A black and white drawing of a building

Description automatically generated

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Розрахунково-графічна робота**

з дисципліни **Бази даних і засоби управління**

*на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”*

Виконав:

студент ІІI курсу

групи КВ-12

Бичко К. В.

Перевірив:

Павловский В. І.

Київ – 2023

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

**Виконання роботи**

Нижче будуть наведені сутності предметної області:

**Сутності предметної області**

Для побудови бази даних обраної області, були виділені такі сутності:

1. Потяг (Train)

Атрибути: номер потягу, кількість місць.

Призначення: збереження даних щодо маршрутів по яким курсують

потяги.

1. Квиток (Ticket)

Атрибути: ідентифікатор квитка, ціна, ПІБ пасажира.

Призначення: збереження даних щодо квитків та проданих місць за

напрямком.

1. Станція (Station)

Атрибути: ідентифікатор станції, назва станції.

Призначення: збереження даних щодо станцій та курсуючих по ній потягів.

Нижче будуть наведені зв’язки між сутностями предметної області:

**Зв’язки між сутностями предметної області**

Зв’язок-сутність «Потяг» - «Станція» є зв’язком N:M. В конкретну Дату один потяг може їхати через багато станцій, і одна станція може приймати багато потягів.

Зв’язок «Потяг-Станція» - «Квиток» є зв’язком 1:N. Один квиток продається на багато пар Потяг-Станція. З цього автоматично випливає:

Зв’язок «Потяг» - «Квиток» є зв’язком 1:N. Один квиток може бути проданим лише на один потяг, але на один потяг можна продати багато квитків.

Зв’язок «Станція» - «Квиток» є зв’язком 1:N. Один квиток може бути проданим лише до однієї станції, але до однієї станції може продаватися багато квитків.

Графічне подання концептуальної моделі «Сутність-зв’язок» зображено на рисунку 1.

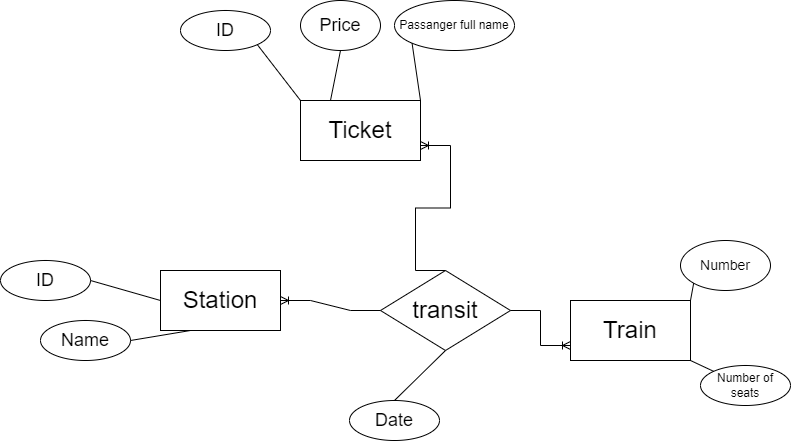


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 2.

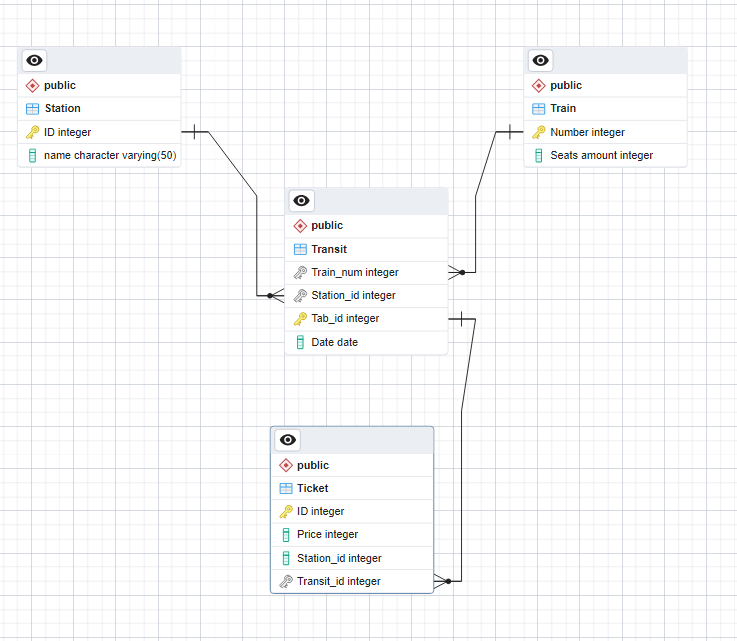


Рисунок 2 – Логічна модель

**Середовище та компоненти розробки**

У процесі розробки була використана мова програмування Python, інтегроване середовище розробки PyCharm, а також була використана бібліотека psycopg2, яка надає API для взаємодії з базою даних PostgreSQL.

