

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: «***Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL***»

Виконав: студент 3 курсу

ФПМ групи КВ-83

Ткачук Тарас

Перевірив: Павловський В.І.

Київ – 2020

**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 2-х 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).

***Вимоги до інтерфейсу користувача***

1. Використовувати консольний інтерфейс користувача.

***Варіант (предметна область):*** база даних для міжміського сполучення.

**Звіт**

**Нормалізована логічна модель даних БД**

Нижче наведено схему нормалізованої бази даних спроектованої в Лабораторній роботі №1.

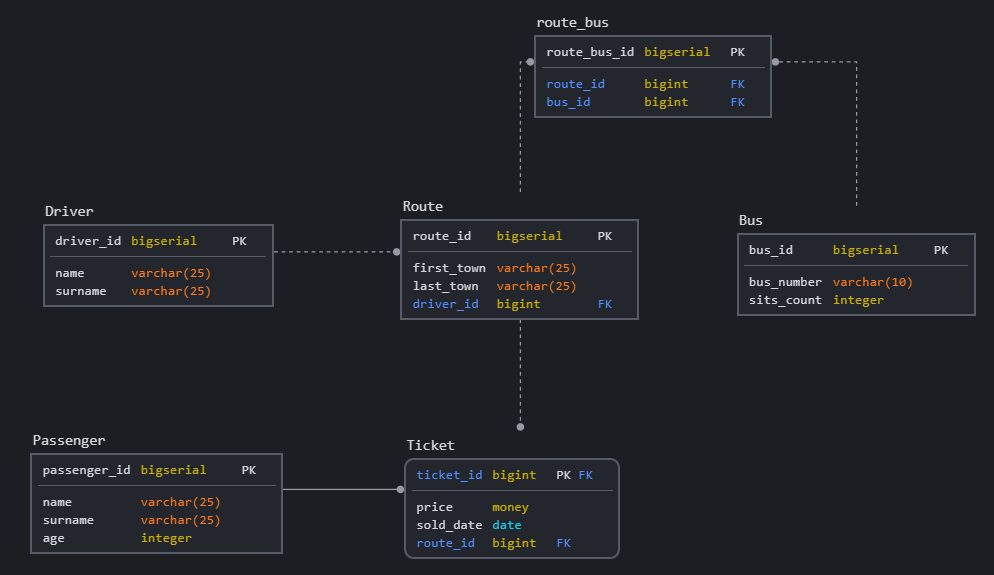


Рисунок 1 — Схема нормалізованої бази даних PostgreSQL на основі   
ER-моделі предметної області "Міжміське сполучення".

*Примітка*. При побудові схеми БД використано сервіс SqlDBM

Середовище розробки – Visual Studio Code. Мова програмування – Python 3.8. Бібліотека роботи з БД – psychopg2

**Опис програми**

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 5 модулів:

1. main.py — точка входу програми. Створення головного меню та підменю, яке являє собою відповідний контролер БД;
2. model.py — модуль містить опис класу Model, в якому виконується управління даними, логікою та правилами програми;
3. view.py — модуль містить опис класу View, який відповідає за обробку та виведення даних, отриманих в результаті запитів до БД;
4. controller.py — модуль містить опис класу Controller, який приймає введені користувачем дані і делегує представлення даних у View та обробку даних у Model;
5. backend.py — модуль, в якому виконується більшість бізнес-логіки за допомогою SQL-запитів;

**Структура меню програми**

Нижче наведено структуру меню програми.

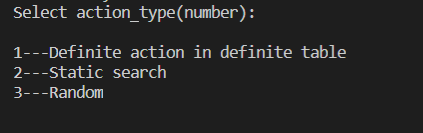


Рисунок 2.1 — Головне меню програми.

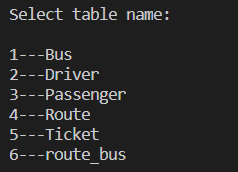


Рисунок 2.2 — Підменю для вибору таблиці

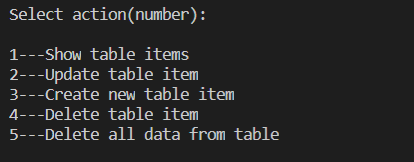
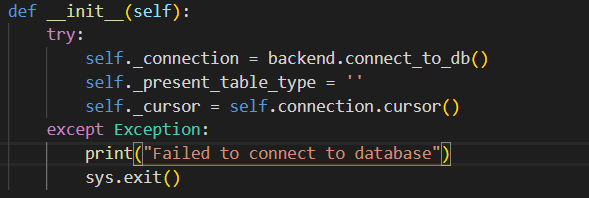


