen\_label.setText('')  
  
 if comboAction == 'delete':  
 Controller.Flag = self.delete\_request(comboTable, textAction)  
 if not Controller.Flag:  
 Controller.error.setText(str(self.error))  
 else:  
 Controller.error.setText('Done')  
 elif comboAction == 'insert':  
 self.Flag = self.insert\_request(comboTable, textAction)  
 if not self.Flag:  
 Controller.error.setText(str(self.error))  
 else:  
 Controller.error.setText('Done')  
 elif comboAction == 'update':  
 self.Flag = self.update\_request(comboTable, textAction)  
 if not self.Flag:  
 Controller.error.setText(str(self.error))  
 else:  
 Controller.error.setText('Done')  
  
 def full\_string(self, Controller):  
 self.columns = str()  
 Controller.full\_text = Controller.textSearch.toPlainText().split(' ')  
 Controller.full\_search\_table = Controller.full\_text\_box.currentText() # from which table to search  
 print(Controller.full\_text)  
 print(Controller.full\_search\_table)  
 if len(Controller.full\_text) == 1:  
 Controller.textSearch.setText('Wrong entering')  
 return  
 req = ''  
 temp = ''  
 req = "SELECT \* FROM {1} WHERE {2} LIKE '{3}';".format(Controller.full\_text[0], Controller.full\_search\_table, Controller.full\_text[0], Controller.full\_text[1])  
 name = self.get\_request(req)  
 if len(Controller.full\_text) == 1:  
 Controller.textSearch.setText('Wrong entering')  
 return  
 if self.erFlag:  
 Controller.textSearch.setText(str(self.gen\_error))  
 return  
 for word in name:  
 for i in word:  
 temp += str(i) + ' '  
 temp+= '\n'  
 for word in self.colnames:  
 Controller.columns += word + " "  
 Controller.columns += '\n'  
 Controller.textSearch.setText(Controller.columns + temp)

Файл controller.py

from view import \*  
from model import \*  
from trip\_d import \*  
import sys  
  
  
  
class TripController( Ui\_Dialog):  
 def \_\_init\_\_(self, MainWindow,db):  
 super().\_\_init\_\_(MainWindow)  
 self.db = db  
 self.columns = ' '  
 self.pushButton.clicked.connect(self.get\_values\_call)  
   
 def get\_values\_call(self):  
 self.db.getValues(self)  
  
class Controller(Ui\_Database):  
   
 def gen\_values\_call(self):  
 self.db.gen\_values(self)  
   
  
 def \_\_init\_\_(self, MainWindow):  
 self.ui = Ui\_Database(MainWindow)  
 self.db = Database()  
 super().\_\_init\_\_(MainWindow)  
 self.gen\_label.setText('')  
  
 self.pushButton.clicked.connect(self.saveInfo)  
 self.genData.clicked.connect(self.gen\_values\_call)  
 self.pushButton\_2.clicked.connect(self.showDialog)  
 #self.search.clicked.connect(self.full\_text\_search\_call)  
 self.pushButton\_3.clicked.connect(self.full\_str\_call)  
 self.comboTable = None  
 self.comboAction = None  
 self.textAction = None  
 self.columns = ' '  
 self.full\_text = ''  
 self.full\_search\_table = ''  
 self.Flag = True  
 self.window = QtWidgets.QDialog(MainWindow)  
 self.ui = TripController(self.window, self.db)  
  
 def saveInfo(self):  
 self.comboAction = self.action.currentText()  
 self.comboTable = self.table.currentText()  
 self.textAction = self.textEdit.toPlainText()  
 self.db.requestFormat(self.comboTable, self.comboAction, self.textAction, self)  
 if not self.Flag:  
 self.error\_dialog = QtWidgets.QErrorMessage()  
 self.error\_dialog.showMessage('Unable to perceive the request')  
 def showDialog(self):  
 self.window.show()  
  
  
 def full\_str\_call(self):  
 self.db.full\_string(self)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QtWidgets.QApplication(sys.argv)  
 MainWindow = QtWidgets.QMainWindow()  
 MainWindow.show()  
 cntr = Controller(MainWindow)  
  
 sys.exit(app.exec\_())

Файл view.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'ui.ui'  
#  
# Created by: PyQt5 UI code generator 5.15.1  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic5 is  
# run again. Do not edit this file unless you know what you are doing.  
  