**Шаблон проектування**

Модель-представлення-контролер (MVC) - це шаблон проектування, який використовується у розробці програм. Кожен компонент відповідає за конкретну функціональну частину:

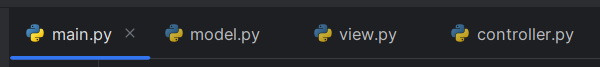
Модель (Model) - це клас, який визначає логіку роботи з даними. Він обробляє всі операції з даними, такі як додавання, оновлення та видалення.

Представлення (View) - це клас, який дозволяє користувачеві взаємодіяти з програмою. В даному випадку це консольний інтерфейс, який відображає дані користувачу та приймає введення з екрану.

Контролер (Controller) - це клас, що відповідає за взаємодію між користувачем та системою. Він приймає дані, введені користувачем, та обробляє їх. Залежно від результатів, він викликає відповідні дії в Model або View.

Цей підхід дозволяє розділити логіку програми на логічні компоненти, що полегшує розробку, тестування та підтримку продукту.

**Структура програми та її опис**

****

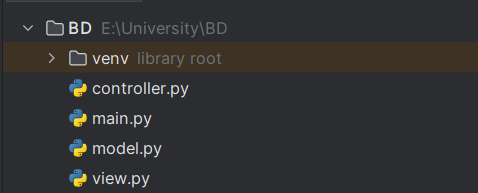
****

Рисунок 3 – Структура програми

З файлу main.py відбувається виклик контролера та передача йому управління.

У файлі model.py описаний клас моделі, який відповідає за управління підключенням до бази даних і виконанням низькорівневих запитів до неї.

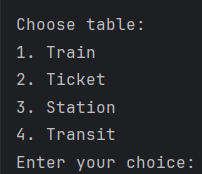
У файлі controller.py реалізовано інтерфейс взаємодії з користувачем, включаючи обробку запитів користувача, виконання пошуку, а також інші дії, необхідні для взаємодії з моделлю та представленням.

У файлі view.py описаний клас, який відображає результати виконання різних дій користувача на екрані консолі. Цей компонент відповідає за представлення даних користувачу в зручному для сприйняття вигляді.

Отже, структура програми відповідає патерну MVC.

**Структура меню програми**

На рисунку 4 зображено меню користувача, яке складається з двох частин, спочатку обирається таблиця для роботи, потім дія яку користувач бажає зробити.



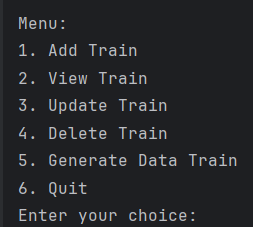


Рисунок 4 – Структура меню користувача

**Фрагмент коду (файл controller.py), в якому наведено головний цикл роботи програми**

def run(self):  
 while True:  
 choice = self.show\_tables()  
 if choice == '1':  
 choice = self.show\_menu\_train()  
 if choice == '1':  
 self.add\_train()  
 elif choice == '2':  
 self.view\_trains()  
 elif choice == '3':  
 self.update\_train()  
 elif choice == '4':  
 self.delete\_train()  
 elif choice == '5':  
 self.generate\_data\_train()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
 elif choice == '2':  
 choice = self.show\_menu\_ticket()  
 if choice == '1':  
 self.add\_ticket()  
 elif choice == '2':  
 self.view\_tickets()  
 elif choice == '3':  
 self.update\_ticket()  
 elif choice == '4':  
 self.delete\_ticket()  
 elif choice == '5':  
 self.generate\_data\_ticket()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
 elif choice == '3':  
 choice = self.show\_menu\_station()  
 if choice == '1':  
 self.add\_station()  
 elif choice == '2':  
 self.view\_stations()  
 elif choice == '3':  
 self.update\_station()  
 elif choice == '4':  
 self.delete\_station()  
 elif choice == '5':  
 self.generate\_data\_station()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
 elif choice == '4':  
 choice = self.show\_menu\_transit()  
 if choice == '1':  
 self.add\_transit()  
 elif choice == '2':  
 self.view\_transits()  
 elif choice == '3':  
 self.update\_transit()  
 elif choice == '4':  
 self.delete\_transit()  
 elif choice == '5':  
 self.generate\_data\_transit()  
 elif choice == '6':  
 self.model.conn.close()  
 break