Рисунок 2.3 — Підменю для вибору типу запиту

**Лістинг програми для підключення до БД за допомогою бібліотек psycopg2**



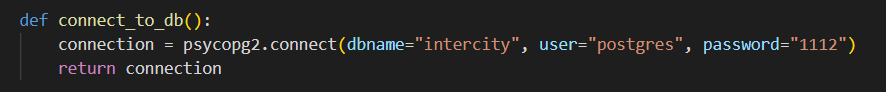


Рисунок 3 — Підключення до БД

**Лістинги програм з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив**

Перевірка наявності рядка у батьківській таблиці при додаванні запису виконується за допомогою конструкції try …. except, дивитись рисунки 4.1 , 4.2

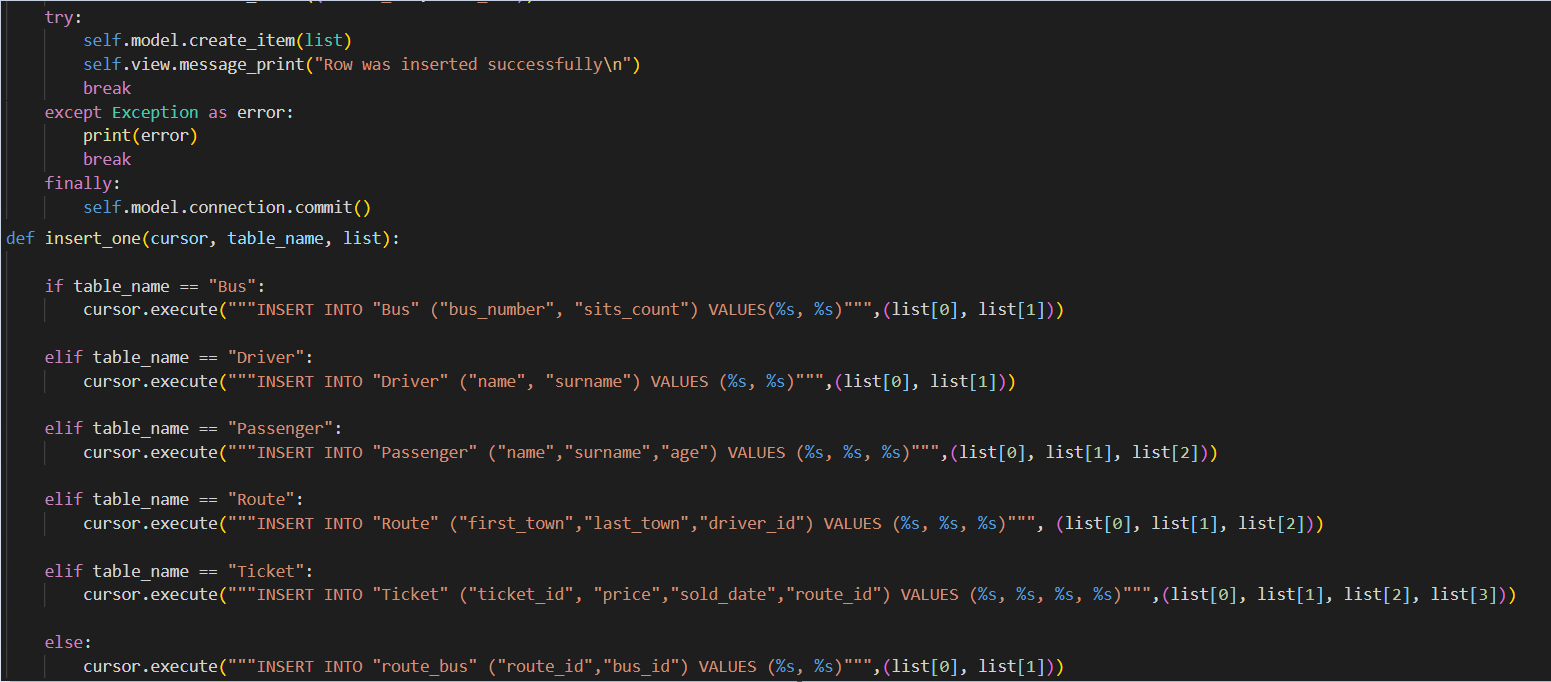


Рисунок 4.1 — Функція додавання запису у таблицю



Рисунок 4.2 — Помилка при спробі додати запис у підлеглу таблицю

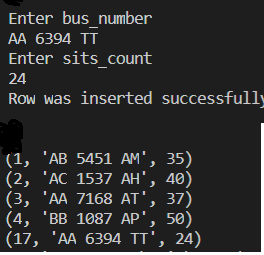
****

Рисунок 4.3 — Результат успішного внесення запису у таблицю "Bus"