  
from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_Database(object):  
 def \_\_init\_\_(self, Database):  
 Database.setObjectName("Database")  
 Database.resize(798, 529)  
 self.centralwidget = QtWidgets.QWidget(Database)  
 self.centralwidget.setObjectName("centralwidget")  
 self.pushButton = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton.setGeometry(QtCore.QRect(710, 70, 71, 41))  
 self.pushButton.setObjectName("pushButton")  
 self.table = QtWidgets.QComboBox(self.centralwidget)  
 self.table.setGeometry(QtCore.QRect(40, 50, 131, 41))  
 self.table.setObjectName("table")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.table.addItem("")  
 self.action = QtWidgets.QComboBox(self.centralwidget)  
 self.action.setGeometry(QtCore.QRect(200, 50, 151, 41))  
 self.action.setObjectName("action")  
 self.action.addItem("")  
 self.action.addItem("")  
 self.action.addItem("")  
 self.actLabel = QtWidgets.QLabel(self.centralwidget)  
 self.actLabel.setGeometry(QtCore.QRect(30, 10, 291, 16))  
 font = QtGui.QFont()  
 font.setFamily("Ubuntu")  
 font.setPointSize(11)  
 font.setBold(False)  
 font.setItalic(False)  
 font.setWeight(50)  
 self.actLabel.setFont(font)  
 self.actLabel.setObjectName("actLabel")  
 self.textEdit = QtWidgets.QTextEdit(self.centralwidget)  
 self.textEdit.setGeometry(QtCore.QRect(360, 50, 321, 70))  
 self.textEdit.setObjectName("textEdit")  
 self.genData = QtWidgets.QPushButton(self.centralwidget)  
 self.genData.setGeometry(QtCore.QRect(20, 330, 611, 51))  
 self.genData.setObjectName("genData")  
 self.textSearch = QtWidgets.QTextEdit(self.centralwidget)  
 self.textSearch.setGeometry(QtCore.QRect(20, 190, 751, 131))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 self.textSearch.setFont(font)  
 self.textSearch.setAutoFormatting(QtWidgets.QTextEdit.AutoAll)  
 self.textSearch.setOverwriteMode(True)  
 self.textSearch.setObjectName("textSearch")  
 self.labelSearch = QtWidgets.QLabel(self.centralwidget)  
 self.labelSearch.setGeometry(QtCore.QRect(30, 140, 151, 31))  
 self.labelSearch.setObjectName("labelSearch")  
 self.pushButton\_2 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_2.setGeometry(QtCore.QRect(20, 390, 751, 61))  
 self.pushButton\_2.setObjectName("pushButton\_2")  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(30, 300, 63, 20))  
 self.label.setText("")  
 self.label.setObjectName("label")  
 self.pushButton\_3 = QtWidgets.QPushButton(self.centralwidget)  
 self.pushButton\_3.setGeometry(QtCore.QRect(640, 350, 111, 31))  
 font = QtGui.QFont()  
 font.setPointSize(11)  
 self.pushButton\_3.setFont(font)  
 self.pushButton\_3.setObjectName("pushButton\_3")  
 self.info = QtWidgets.QLabel(self.centralwidget)  
 self.info.setGeometry(QtCore.QRect(380, 120, 241, 61))  
 font = QtGui.QFont()  
 font.setPointSize(9)  
 self.info.setFont(font)  
 self.info.setStyleSheet("color:green")  
 self.info.setObjectName("info")  
 self.error = QtWidgets.QLabel(self.centralwidget)  
 self.error.setGeometry(QtCore.QRect(370, 10, 401, 31))  
 self.error.setStyleSheet("color:red;")  
 self.error.setText("")  
 self.error.setObjectName("error")  
 self.full\_text\_box = QtWidgets.QComboBox(self.centralwidget)  
 self.full\_text\_box.setGeometry(QtCore.QRect(190, 130, 131, 41))  
 self.full\_text\_box.setObjectName("full\_text\_box")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.full\_text\_box.addItem("")  
 self.gen\_label = QtWidgets.QLabel(self.centralwidget)  
 self.gen\_label.setGeometry(QtCore.QRect(660, 320, 62, 19))  
 self.gen\_label.setText("")  
 self.gen\_label.setObjectName("gen\_label")  
 Database.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(Database)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 798, 25))  
 self.menubar.setObjectName("menubar")  
 Database.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(Database)  
 self.statusbar.setObjectName("statusbar")  
 Database.setStatusBar(self.statusbar)  
  
 self.retranslateUi(Database)  
 QtCore.QMetaObject.connectSlotsByName(Database)  
  