**Фрагменти коду (файл model.py), в якому наведено функції внесення, редагування, вилучення та генерації даних у базі даних**

Функції внесення даних:

Дані функції, за наявності в таблиці зовнішніх ключів перевіряють чи введений користувачем зовнішній ключ існує в зв’язаній таблиці. За умови проходження всіх перевірок, вставка дозволяється. В іншому випадку, необхідно почати спочатку.

def add\_train(self, stsam):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Train" ("Seats amount") VALUES (%s)', (stsam,))  
 self.conn.commit()

def add\_ticket(self, prc, trs\_id, pas\_nm):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 if trs\_id\_exists:  
 c.execute('INSERT INTO "Ticket" ("Price","Transit\_id","Pas\_full\_name") VALUES (%s,%s,%s)',  
 (prc, trs\_id, pas\_nm, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def add\_station(self, name):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Station" ("name") VALUES (%s)',  
 (name, ))  
 self.conn.commit()

def add\_transit(self, tr\_num, st\_id, date):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {tr\_num})')  
 tr\_num\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {st\_id})')  
 st\_id\_exists = c.fetchone()[0]  
 if tr\_num\_exists and st\_id\_exists:  
 c.execute('INSERT INTO "Transit" ("Train\_num","Station\_id","Date") VALUES (%s,%s,%s)',  
 (tr\_num, st\_id, date, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

Функції оновлення даних:

Дані функції перевіряють валідність внесених змін. Якщо користувач намагається ввести зміни у поле зовнішніх ключів, то перевіряє чи введений користувачем зовнішній ключ існує в зв’язаній таблиці. За умови проходження всіх перевірок, вставка дозволяється. В іншому випадку, необхідно почати спочатку.

def update\_train(self, num, stsam):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {num})')  
 num\_exists = c.fetchone()[0]  
 if num\_exists:  
 c.execute('UPDATE "Train" SET "Seats amount"=%s WHERE "Number"=%s', (stsam, num))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def update\_ticket(self, idd, prc, trs\_id, pas\_nm):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Ticket" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if trs\_id\_exists and idd\_exists:  
 c.execute('UPDATE "Ticket" SET "Price"=%s, "Transit\_id"=%s, "Pas\_full\_name"=%s WHERE "ID"=%s',  
 (prc, trs\_id, pas\_nm, idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def update\_station(self, idd, name):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if idd\_exists:  
 c.execute('UPDATE "Station" SET "name"=%s WHERE "ID"=%s',  
 (name, idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def update\_transit(self, tr\_num, st\_id, date, trs\_id):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {tr\_num})')  
 tr\_num\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {st\_id})')  
 st\_id\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 if trs\_id\_exists and tr\_num\_exists and st\_id\_exists:  
 c.execute('UPDATE "Transit" SET "Train\_num"=%s, "Station\_id"=%s, "Date"=%s WHERE "Tab\_id"=%s',  
 (tr\_num, st\_id, date, trs\_id, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

Функції видалення значень:

Дані функції перевіряють чи існують зв’язані таблиці. Якщо так, то виконується перевірка чи значення, яке користувач хоче видалити, існує в зв’язаній таблиці. За умови його існування, відбувається каскадне видалення.

def delete\_train(self, num):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {num})')  
 num\_exists = c.fetchone()[0]  
 if num\_exists:  
 c.execute('DELETE FROM "Train" WHERE "Number"=%s', (num,))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def delete\_ticket(self, idd):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Ticket" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if idd\_exists:  
 c.execute('DELETE FROM "Ticket" WHERE "ID"=%s', (idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def delete\_station(self, idd):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if idd\_exists:  
 c.execute('DELETE FROM "Station" WHERE "ID"=%s', (idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

def delete\_transit(self, trs\_id):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 if trs\_id\_exists:  
 c.execute('DELETE FROM "Transit" WHERE "Tab\_id"=%s', (trs\_id, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1

Функції генерування даних:

Дані функції генерують задану кількість нових записів у задану таблицю. Коли відбувається вставка у поле зовнішніх ключів, з зв’язаної таблиці обирається значення уже існуючих значень зовнішніх ключів. Завдяки цьому досягається умова зв’язаності таблиць зовнішніми ключами. Для всіх інших полів вставка відбувається згідно їх типу.