Рисунок 4.4 — Функція оновлення запису в таблиці

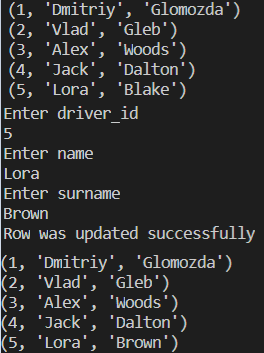


Рисунок 4.5 — Результат оновлення запису в таблиці "Driver"

Перевірка усіх можливих виключень, при видалені запису виконується за допомогою конструкції try …. except, дивитись рисункок 4.6

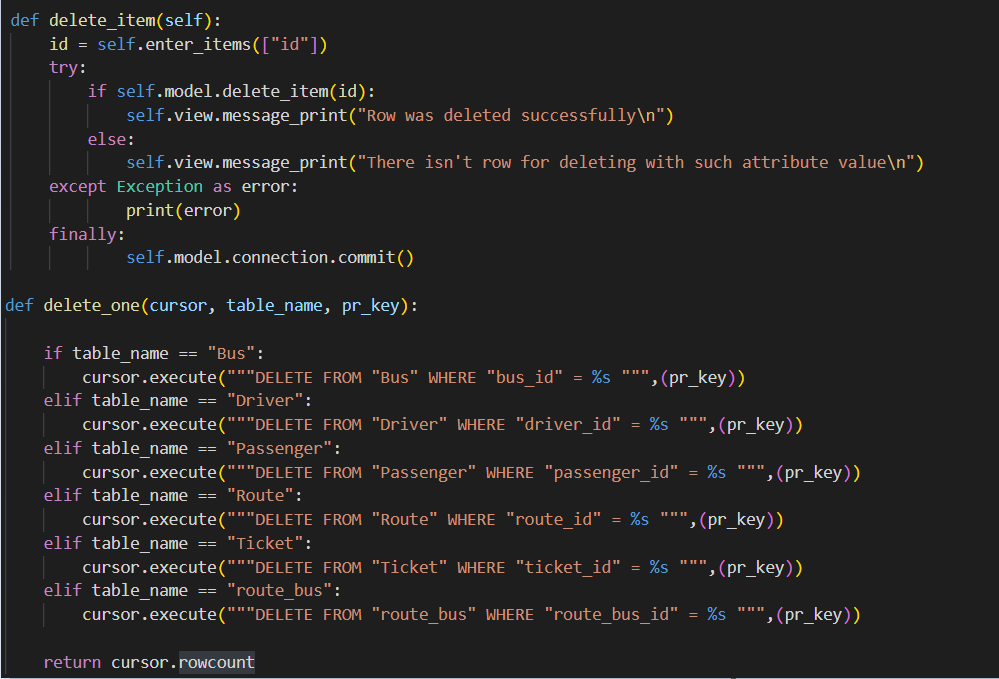


Рисунок 4.6 — Функція видалення запису в таблиці

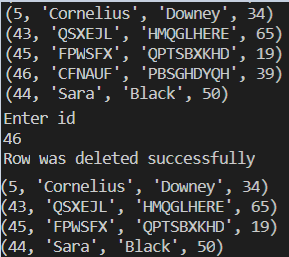


Рисунок 4.7 — Результат видалення запису в таблиці "Passenger"

**Лістинг програми зі статичним запитом пошуку**

Для демонстрації роботи пошуку за двома-трьома атрибутами з чотирьох сутностей одночасно використовуються таблиці Passenger, Ticket, Route та Driver.

