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.

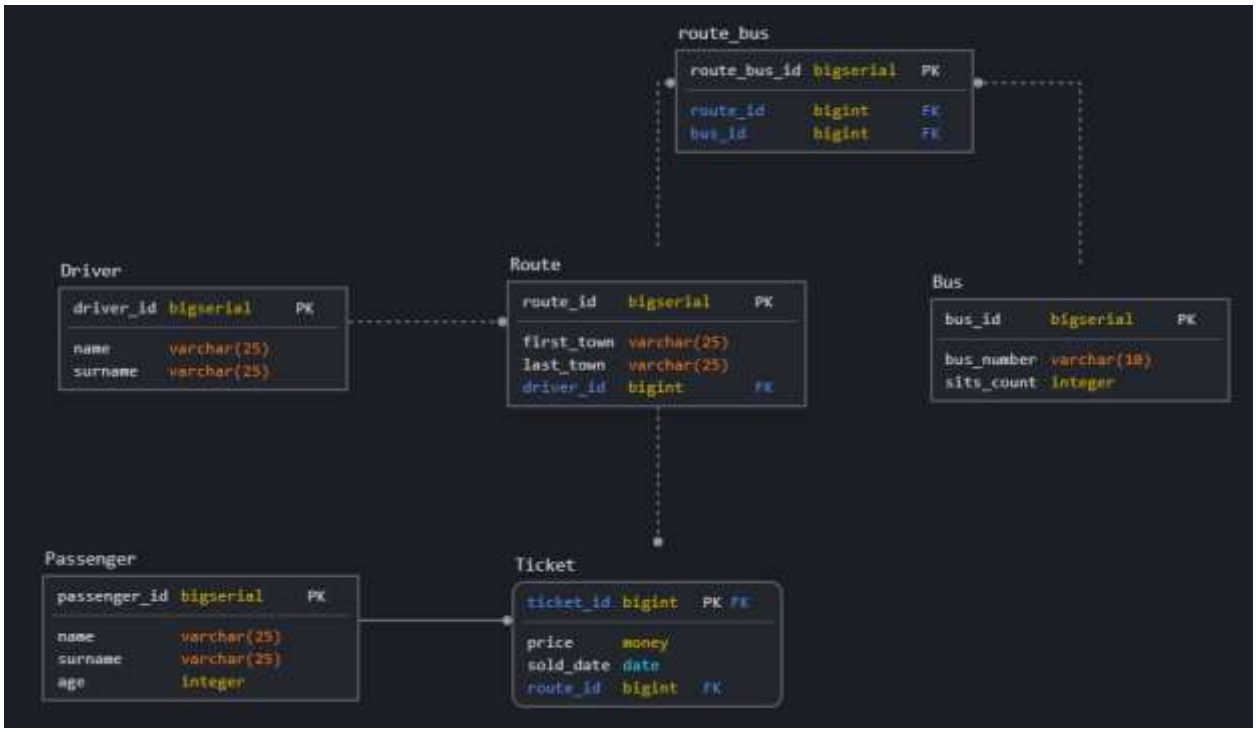


Рисунок 1 — Схема нормалізованої бази даних PostgreSQL на основі ER-моделі предметної області "Міжміське сполучення".

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.

```
Select action_type(number):

1---Definite action in definite table
2---Static search
3---Random
```

Рисунок 2.1 — Головне меню програми.

```
Select table name:

1---Bus
2---Driver
3---Passenger
4---Route
5---Ticket
6---route_bus
```

Рисунок 2.2 — Підменю для вибору таблиці

```
Select action(number):

1---Show table items
2---Update table item
3---Create new table item
4---Delete table item
5---Delete all data from table
```

Рисунок 2.3 — Підменю для вибору типу запиту

Лістинг програми для підключення до БД за допомогою бібліотек psycopg2

```
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()
```

```
def connect_to_db():
    connection = psycopg2.connect(dbname="intercity", user="postgres", password="1112")
    return connection
```

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

```
    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 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]))
```