def generate\_data\_train(self, count):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Train" ("Seats amount") SELECT ((RANDOM() \* 1000 + 1000)::INT) '  
 'FROM generate\_series(1,%s)', (count,))  
 self.conn.commit()  
  
def generate\_data\_station(self, count):  
 c = self.conn.cursor()  
 for i in range(count):  
 c.execute('INSERT INTO "Station" (name) SELECT random\_names.name '  
 'FROM unnest(ARRAY[\'Kiyv\', \'Chernihiv\', \'Nizhyn\','  
 ' \'Novoselivka\', \'Sumy\', \'Studenyky\']) AS random\_names(name) '  
 'ORDER BY random() '  
 'LIMIT 1;', (count,))  
 self.conn.commit()  
  
def generate\_data\_transit(self, count):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Transit" ("Train\_num","Station\_id","Date") SELECT '  
 'T1."Number" AS "Train\_num",'  
 'S1."ID" AS "Station\_id",'  
 'current\_timestamp - random() \* interval \'365 days\' AS "Date" '  
 'FROM '  
 '"Train" T1 '  
 'CROSS JOIN "Station" S1 '  
 'WHERE RANDOM() < 1 '  
 'LIMIT %s;', (count,))  
 self.conn.commit()  
  
def generate\_data\_ticket(self, count):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Ticket" ("Price", "Transit\_id", "Pas\_full\_name")'  
 'SELECT'  
 'random() \* 1000 + 1000, '  
 '"Tab\_id" AS "Transit\_id",'  
 'unnest(ARRAY[\'Alice\', \'Bob\', \'Charlie\', \'David\', \'Eva\', \'Frank\'] AS "Pas\_full\_name"'  
 'FROM "Transit")'  
 'ORDER BY random()'  
 'LIMIT 1000;', (count,))  
 self.conn.commit()

**Повний код програми**

Файл main.py:

from controller import Controller  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 controller = Controller()  
 controller.run()

Файл model.py:

import psycopg2  
  
  
class Model:  
 def \_\_init\_\_(self):  
 self.conn = psycopg2.connect(  
 dbname='KiriloZaliznitsya',  
 user='postgres',  
 password='q123w456',  
 host='localhost',  
 port=8080  
 )  
 self.create\_table\_train()  
 self.create\_table\_station()  
 self.create\_table\_transit()  
 self.create\_table\_ticket()  
  
 def get\_all\_attr\_table(self, tblnm):  
 c = self.conn.cursor()  
 c.execute(f'SELECT \* FROM {tblnm}')  
 return c.fetchall()  
  
 def create\_table\_train(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Train" (  
 "Number" SERIAL PRIMARY KEY,  
 "Seats amount" integer NOT NULL  
 )  
 ''')  
  
 # Check if the table exists  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Train')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Table does not exist, so create it  
 c.execute('''  
 CREATE TABLE "Train" (  
 "Number" SERIAL PRIMARY KEY,  
 "Seats amount" TEXT NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def add\_train(self, stsam):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Train" ("Seats amount") VALUES (%s)', (stsam,))  
 self.conn.commit()  
  
 def update\_train(self, num, stsam):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {num})')  
 num\_exists = c.fetchone()[0]  
 if num\_exists:  
 c.execute('UPDATE "Train" SET "Seats amount"=%s WHERE "Number"=%s', (stsam, num))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def delete\_train(self, num):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {num})')  
 num\_exists = c.fetchone()[0]  
 if num\_exists:  
 c.execute('DELETE FROM "Train" WHERE "Number"=%s', (num,))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def create\_table\_ticket(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Ticket" (  
 "ID" SERIAL PRIMARY KEY,  
 "Price" integer NOT NULL,  
 "Transit\_id" integer NOT NULL,  
 "Pas\_full\_name" character varying(50) NOT NULL  
 )  
 ''')  
  