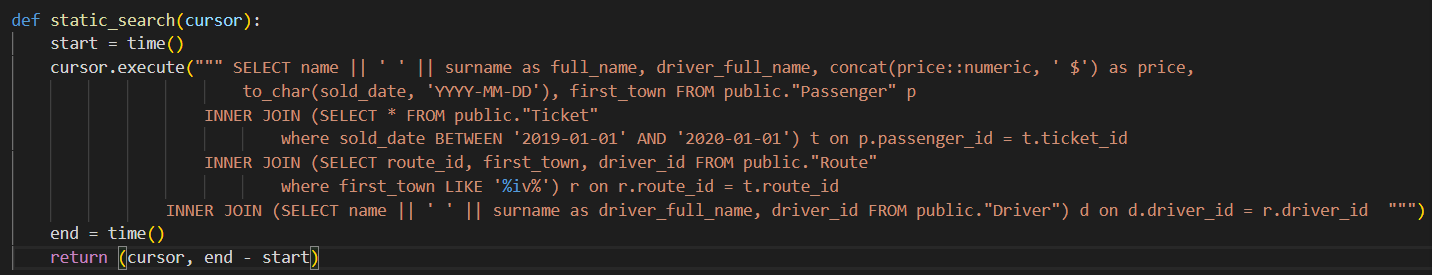


Рисунок 5.1 — Функція статичного запиту пошуку записів в таблицях

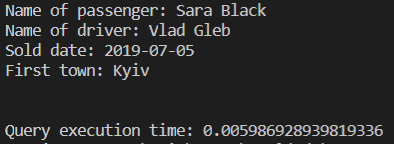


Рисунок 5.2 — Результат статичного запиту пошуку записів в таблицях та час виконання цього запиту

**Лістинг програми генерування «рандомізованих» даних**

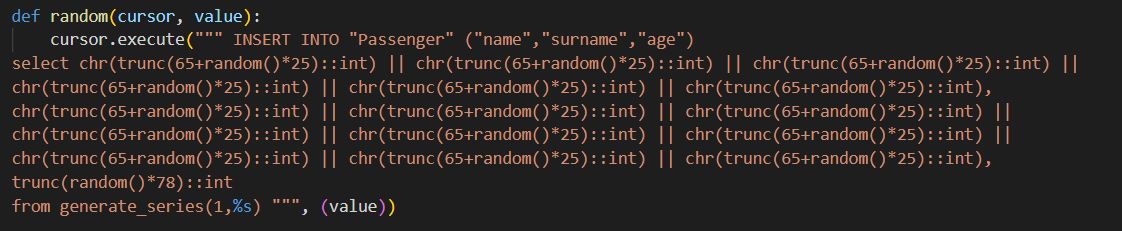


Рисунок 6.1 — Функція генерування «рандомізованих» даних

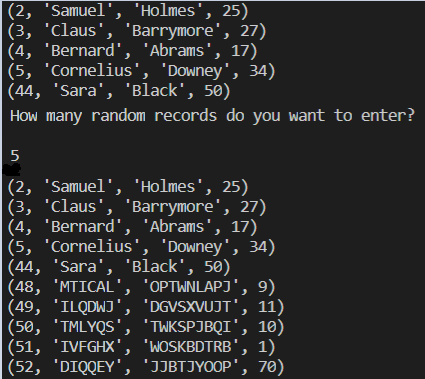


Рисунок 6.2 — Результат генерування 5 «рандомізованих» записів у таблицю "Passenger"

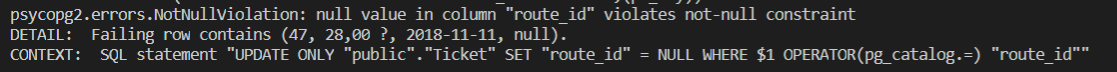
**Дослідження режимів обмеження ON DELETE**

*1. Режим CASCADE*

При видаленні запису з таблиці Route, запис з таблиці Ticket видаляється.

*2. Режим SET NULL*

При видаленні запису з таблиці Route, route\_id запис з таблиці Ticket встановлюється в null. Якщо в налаштуваннях таблиці вказати, що route\_id не може бути null, то перехоплюємо повідомлення про помилку.

****

*3. Режим SET DEFAULT (значення за замовчуванням = 0)*

При видаленні запису з таблиці Route, перехоплюємо повідомлення про помилку, так як маршруту з id = 0 не існує.

****

*4. Режим NO ACTION*

При видаленні запису з таблиці Route, виникає помилка; це поведінка за замовчуванням.



*5.Режим RESTRICT*

При видаленні запису з таблиці Route, виникає помилка; це пояснюється тим, що режим RESTRICT не дає можливості видалити батьківський рядок, якщо в нього є дочірні.