Рисунок 4.1 — Функція додавання запису у таблицю

```
insert or update on table "Route" violates foreign key constraint "routeFK"
DETAIL:  Key (driver_id)=(44) is not present in table "Driver".
```

Рисунок 4.2 — Помилка при спробі додати запис у підлеглу таблицю

```
Enter bus_number
AA 6394 TT
Enter sits_count
24
Row was inserted successfully

(1, 'AB 5451 AM', 35)
(2, 'AC 1537 AH', 40)
(3, 'AA 7168 AT', 37)
(4, 'BB 1087 AP', 50)
(17, 'AA 6394 TT', 24)
```

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

```
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
```

Рисунок 4.4 — Функція оновлення запису в таблиці

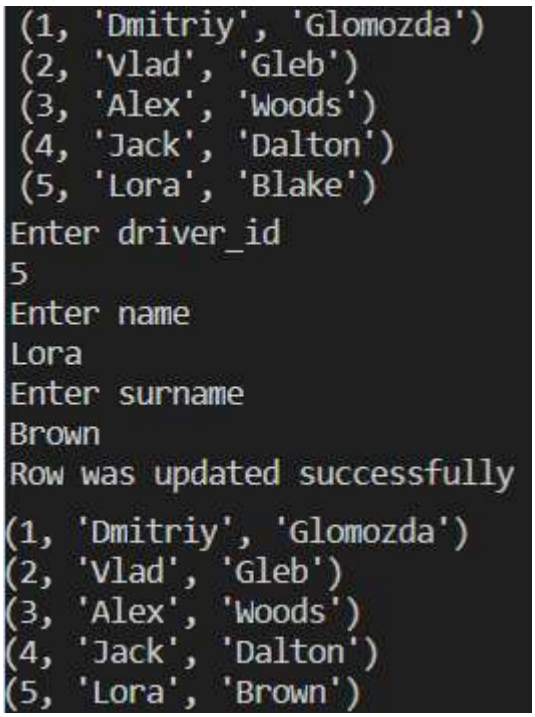


Рисунок 4.5 — Результат оновлення запису в таблиці "Driver"

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

```python
def delete_item(self):
    id = self.enter_items(["id"])
    try:
        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")
    except Exception as error:
            print(error)
    finally:
            self.model.connection.commit()

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
```

Рисунок 4.6 — Функція видалення запису в таблиці

```
(5, 'Cornelius', 'Downey', 34)
(43, 'QSXEJL', 'HMQGLHERE', 65)
(45, 'FPWSFX', 'QPTSBXKHD', 19)
(46, 'CFNAUF', 'PBSGHDYQH', 39)
(44, 'Sara', 'Black', 50)
Enter id
46
Row was deleted successfully
(5, 'Cornelius', 'Downey', 34)
(43, 'QSXEJL', 'HMQGLHERE', 65)
(45, 'FPWSFX', 'QPTSBXKHD', 19)
(44, 'Sara', 'Black', 50)
```

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.