 # Check if the table exists  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Ticket')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Table does not exist, so create it  
 c.execute('''  
 CREATE TABLE "Train" (  
 "ID" SERIAL PRIMARY KEY,  
 "Price" integer NOT NULL,  
 "Transit\_id" integer NOT NULL,  
 "Pas\_full\_name" character varying(50) NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def add\_ticket(self, prc, trs\_id, pas\_nm):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 if trs\_id\_exists:  
 c.execute('INSERT INTO "Ticket" ("Price","Transit\_id","Pas\_full\_name") VALUES (%s,%s,%s)',  
 (prc, trs\_id, pas\_nm, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def update\_ticket(self, idd, prc, trs\_id, pas\_nm):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Ticket" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if trs\_id\_exists and idd\_exists:  
 c.execute('UPDATE "Ticket" SET "Price"=%s, "Transit\_id"=%s, "Pas\_full\_name"=%s WHERE "ID"=%s',  
 (prc, trs\_id, pas\_nm, idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def delete\_ticket(self, idd):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Ticket" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if idd\_exists:  
 c.execute('DELETE FROM "Ticket" WHERE "ID"=%s', (idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def create\_table\_station(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Station" (  
 "ID" SERIAL PRIMARY KEY,  
 "name" character varying(50) NOT NULL  
 )  
 ''')  
  
 # Check if the table exists  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Station')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Table does not exist, so create it  
 c.execute('''  
 CREATE TABLE "Station" (  
 "ID" SERIAL PRIMARY KEY,  
 "name" character varying(50) NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def add\_station(self, name):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Station" ("name") VALUES (%s)',  
 (name, ))  
 self.conn.commit()  
  
 def update\_station(self, idd, name):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if idd\_exists:  
 c.execute('UPDATE "Station" SET "name"=%s WHERE "ID"=%s',  
 (name, idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def delete\_station(self, idd):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {idd})')  
 idd\_exists = c.fetchone()[0]  
 if idd\_exists:  
 c.execute('DELETE FROM "Station" WHERE "ID"=%s', (idd, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def create\_table\_transit(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Transit" (  
 "Train\_num" integer NOT NULL,  
 "Station\_id" integer NOT NULL,  
 "Tab\_id" SERIAL PRIMARY KEY,  
 "Date" date NOT NULL  
 )  
 ''')  
  
 # Check if the table exists  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Transit')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Table does not exist, so create it  
 c.execute('''  
 CREATE TABLE "Transit" (  
 "Train\_num" integer NOT NULL,  
 "Station\_id" integer NOT NULL,  
 "Tab\_id" SERIAL PRIMARY KEY,  
 "Date" date NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def add\_transit(self, tr\_num, st\_id, date):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {tr\_num})')  
 tr\_num\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {st\_id})')  
 st\_id\_exists = c.fetchone()[0]  
 if tr\_num\_exists and st\_id\_exists:  
 c.execute('INSERT INTO "Transit" ("Train\_num","Station\_id","Date") VALUES (%s,%s,%s)',  
 (tr\_num, st\_id, date, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def update\_transit(self, tr\_num, st\_id, date, trs\_id):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Number" FROM "Train" WHERE "Number" = {tr\_num})')  
 tr\_num\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "ID" FROM "Station" WHERE "ID" = {st\_id})')  
 st\_id\_exists = c.fetchone()[0]  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 if trs\_id\_exists and tr\_num\_exists and st\_id\_exists:  
 c.execute('UPDATE "Transit" SET "Train\_num"=%s, "Station\_id"=%s, "Date"=%s WHERE "Tab\_id"=%s',  
 (tr\_num, st\_id, date, trs\_id, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def delete\_transit(self, trs\_id):  
 c = self.conn.cursor()  
 c.execute(f'SELECT EXISTS(SELECT "Tab\_id" FROM "Transit" WHERE "Tab\_id" = {trs\_id})')  
 trs\_id\_exists = c.fetchone()[0]  
 if trs\_id\_exists:  
 c.execute('DELETE FROM "Transit" WHERE "Tab\_id"=%s', (trs\_id, ))  
 self.conn.commit()  
 return 0  
 else:  
 return 1  
  
 def generate\_data\_train(self, count):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Train" ("Seats amount") SELECT ((RANDOM() \* 1000 + 1000)::INT) '  
 'FROM generate\_series(1,%s)', (count,))  
 self.conn.commit()  
  
 def generate\_data\_station(self, count):  
 c = self.conn.cursor()  
 for i in range(count):  
 c.execute('INSERT INTO "Station" (name) SELECT random\_names.name '  
 'FROM unnest(ARRAY[\'Kiyv\', \'Chernihiv\', \'Nizhyn\','  
 ' \'Novoselivka\', \'Sumy\', \'Studenyky\']) AS random\_names(name) '  
 'ORDER BY random() '  
 'LIMIT 1;', (count,))  
 self.conn.commit()  
  