 def retranslateUi(self, Database):  
 \_translate = QtCore.QCoreApplication.translate  
 Database.setWindowTitle(\_translate("Database", "MainWindow"))  
 self.pushButton.setText(\_translate("Database", "Action"))  
 self.table.setItemText(0, \_translate("Database", "Person"))  
 self.table.setItemText(1, \_translate("Database", "Transport"))  
 self.table.setItemText(2, \_translate("Database", "Stop"))  
 self.table.setItemText(3, \_translate("Database", "Ticket"))  
 self.table.setItemText(4, \_translate("Database", "Schedule"))  
 self.table.setItemText(5, \_translate("Database", "Trip"))  
 self.action.setItemText(0, \_translate("Database", "delete"))  
 self.action.setItemText(1, \_translate("Database", "update"))  
 self.action.setItemText(2, \_translate("Database", "insert"))  
 self.actLabel.setText(\_translate("Database", "Choose table and action to do:"))  
 self.genData.setText(\_translate("Database", "Generate random data to database!"))  
 self.textSearch.setHtml(\_translate("Database", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"  
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"  
"p, li { white-space: pre-wrap; }\n"  
"</style></head><body style=\" font-family:\'Noto Sans\'; font-size:9pt; font-weight:400; font-style:normal;\">\n"  
"<p style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px; font-family:\'Cantarell\';\"><br /></p></body></html>"))  
 self.labelSearch.setText(\_translate("Database", "Enter word or phrase:"))  
 self.pushButton\_2.setText(\_translate("Database", "Find trips in time interval -->"))  
 self.pushButton\_3.setText(\_translate("Database", "Search "))  
 self.info.setText(\_translate("Database", " delete: pid = 10;\n"  
" insert: pid =10, name=\'Andriy\'... \n"  
" update: (where) pid = 10 \n"  
" (what)exemption = \'pensioner\' "))  
 self.full\_text\_box.setItemText(0, \_translate("Database", "Person"))  
 self.full\_text\_box.setItemText(1, \_translate("Database", "Transport"))  
 self.full\_text\_box.setItemText(2, \_translate("Database", "Stop"))  
 self.full\_text\_box.setItemText(3, \_translate("Database", "Ticket"))  
 self.full\_text\_box.setItemText(4, \_translate("Database", "Schedule"))  
 self.full\_text\_box.setItemText(5, \_translate("Database", "Trip"))

Файл trip\_d.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'trip\_dial.ui'  
#  
# Created by: PyQt5 UI code generator 5.15.1  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic5 is  
# run again. Do not edit this file unless you know what you are doing.  
  
  
from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_Dialog(object):  
 def \_\_init\_\_(self, Dialog):  
 Dialog.setObjectName("Dialog")  
 Dialog.resize(1158, 586)  
 self.pushButton = QtWidgets.QPushButton(Dialog)  
 self.pushButton.setGeometry(QtCore.QRect(10, 90, 1121, 31))  
 self.pushButton.setObjectName("pushButton")  
 self.label = QtWidgets.QLabel(Dialog)  
 self.label.setGeometry(QtCore.QRect(10, 0, 391, 51))  
 self.label.setObjectName("label")  
 self.label\_2 = QtWidgets.QLabel(Dialog)  
 self.label\_2.setGeometry(QtCore.QRect(20, 40, 391, 51))  
 self.label\_2.setObjectName("label\_2")  
 self.label\_3 = QtWidgets.QLabel(Dialog)  
 self.label\_3.setGeometry(QtCore.QRect(360, 40, 391, 51))  
 self.label\_3.setObjectName("label\_3")  
 self.textEdit = QtWidgets.QTextEdit(Dialog)  
 self.textEdit.setGeometry(QtCore.QRect(60, 50, 271, 31))  
 self.textEdit.setObjectName("textEdit")  
 self.textEdit\_2 = QtWidgets.QTextEdit(Dialog)  
 self.textEdit\_2.setGeometry(QtCore.QRect(400, 50, 281, 31))  
 self.textEdit\_2.setObjectName("textEdit\_2")  
 self.plainTextEdit = QtWidgets.QPlainTextEdit(Dialog)  
 self.plainTextEdit.setGeometry(QtCore.QRect(10, 130, 1121, 451))  
 font = QtGui.QFont()  
 font.setPointSize(8)  
 self.plainTextEdit.setFont(font)  
 self.plainTextEdit.setObjectName("plainTextEdit")  
  
 self.retranslateUi(Dialog)  
 QtCore.QMetaObject.connectSlotsByName(Dialog)  
  
 def retranslateUi(self, Dialog):  
 \_translate = QtCore.QCoreApplication.translate  
 Dialog.setWindowTitle(\_translate("Dialog", "Dialog"))  
 self.pushButton.setText(\_translate("Dialog", "Get Values"))  
 self.label.setText(\_translate("Dialog", "Please, enter time interval:"))  
 self.label\_2.setText(\_translate("Dialog", "From:"))  
 self.label\_3.setText(\_translate("Dialog", "To:"))