****

**Структура програми**

Нижче наведено структуру програми та взаємодію окремих модулів.

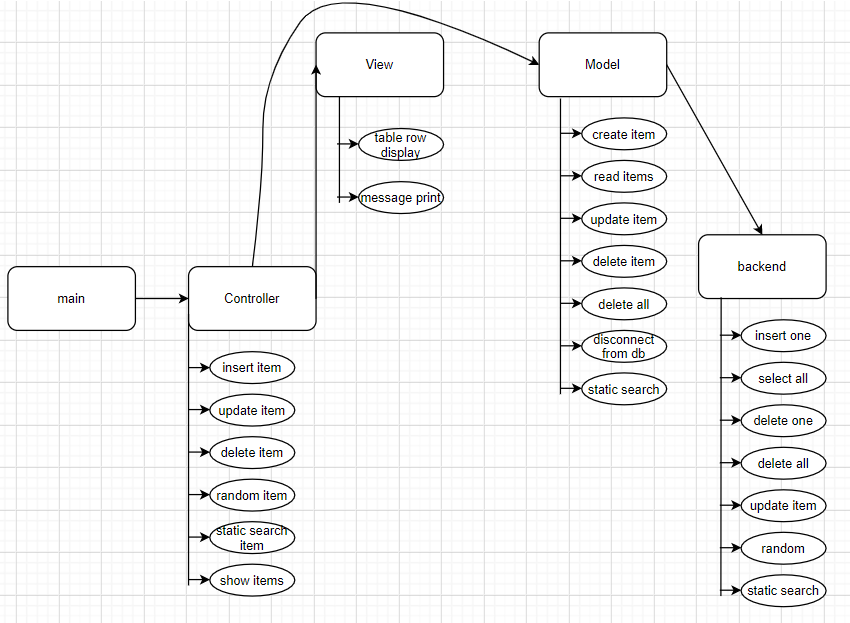
****

Рисунок 6 — Діаграма зв’язків модулів програми

**Код програми**

**main.py**

from controller import Controller

from model import ModelPostgreSQL

from view import View

if \_\_name\_\_ == '\_\_main\_\_':

    c = Controller(ModelPostgreSQL(),View())

    while True:

        type = c.action\_type\_select()

        if type == "1":

            c.table\_type\_select()

            c.action\_select()

        elif type == "2":

            c.static\_search()

        else:

            c.random\_insert()

        if not c.question\_about\_end():

            break

    c.disconnect\_from\_db()

**controller.py**

from dateutil import parser

import random

class Controller(object):

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

        self.model = model

        self.view = view

    def show\_items(self):

        items = self.model.read\_items()

        if items.rowcount:

            self.view.table\_rows\_display(items)

            return

        self.view.message\_print("This table was already empty\n")

    def enter\_items(self, table\_item\_names):

        return\_array = []

        for name in table\_item\_names:

            while True:

                self.view.enter\_cortege\_item\_display(name)

                inp = str(input())

                if validate\_input(name, inp):

                    return\_array.append(inp)

                    break

                else:

                    self.view.message\_print("Error:enter valid value\n")

        return return\_array

    def table\_type\_select(self):

        self.view.table\_name\_select\_display()

        while True:

            table\_type = str(input())

            if 0 < int(table\_type) < 7:

                if table\_type == "1":

                    self.model.present\_table\_type = "Bus"

                elif table\_type == "2":

                    self.model.present\_table\_type = "Driver"

                elif table\_type == "3":

                    self.model.present\_table\_type = "Passenger"

                elif table\_type == "4":

                    self.model.present\_table\_type = "Route"

                elif table\_type == "5":

                    self.model.present\_table\_type = "Ticket"

                elif table\_type == "6":

                    self.model.present\_table\_type = "route\_bus"

                return

            self.view.message\_print("Error:enter number from 1 to 6\n")

    def action\_type\_select(self):

        self.view.action\_type\_select\_display()

        while True:

            action\_type = str(input())

            if action\_type != "1" and action\_type != "2" and action\_type != "3":

                self.view.message\_print("Error:enter number from 1 to 3\n")

                continue

            break

        return action\_type

    def action\_select(self):

        self.view.action\_select\_display()

        while True:

            action = str(input())

            if action == "1":

                self.show\_items()

            elif action == "2":

                self.update\_item()

            elif action == "3":

                self.insert\_item()