 def generate\_data\_transit(self, count):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Transit" ("Train\_num","Station\_id","Date") SELECT '  
 'T1."Number" AS "Train\_num",'  
 'S1."ID" AS "Station\_id",'  
 'current\_timestamp - random() \* interval \'365 days\' AS "Date" '  
 'FROM '  
 '"Train" T1 '  
 'CROSS JOIN "Station" S1 '  
 'WHERE RANDOM() < 1 '  
 'LIMIT %s;', (count,))  
 self.conn.commit()  
  
 def generate\_data\_ticket(self, count):  
 c = self.conn.cursor()  
 c.execute('INSERT INTO "Ticket" ("Price", "Transit\_id", "Pas\_full\_name")'  
 'SELECT'  
 'random() \* 1000 + 1000, '  
 '"Tab\_id" AS "Transit\_id",'  
 'unnest(ARRAY[\'Alice\', \'Bob\', \'Charlie\', \'David\', \'Eva\', \'Frank\'] AS "Pas\_full\_name"'  
 'FROM "Transit")'  
 'ORDER BY random()'  
 'LIMIT 1000;', (count,))  
 self.conn.commit()

Файл view.py:

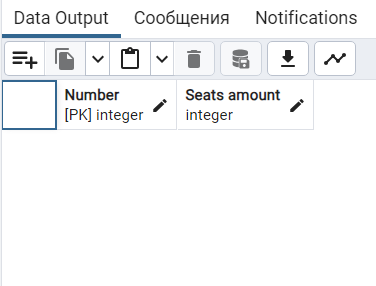
class View:  
 def show\_message(self, message):  
 print(message)  
  
 def get\_id(self):  
 return int(input("Enter Id/Num: "))  
  
 def get\_count(self):  
 return int(input("Enter Count: "))  
  
 def show\_trains(self, trains):  
 print("Trains:")  
 for train in trains:  
 print(f"Num: {train[0]}, Seats amount: {train[1]}")  
  
 def get\_train\_input(self):  
 stsamnt = input("Enter train seats amount: ")  
 return stsamnt  
  
 def show\_tickets(self, tickets):  
 print("Tickets:")  
 for ticket in tickets:  
 print(f"Id: {ticket[0]}, Price: {ticket[1]}, Transit\_id: {ticket[2]}, Pas\_full\_name: {ticket[3]}")  
  
 def get\_ticket\_input(self):  
 prc = input("Enter ticket price: ")  
 trs\_id = input("Enter ticket transit\_id: ")  
 pas\_fn = input("Enter passenger full name: ")  
 return prc, trs\_id, pas\_fn  
  
 def show\_stations(self, stations):  
 print("Stations:")  
 for station in stations:  
 print(f"Station ID: {station[0]}, Name: {station[1]}")  
  
 def get\_station\_input(self):  
 name = input("Enter station name: ")  
 return name  
  
 def show\_transits(self, transits):  
 print("Transits:")  
 for transit in transits:  
 print(f"Train Num: {transit[0]}, Station ID: {transit[1]}, Transit ID: {transit[2]} , Date: {transit[3]}")  
  
 def get\_transit\_input(self):  
 st\_id = input("Enter station id: ")  
 num = input("Enter train num: ")  
 date = input("Enter date")  
 return num, st\_id, date

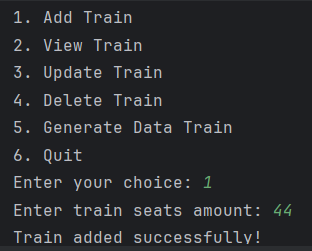
Файл controller.py:

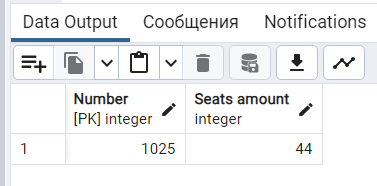
from model import Model  
from view import View  
  