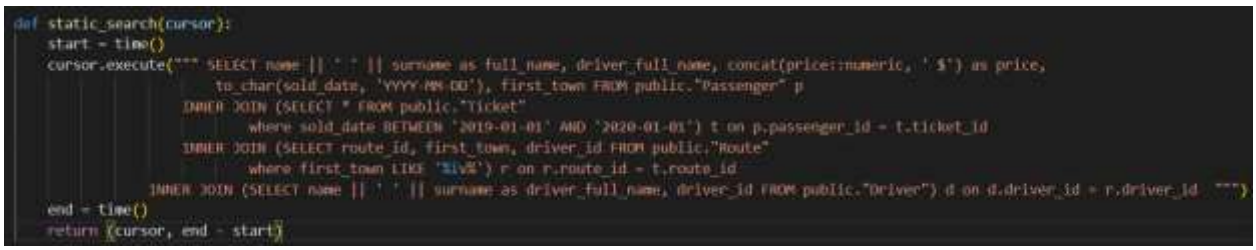


Рисунок 5.1 — Функція статичного запиту пошуку записів в таблицях

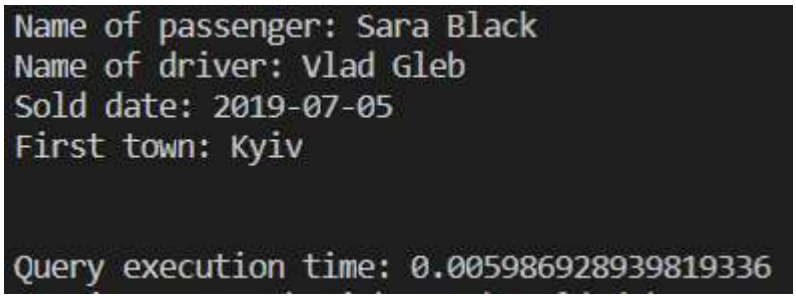


Рисунок 5.2 — Результат статичного запиту пошуку записів в таблицях та час виконання цього запиту

Лістинг програми генерування «рандомізованих» даних

```
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))
```

Рисунок 6.1 — Функція генерування «рандомізованих» даних

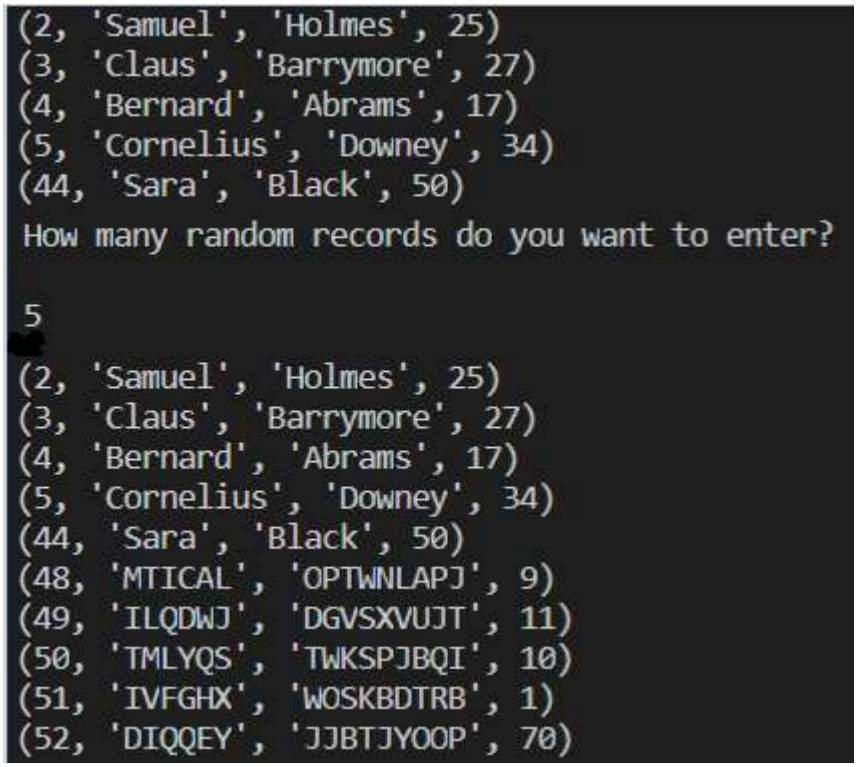


Рисунок 6.2 — Результат генерування 5 «рандомізованих» записів у таблицю "Passenger"

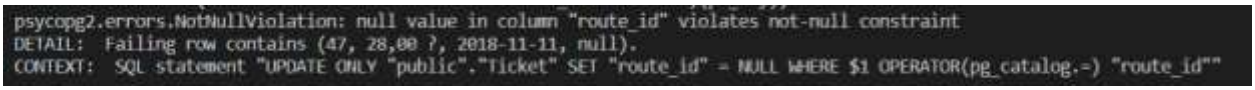
Дослідження режимів обмеження ON DELETE

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.

```
psycopg2.errors.ForeignKeyViolation: update or delete on table "Route" violates foreign key constraint "ticketFK" on table "Ticket"
DETAIL:  Key (route_id)=(8) is still referenced from table "Ticket".
```

Структура програми

Нижче наведено структуру програми та взаємодію окремих модулів.