            elif action == "4":

                self.delete\_item()

            elif action == "5":

                self.delete\_all()

            else:

                self.view.message\_print("Error:Enter number from 1-5\n")

                continue

            break

        self.model.connection.commit()

    def question\_about\_end(self):

        self.view.question\_about\_end\_display()

        while True:

            inp = str(input())

            if inp == "Y" or inp == "y":

                return True

            elif inp == "N" or inp == "n":

                return False

            else:

                self.view.message\_print("""Error:enter "Y" or "N"\n """)

    def disconnect\_from\_db(self):

        self.model.disconnect\_from\_db()

    def insert\_item(self):

        while True:

            if self.model.present\_table\_type == 'Bus':

                list = self.enter\_items(("bus\_number", "sits\_count"))

            elif self.model.present\_table\_type == 'Driver':

                list = self.enter\_items(("name", "surname"))

            elif self.model.present\_table\_type == 'Passenger':

                list = self.enter\_items(("name", "surname", "age"))

            elif self.model.present\_table\_type == 'Route':

                list = self.enter\_items(("first\_town", "last\_town", "driver\_id"))

            elif self.model.present\_table\_type == 'Ticket':

                list = self.enter\_items(("ticket\_id", "price", "sold\_date", "route\_id"))

            else:

                list = self.enter\_items(("route\_id", "bus\_id"))

            try:

                self.model.create\_item(list)

                self.view.message\_print("Row was inserted successfully\n")

                break

            except Exception as error:

                print(error)

                break

            finally:

                self.model.connection.commit()

    def update\_item(self):

        while True:

            if self.model.present\_table\_type == 'Bus':

                list = self.enter\_items(("bus\_id","bus\_number", "sits\_count"))

            elif self.model.present\_table\_type == 'Driver':

                list = self.enter\_items(("driver\_id","name", "surname"))

            elif self.model.present\_table\_type == 'Passenger':

                list = self.enter\_items(("passenger\_id","name", "surname", "age"))

            elif self.model.present\_table\_type == 'Route':

                list = self.enter\_items(("route\_id","first\_town", "last\_town", "driver\_id"))

            elif self.model.present\_table\_type == 'Ticket':

                list = self.enter\_items(("ticket\_id","price", "sold\_date", "route\_id"))

            else:

                list = self.enter\_items(("route\_bus\_id","route\_id", "bus\_id"))

            try:

                self.model.update\_item(list)

                self.view.message\_print("Row was updated successfully\n")

                break

            except Exception as error:

                print(error)

            finally:

                self.model.connection.commit()

    def delete\_item(self):

        id = self.enter\_items(["id"])

        if self.model.delete\_item(id):

            self.view.message\_print("Row was deleted successfully\n")

        else:

            self.view.message\_print("There isn't row for deleting with such attribute value\n")

    def delete\_all(self):

        if self.model.delete\_all():

            self.view.message\_print("All rows in table were deleted successfully\n")

        else:

            self.view.message\_print("Table was already empty\n")

    def random\_insert(self):

        self.view.message\_print("How many random records do you want to enter?\n")

        value = str(input())

        self.model.random(value)

        self.model.connection.commit()

    def static\_search(self):

        c = self.model.static\_search()

        if c:

            for row in c[0]:

                # self.view.message\_print(row)

                self.view.message\_print(f"Name of passenger: {row[0]}\nName of driver: {row[1]}\nCost of ticket: {row[2]}\n\

Sold date: {row[3]}\nFirst town: {row[4]}\n\n")

            self.view.message\_print(f"Query execution time: {c[1]}")

        else:

            self.view.message\_print("No data found")

def validate\_input(attr\_name, attr\_value):

    bound\_check = list(attr\_name.split(' '))

    if len(bound\_check) > 1:

        if bound\_check[1] == "Lower" or bound\_check[1] == "Upper":

            attr\_name = bound\_check[0]

    if attr\_name.find("table") != -1:

        if (attr\_value == "Bus" or attr\_value =="Driver" or attr\_value == "Passenger"

        or attr\_value == "Route" or attr\_value == "Ticket" or attr\_value == "route\_bus"):

            return True

        return False

    if "id" in attr\_name:

        if attr\_value.isdecimal():

            return True

    elif attr\_name == "sits\_count":

        if attr\_value.isdecimal():

            return True

    elif attr\_name == "name":

        if attr\_value.isalpha():

            return True

    elif attr\_name == "surname":

        if attr\_value.isalpha():

            return True

    elif attr\_name == "age":

        if attr\_value.isdecimal():

            return True

    elif attr\_name == "bus\_number":

        li = list(attr\_value.split(" "))

        if li[0].isalpha() and li[2].isalpha():

            if li[1].isdecimal:

                return True

        return False

    elif "town" in attr\_name:

        li = list(attr\_value.split(" "))

        for item in li:

            if not item.isalpha():

                return False

        return True

    elif attr\_name == "price":

        if attr\_value.isdecimal():

            return True

    elif attr\_name == "sold\_date":

        try:

            parser.parse(attr\_value)

            return True

        except ValueError:

            return False

**model.py**

import backend

import sys

class ModelPostgreSQL(object):

    def \_\_init\_\_(self):

        try:

            self.\_connection = backend.connect\_to\_db()

            self.\_present\_table\_type = ''

            self.\_cursor = self.connection.cursor()

        except Exception:

            print("Failed to connect to database")

            sys.exit()

    @property

    def connection(self):

        return self.\_connection

    @property

    def cursor(self):

        return self.\_cursor

    @property

    def present\_table\_type(self):

        return self.\_present\_table\_type

    @present\_table\_type.setter

    def present\_table\_type(self,new\_present\_table\_type):

        self.\_present\_table\_type = new\_present\_table\_type

    def create\_item(self,cortage):

        backend.insert\_one(self.cursor,self.present\_table\_type,cortage)

    def read\_items(self):

        return backend.select\_all(self.cursor,self.present\_table\_type)

    def update\_item(self, list):

        backend.update\_item(self.cursor, self.present\_table\_type, list)

    def delete\_item(self,pr\_key):

        return backend.delete\_one(self.cursor,self.present\_table\_type,pr\_key)

    def delete\_all(self):

        return backend.delete\_all(self.cursor,self.present\_table\_type)

    def disconnect\_from\_db(self):

        backend.disconnect\_from\_db(self.connection,self.cursor)

    def random(self, value):

        backend.random(self.cursor, value)

    def static\_search(self):

        return backend.static\_search(self.cursor)

**backend.py**

import psycopg2

from time import time

def insert\_one(cursor, table\_name, list):

    if table\_name == "Bus":

        cursor.execute("""INSERT INTO "Bus" ("bus\_number", "sits\_count") VALUES(%s, %s)""",(list[0], list[1]))

    elif table\_name == "Driver":

        cursor.execute("""INSERT INTO "Driver" ("name", "surname") VALUES (%s, %s)""",(list[0], list[1]))

    elif table\_name == "Passenger":

        cursor.execute("""INSERT INTO "Passenger" ("name","surname","age") VALUES (%s, %s, %s)""",(list[0], list[1], list[2]))

    elif table\_name == "Route":

        cursor.execute("""INSERT INTO "Route" ("first\_town","last\_town","driver\_id") VALUES (%s, %s, %s)""", (list[0], list[1], list[2]))

    elif table\_name == "Ticket":

        cursor.execute("""INSERT INTO "Ticket" ("ticket\_id", "price","sold\_date","route\_id") VALUES (%s, %s, %s, %s)""",(list[0], list[1], list[2], list[3]))

    else:

        cursor.execute("""INSERT INTO "route\_bus" ("route\_id","bus\_id") VALUES (%s, %s)""",(list[0], list[1]))

def select\_all(cursor, table\_name):

    if table\_name == "Ticket":

        cursor.execute(""" SELECT ticket\_id, concat(price::numeric, ' $') as price, to\_char(sold\_date, 'YYYY-MM-DD'), route\_id from "Ticket" """)

    else:

        cursor.execute(""" SELECT \* FROM "{}" """.format(table\_name))

    return cursor

def delete\_one(cursor, table\_name, pr\_key):

    if table\_name == "Bus":

        cursor.execute("""DELETE FROM "Bus" WHERE "bus\_id" = %s """,(pr\_key))

    elif table\_name == "Driver":

        cursor.execute("""DELETE FROM "Driver" WHERE "driver\_id" = %s """,(pr\_key))

    elif table\_name == "Passenger":

        cursor.execute("""DELETE FROM "Passenger" WHERE "passenger\_id" = %s """,(pr\_key))

    elif table\_name == "Route":

        cursor.execute("""DELETE FROM "Route" WHERE "route\_id" = %s """,(pr\_key))

    elif table\_name == "Ticket":

        cursor.execute("""DELETE FROM "Ticket" WHERE "ticket\_id" = %s """,(pr\_key))

    elif table\_name == "route\_bus":

        cursor.execute("""DELETE FROM "route\_bus" WHERE "route\_bus\_id" = %s """,(pr\_key))

    return cursor.rowcount

def delete\_all(cursor,table\_name):

    cursor.execute("""DELETE FROM "{}" """.format(table\_name))

    return cursor.rowcount

def update\_item(cursor, table\_name, list):

    if table\_name == "Bus":

        cursor.execute("""UPDATE "Bus" SET "bus\_number" = %s, "sits\_count" = %s

        WHERE "bus\_id" = %s """, (list[1], list[2], list[0]))

    elif table\_name == "Driver":

        cursor.execute("""UPDATE "Driver" SET "name" = %s, "surname" = %s

        WHERE "driver\_id" = %s """, (list[1], list[2], list[0]))

    elif table\_name == "Passenger":

        cursor.execute("""UPDATE "Passenger" SET "name" = %s, "surname" = %s, "age" = %s

        WHERE "passenger\_id" = %s """, (list[1], list[2], list[3], list[0]))

    elif table\_name == "Route":

        cursor.execute("""UPDATE "Route" SET "first\_town" = %s, "last\_town" = %s, "driver\_id" = %s

        WHERE "route\_id" = %s """, (list[1], list[2], list[3], list[0]))

    elif table\_name == "Ticket":

        cursor.execute("""UPDATE "Ticket" SET "price" = %s, "sold\_date" = %s, "route\_id" = %s

        WHERE "ticket\_id" = %s """, (list[1], list[2], list[3], list[0]))

    else:

        cursor.execute("""UPDATE "route\_bus" SET "route\_id" = %s, "bus\_id" = %s

        WHERE "route\_bus\_id" = %s """, (list[1], list[2], list[0]))

    return cursor.rowcount

def connect\_to\_db():

    connection = psycopg2.connect(dbname="intercity", user="postgres", password="1112")

    return connection

def disconnect\_from\_db(connection,cursor):

    cursor.close()

    connection.close()

    print("Connection with PostgreSQL is closed")

def random(cursor, value):

    cursor.execute(""" INSERT INTO "Passenger" ("name","surname","age")

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

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

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

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

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

trunc(random()\*78)::int

from generate\_series(1,%s) """, (value))

def static\_search(cursor):

    start = time()

    cursor.execute(""" SELECT name || ' ' || surname as full\_name, driver\_full\_name, concat(price::numeric, ' $') as price,

                        to\_char(sold\_date, 'YYYY-MM-DD'), first\_town FROM public."Passenger" p

                    INNER JOIN (SELECT \* FROM public."Ticket"

                            where sold\_date BETWEEN '2019-01-01' AND '2020-01-01') t on p.passenger\_id = t.ticket\_id

                    INNER JOIN (SELECT route\_id, first\_town, driver\_id FROM public."Route"

                            where first\_town LIKE '%iv%') r on r.route\_id = t.route\_id

                INNER JOIN (SELECT name || ' ' || surname as driver\_full\_name, driver\_id FROM public."Driver") d on d.driver\_id = r.driver\_id  """)

    end = time()

    return (cursor, end - start)

**view.py**

class View(object):

    @staticmethod

    def action\_type\_select\_display():

        print("Select action\_type(number):\n")

        print("1---Definite action in definite table\n2---Static search\n3---Random")

    @staticmethod

    def table\_name\_select\_display():

        print("Select table name:\n")

        print("1---Bus\n2---Driver\n3---Passenger\n4---Route\n5---Ticket\n6---route\_bus\n")

    @staticmethod

    def action\_select\_display():

        print("Select action(number):\n")

        print("1---Show table items\n2---Update table item\n3---Create new table item\n"

        "4---Delete table item\n5---Delete all data from table\n")

    @staticmethod

    def enter\_cortege\_item\_display(item):

        print("Enter {}".format(item))

    @staticmethod

    def table\_rows\_display(items):

        cursor = items

        row = items.fetchone()

        while row is not None:

            print(row)

            row = cursor.fetchone()

    @staticmethod

    def question\_about\_end\_display():

        print("Continue to work with Database?(Y/N)\n")

    @staticmethod

    def message\_print(message):

        print(message)