  
class Controller:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
  
 def run(self):  
 while True:  
 choice = self.show\_tables()  
 if choice == '1':  
 choice = self.show\_menu\_train()  
 if choice == '1':  
 self.add\_train()  
 elif choice == '2':  
 self.view\_trains()  
 elif choice == '3':  
 self.update\_train()  
 elif choice == '4':  
 self.delete\_train()  
 elif choice == '5':  
 self.generate\_data\_train()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
 elif choice == '2':  
 choice = self.show\_menu\_ticket()  
 if choice == '1':  
 self.add\_ticket()  
 elif choice == '2':  
 self.view\_tickets()  
 elif choice == '3':  
 self.update\_ticket()  
 elif choice == '4':  
 self.delete\_ticket()  
 elif choice == '5':  
 self.generate\_data\_ticket()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
 elif choice == '3':  
 choice = self.show\_menu\_station()  
 if choice == '1':  
 self.add\_station()  
 elif choice == '2':  
 self.view\_stations()  
 elif choice == '3':  
 self.update\_station()  
 elif choice == '4':  
 self.delete\_station()  
 elif choice == '5':  
 self.generate\_data\_station()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
 elif choice == '4':  
 choice = self.show\_menu\_transit()  
 if choice == '1':  
 self.add\_transit()  
 elif choice == '2':  
 self.view\_transits()  
 elif choice == '3':  
 self.update\_transit()  
 elif choice == '4':  
 self.delete\_transit()  
 elif choice == '5':  
 self.generate\_data\_transit()  
 elif choice == '6':  
 self.model.conn.close()  
 break  
  
 def show\_tables(self):  
 self.view.show\_message("\nChoose table:")  
 self.view.show\_message("1. Train")  
 self.view.show\_message("2. Ticket")  
 self.view.show\_message("3. Station")  
 self.view.show\_message("4. Transit")  
 return input("Enter your choice: ")  
  
 def show\_menu\_train(self):  
 self.view.show\_message("\nMenu:")  
 self.view.show\_message("1. Add Train")  
 self.view.show\_message("2. View Train")  
 self.view.show\_message("3. Update Train")  
 self.view.show\_message("4. Delete Train")  
 self.view.show\_message("5. Generate Data Train")  
 self.view.show\_message("6. Quit")  
 return input("Enter your choice: ")  
  
 def show\_menu\_ticket(self):  
 self.view.show\_message("\nMenu:")  
 self.view.show\_message("1. Add Ticket")  
 self.view.show\_message("2. View Tickets")  
 self.view.show\_message("3. Update Ticket")  
 self.view.show\_message("4. Delete Ticket")  
 self.view.show\_message("5. Generate random data ticket")  
 self.view.show\_message("6. Quit")  
 return input("Enter your choice: ")  
  
 def show\_menu\_transit(self):  
 self.view.show\_message("\nMenu:")  
 self.view.show\_message("1. Add Transit")  
 self.view.show\_message("2. View Transits")  
 self.view.show\_message("3. Update Transit")  
 self.view.show\_message("4. Delete Transit")  
 self.view.show\_message("5. Generate random data transit")  
 self.view.show\_message("6. Quit")  
 return input("Enter your choice: ")  
  
 def show\_menu\_station(self):  
 self.view.show\_message("\nMenu:")  
 self.view.show\_message("1. Add Station")  
 self.view.show\_message("2. View Station")  
 self.view.show\_message("3. Update Station")  
 self.view.show\_message("4. Delete Station")  
 self.view.show\_message("5. Generate random data station")  
 self.view.show\_message("6. Quit")  
 return input("Enter your choice: ")  
  
 def add\_train(self):  
 stsamnt = self.view.get\_train\_input()  
 self.model.add\_train(stsamnt)  
 self.view.show\_message("Train added successfully!")  
  
 def view\_trains(self):  
 trains = self.model.get\_all\_attr\_table('"Train"')  
 self.view.show\_trains(trains)  
  
 def update\_train(self):  
 train\_num = self.view.get\_id()  
 stsamnt = self.view.get\_train\_input()  
 if self.model.update\_train(train\_num, stsamnt):  
 self.view.show\_message("Error, Train with such number doesn't exist!")  
 else:  
 self.view.show\_message("Train updated successfully!")  
  
 def delete\_train(self):  
 train\_num = self.view.get\_id()  
 if self.model.delete\_train(train\_num):  
 self.view.show\_message("Error, Train with such number doesn't exist!")  
 else:  
 self.view.show\_message("Train deleted successfully!")  
  
 def generate\_data\_train(self):  
 count = self.view.get\_count()  
 self.model.generate\_data\_train(count)  
  
 def add\_ticket(self):  
 prc, trs\_id, pas\_fn = self.view.get\_ticket\_input()  
 if self.model.add\_ticket(prc, trs\_id, pas\_fn):  
 self.view.show\_message("Error, Transit id doesn't exist!")  
 else:  
 self.view.show\_message("Ticket added successfully!")  
  