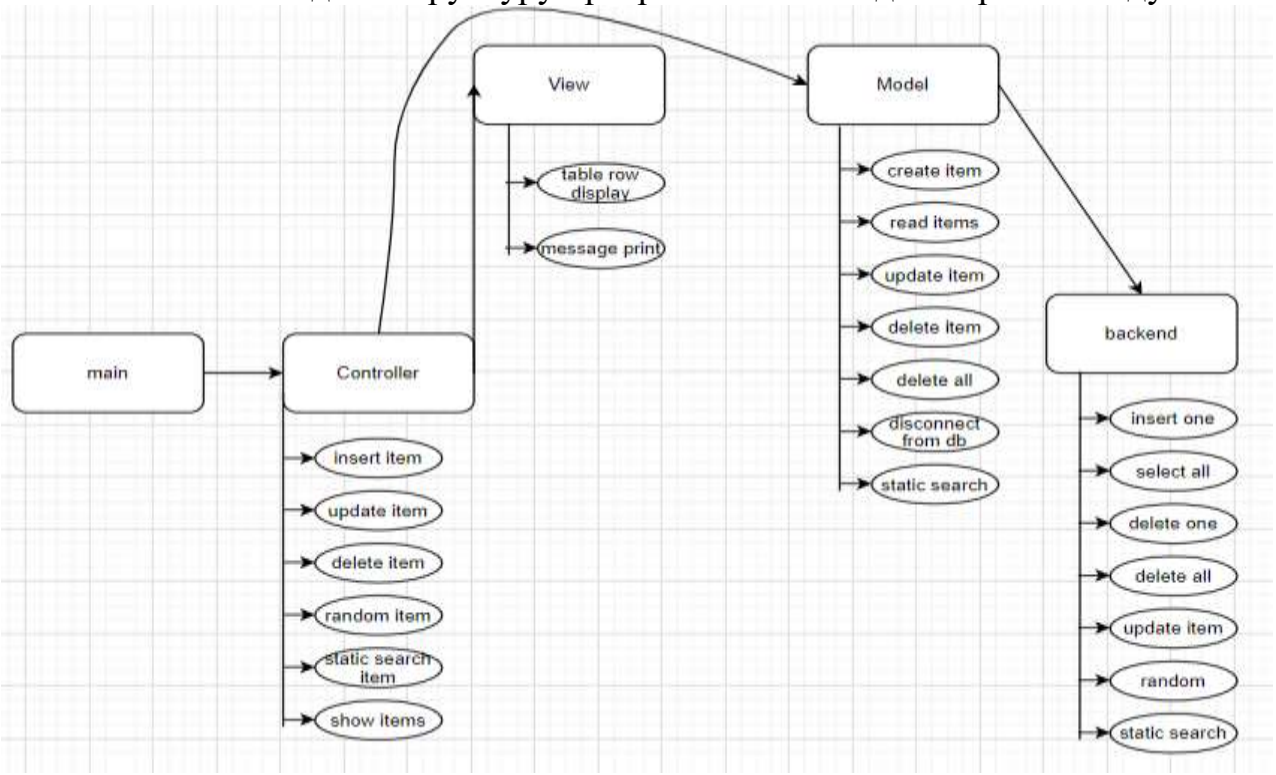


Рисунок 6 — Діаграма зв’язків модулів програми

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", "rout
e_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", "d
river_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 attri
bute 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 dr
iver: {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 == "Passen
ger"
        or attr_value == "Route" or attr_value == "Ticket" or attr_value == "rout
e_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_i
d") 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 pric
e, 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, "ag
e" = %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, "ro
ute_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(6
5+random()*25)::int),
chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(6
5+random()*25)::int) ||
chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(6
5+random()*25)::int) ||
chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(6
5+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_na
me, 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 publi
c."Route"
                            where first_town LIKE '%iv%') r on r.route_id = t.rou
te_id
                INNER JOIN (SELECT name || ' ' || surname as driver_full_name, dr
iver_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)
```