 def view\_tickets(self):  
 tickets = self.model.get\_all\_attr\_table('"Ticket"')  
 self.view.show\_tickets(tickets)  
  
 def update\_ticket(self):  
 idd = self.view.get\_id()  
 prc, trs\_id, pas\_fn = self.view.get\_ticket\_input()  
 if self.model.update\_ticket(idd, prc, trs\_id, pas\_fn):  
 self.view.show\_message("Error, Ticket ID or Transit ID incorrect")  
 else:  
 self.view.show\_message("Ticket updated successfully!")  
  
 def delete\_ticket(self):  
 idd = self.view.get\_id()  
 if self.model.delete\_ticket(idd):  
 self.view.show\_message("Error, no such ticket!")  
 else:  
 self.view.show\_message("Ticket deleted successfully!")  
  
 def generate\_data\_ticket(self):  
 count = self.view.get\_count()  
 self.model.generate\_data\_ticket(count)  
  
 def add\_station(self):  
 name = self.view.get\_station\_input()  
 self.model.add\_station(name)  
 self.view.show\_message("Station added successfully!")  
  
 def view\_stations(self):  
 stations = self.model.get\_all\_attr\_table('"Station"')  
 self.view.show\_stations(stations)  
  
 def update\_station(self):  
 st\_id = self.view.get\_id()  
 name = self.view.get\_station\_input()  
 if self.model.update\_station(st\_id, name):  
 self.view.show\_message("Error, no such station!")  
 else:  
 self.view.show\_message("Station updated successfully!")  
  
 def delete\_station(self):  
 st\_id = self.view.get\_id()  
 if self.model.delete\_station(st\_id):  
 self.view.show\_message("Error, no such station!")  
 else:  
 self.view.show\_message("Station deleted successfully!")  
  
 def generate\_data\_station(self):  
 count = self.view.get\_count()  
 self.model.generate\_data\_station(count)  
  
 def add\_transit(self):  
 tr\_num, st\_id, date = self.view.get\_transit\_input()  
 if self.model.add\_transit(tr\_num, st\_id, date):  
 self.view.show\_message("Error, one of id doesn't exist!")  
 else:  
 self.view.show\_message("Transit added successfully!")  
  
 def view\_transits(self):  
 transits = self.model.get\_all\_attr\_table('"Transit"')  
 self.view.show\_transits(transits)  
  
 def update\_transit(self):  
 trs\_id = self.view.get\_id()  
 tr\_num, st\_id, date = self.view.get\_transit\_input()  
 if self.model.update\_transit(tr\_num, st\_id, date, trs\_id):  
 self.view.show\_message("Error, Transit id or Train num or Station ID incorrect!")  
 else:  
 self.view.show\_message("Transit updated successfully!")  
  
 def delete\_transit(self):  
 trs\_id = self.view.get\_id()  
 if self.model.delete\_transit(trs\_id):  
 self.view.show\_message("Error no such Transit id!")  
 else:  
 self.view.show\_message("Transit deleted successfully!")  
  
 def generate\_data\_transit(self):  
 count = self.view.get\_count()  
 self.model.generate\_data\_transit(count)

**Результати виконання програми**

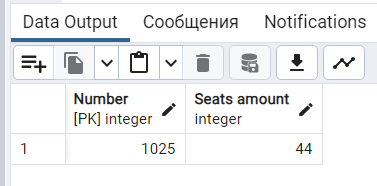
Вставка даних:

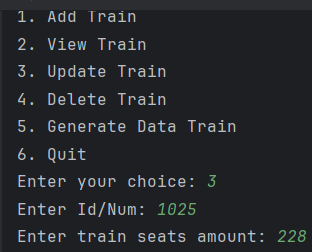


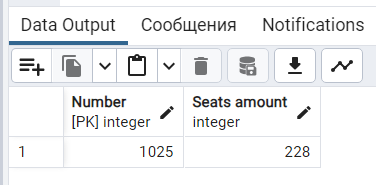




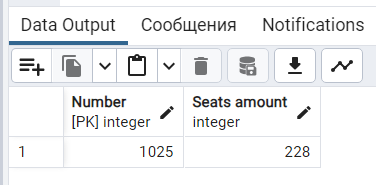
Редагування даних:

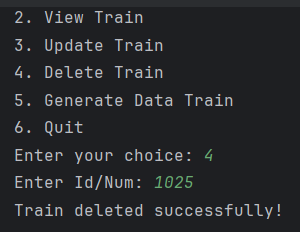


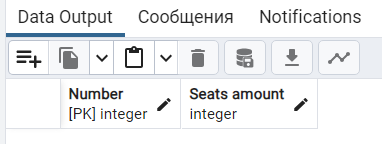




Видалення даних:







Генерування даних:

