



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

Факультет прикладної математики

**Кафедра системного програмування і спеціалізованих комп'ютерних
систем**

Розрахунково-графічна робота

з дисципліни
«Бази даних і засоби управління»

Виконав: студент III курсу

ФПМ групи КВ-31

Ільницький П.П.

Перевірив:

Київ – 2025

Загальне завдання: Реалізувати консольний додаток для взаємодії з базою даних згідно шаблону MVC (Model-View-Controller). Додаток повинен забезпечувати виконання операцій перегляду, внесення, редагування та вилучення даних (CRUD).

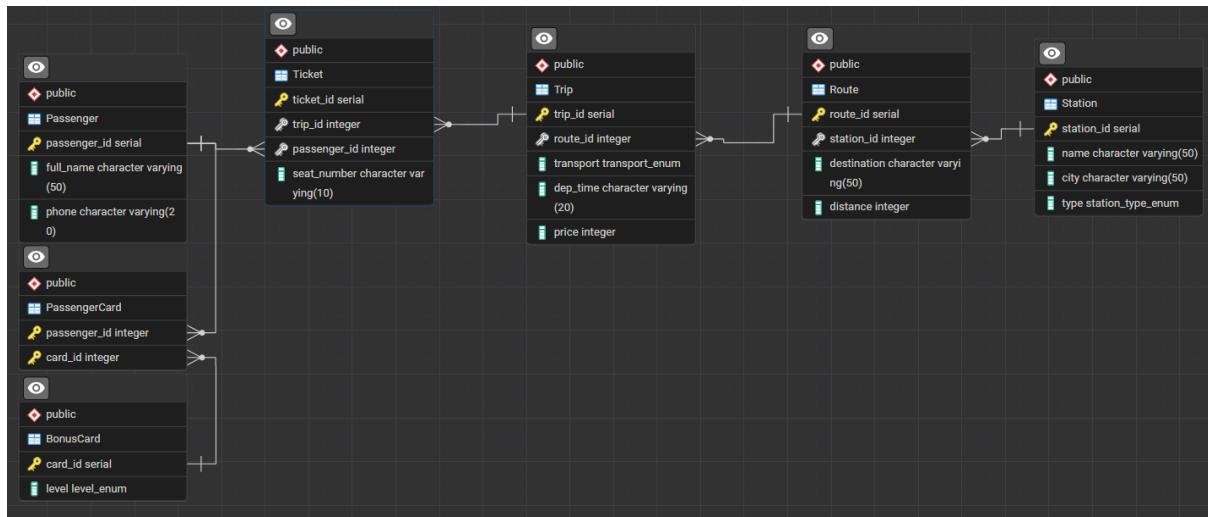
Деталізовані вимоги:

1. **Контроль цілісності:** Реалізувати перевірку типів даних та обробку помилок зовнішніх ключів (Foreign Key) при видаленні батьківських записів та вставці дочірніх .
2. **Генерація даних:** Забезпечити автоматичне наповнення таблиць великим обсягом даних (100 000+ записів) за допомогою SQL-запитів та функції random() . Кількість даних вводиться користувачем.
3. **Пошук:** Реалізувати пошук за декількома атрибутами з використанням фільтрації (WHERE) та групування (GROUP BY), із виведенням часу виконання запиту .
4. **Технології:** Використання мови SQL без ORM, бібліотека psycopg2 .

Структура бази даних:

1. **Station (Станція):** Батьківська сутність. Зберігає назву, місто та тип вокзалу (ENUM: Bus, Railway, Airport).
2. **Route (Маршрут):** Дочірня до Station. Містить інформацію про пункт призначення та відстань.
3. **Trip (Рейс):** Дочірня до Route. Містить дані про транспорт, час відправлення та ціну.
4. **Passenger (Пасажир):** Зберігає персональні дані клієнтів (ім'я, телефон).
5. **BonusCard (Бонусна картка):** Довідник типів карток (Silver, Gold, Platinum).
6. **PassengerCard:** Проміжна таблиця для зв'язку 1:1 (або 0:1) між пасажиром та карткою.
7. **Ticket (Квиток):** Основна сутність, що фіксує факт покупки. Пов'язує рейс та пасажира.

Схема даних (ER-діаграма):



4. Схема меню користувача

Програма має консольний інтерфейс користувача з наступними функціональними блоками:

1. Згенерувати базу даних (SQL Random):

- Запитує у користувача кількість записів.
- Повністю очищає базу даних (TRUNCATE).
- Генерує зв'язані тестові дані SQL-скриптом.
- Виводить час виконання операції.

2. Показати квитки (Top 50):

- Виводить таблицю з останніми проданими квитками, об'єднуючи дані з 5 таблиць (JOIN).

3. Додати станцію:

- Дозволяє ввести нову станцію (Назва, Місто, Тип).

4. Видалити станцію:

- Приймає ID станції.
- Функціональність: Перевіряє наявність маршрутів. Якщо маршрути є — блокує видалення і виводить попередження (захист цілісності). Якщо немає — видаляє.

5. Пошук рейсів (з таймером):

- Фільтрує рейси за ціною та назвою міста.
- Групує результати за напрямком.
- Виводить статистику та час виконання запиту у мілісекундах.

6. Редагувати станцію (UPDATE):

- Дозволяє змінити назву, місто або тип існуючої станції за її ID.

7. Додати маршрут (Тест Foreign Key):

- Додає новий маршрут.
- Функціональність: Демонструє перехоплення системної помилки PostgreSQL, якщо введено ID неіснуючої станції.

Використані засоби

- **Мова програмування:** Python 3.12.
- **СУБД:** PostgreSQL 16.
- **Бібліотека взаємодії з БД:** psycopg2 (реалізація драйвера PostgreSQL для Python).
- **Середовище розробки:** PyCharm Community Edition.
- **Середовище адміністрування БД:** PgAdmin 4.

Виконання пункту №1 деталізованого завдання

У розробленій системі реалізовано механізм контролю посилальної цілісності (Referential Integrity) між таблицями Station (батьківська сутність) та Route (дочірня сутність, зв'язок 1:N). Згідно із завданням, перед видаленням запису з батьківської таблиці виконується перевірка на наявність залежних записів у дочірній таблиці.

Генеруємо базу даних:

```
--- ЗАЛІЗНИЧНА КАСА (RGR) ---
1. Згенерувати базу даних (SQL Random)
2. Показати квитки (Top 50)
3. Додати станцію
4. Видалити станцію (з перевіркою зв'язків)
5. Пошук рейсів (з таймером)
6. Редагувати станцію (UPDATE)
7. Додати маршрут (Тест Foreign Key)
0. Вихід
Ваш вибір: 1
Скільки пасажирів/квитків згенерувати? (рекомендовано 1000 - 100000): 100000

[INFO]: Генерація 100000 записів... Це може зайняти час.

[INFO]: База успішно перезаписана!

[INFO]: Час генерації: 26.04 сек
```

Сценарій 1: Спроба видалення запису, що має залежні дані

При спробі видалити станцію, яка вже використовується у маршрутах (Route), програма перехоплює цю операцію та забороняє її виконання.

| Ваш вибір: 2 | | | | | |
|--------------|--------------|---------|-------------|-------|-----------|
| ticket_id | full_name | city | destination | price | transport |
| 1 | Марія Коваль | Вінниця | Берлін | 4462 | Plane |

```
--- ЗАЛІЗНИЧНА КАСА (RGR) ---
1. Згенерувати базу даних (SQL Random)
2. Показати квитки (Top 50)
3. Додати станцію
4. Видалити станцію (з перевіркою зв'язків)
5. Пошук рейсів (з таймером)
6. Редагувати станцію (UPDATE)
7. Додати маршрут (Тест Foreign Key)
0. Вихід
Ваш вибір: 4
Введіть ID станції для видалення: 1

[ERROR]: Неможливо видалити станцію ID 1: існують пов'язані маршрути!
```

Пояснення причин помилки: Видалення даної станції неможливе, оскільки її первинний ключ (station_id) використовується як зовнішній ключ (Foreign Key) у таблиці Route. Пряме видалення призвело б до появи "сирітських" записів у таблиці маршрутів або порушення логіки бази даних.

Сценарій 2: Успішне видалення запису

Якщо у станції немає залежних маршрутів (наприклад, новостворена станція), операція видалення проходить успішно.

```
--- ЗАЛІЗНИЧНА КАСА (RGR) ---
1. Згенерувати базу даних (SQL Random)
2. Показати квитки (Top 50)
3. Додати станцію
4. Видалити станцію (з перевіркою зв'язків)
5. Пошук рейсів (з таймером)
6. Редагувати станцію (UPDATE)
7. Додати маршрут (Тест Foreign Key)
0. Вихід
Ваш вибір: 3
Введіть дані нової станції:
Назва (напр. Вокзал Північний): Вокзал Північний
Місто: Київ
Тип (Bus Station, Railway Station, Airport): Railway Station

[INFO]: Станцію додано.
```

| | station_id [PK] integer | name character varying (50) | city character varying (50) | type station_type_enum |
|---|----------------------------|--------------------------------|--------------------------------|---------------------------|
| 1 | 51 | Вокзал Північний | Київ | Railway Station |

```
--- ЗАЛІЗНИЧНА КАСА (RGR) ---
1. Згенерувати базу даних (SQL Random)
2. Показати квитки (Top 50)
3. Додати станцію
4. Видалити станцію (з перевіркою зв'язків)
5. Пошук рейсів (з таймером)
6. Редагувати станцію (UPDATE)
7. Додати маршрут (Тест Foreign Key)
0. Вихід
Ваш вибір: 4
Введіть ID станції для видалення: 51

[INFO]: Станцію 51 видалено.
```

| | station_id [PK] integer | name character varying (50) | city character varying (50) | type station_type_enum |
|---|----------------------------|--------------------------------|--------------------------------|---------------------------|
| 1 | 50 | Вокзал Східний | Харків | Airport |

Результат: Запис фізично вилучається з таблиці Station за допомогою команди DELETE, оскільки це не порушує цілісність бази даних.

Контроль цілісності при вставці даних (INSERT)

При додаванні нових записів у дочірню таблицю Route необхідно контролювати наявність відповідного запису у батьківській таблиці Station. Якщо користувач намагається створити маршрут, посилаючись на неіснуючу станцію (station_id), система повинна заблокувати цю операцію та повідомити про помилку.

Сценарій: Користувач намагається додати новий маршрут, вказуючи station_id = 999, хоча станції з таким ідентифікатором у базі даних не існує.

```
--- ЗАЛІЗНИЧНА КАСА (RGR) ---
1. Згенерувати базу даних (SQL Random)
2. Показати квитки (Top 50)
3. Додати станцію
4. Видалити станцію (з перевіркою зв'язків)
5. Пошук рейсів (з таймером)
6. Редагувати станцію (UPDATE)
7. Додати маршрут (Тест Foreign Key)
0. Вихід
Ваш вибір: 7
--- Новий маршрут ---
Увага! station_id має існувати в базі Station.
Введіть ID станції (FK): 999
Пункт призначення: Берлін
відстань (км): 450

[ERROR]: Інша помилка: ОШИБКА: INSERT или UPDATE в таблице "Route" нарушает ограничение внешнего ключа "station_fk"
DETAIL: Ключ (station_id)=(999) отсутствует в таблице "Station".
```

Аналіз помилки: СУБД PostgreSQL при спробі виконання запиту INSERT перевіряє обмеження зовнішнього ключа (FOREIGN KEY). Оскільки запис у батьківській таблиці відсутній, виникає помилка foreign_keyViolation. Програмний додаток перехоплює цю системну помилку у блоці try...except і виводить користувачеві зрозуміле повідомлення замість аварійного завершення роботи.

Лістинг програмного коду реалізації (Python + SQL):

Реалізація SQL-запиту в моделі (*model.py*):

```
def add_route(self, station_id, destination, distance):
    query = """
        INSERT INTO public."Route" (station_id, destination, distance)
        VALUES (%s, %s, %s)
    """
    with self.conn.cursor() as cur:
        cur.execute(query, (station_id, destination, distance))
```

Обробка помилки в контролери (*controller.py*):

```
try:
    self.model.add_route(s_id, dest, dist)
    self.view.show_message("Маршрут успішно додано!")
except Exception as e:

    error_msg = str(e)
    if "foreign key constraint" in error_msg or "violates foreign key" in error_msg:
        self.view.show_error(f"Неможливо додати маршрут: Станції з ID={s_id} не існує!")
        self.view.show_error(f"Технічні деталі: {e}")
    else:
        self.view.show_error(f"Інша помилка: {e}")
```

Виконання пункту №2:

Для наповнення бази даних тестовою інформацією реалізовано механізм пакетної генерації даних за допомогою мови SQL. Генерація ініціюється користувачем з консолі, де він вказує бажану кількість записів. Програма очищає попередні дані та створює нові, зберігаючи логічні зв'язки між таблицями.

Data Output Messages Notifications

Showing rows: 1 to 100000 | Page No: 1 of 1 | Back | Next |

| | ID Квитка | Місце | Пасажир | Телефон | Рівень картки | Транспорт | Час відрп. | Ціна | Маршрут |
|-------|-----------|-------|------------------|---------------|---------------|-----------|--------------|----------|-------------------|
| | ID | Код | Ім'я | Номер | Тип | Середня | Відправлення | Вартість | Місця |
| 42641 | 42641 | 50A | Дмитро Шевченко | +380757729642 | Nemas | Plane | 18:45 | 4014 грн | Харків -> Лондон |
| 42642 | 42642 | 31A | Іван Коваль | +380984353421 | Standard | Train | 10:25 | 2345 грн | Одеса -> Прага |
| 42643 | 42643 | 4A | Анна Коваль | +380745474019 | Nemas | Plane | 19:47 | 4194 грн | Вінниця -> Прага |
| 42644 | 42644 | 20A | Дмитро Бойко | +380703261580 | Nemas | Bus | 00:00 | 207 грн | Дніпро -> Прага |
| 42645 | 42645 | 50A | Анна Коваль | +380737863026 | Nemas | Plane | 22:55 | 4807 грн | Дніпро -> Берлін |
| 42646 | 42646 | 9A | Іван Бойко | +380942507017 | Gold | Train | 10:27 | 2482 грн | Харків -> Лондон |
| 42647 | 42647 | 56A | Олександр Бойко | +380880267802 | Standard | Bus | 06:16 | 1588 грн | Харків -> Лондон |
| 42648 | 42648 | 22A | Олександр Коваль | +380676435575 | Nemas | Plane | 16:41 | 3676 грн | Київ -> Варшава |
| 42649 | 42649 | 7A | Дмитро Бойко | +380854834775 | Nemas | Bus | 01:04 | 585 грн | Київ -> Варшава |
| 42650 | 42650 | 26A | Іван Коваль | +380801711604 | Nemas | Plane | 23:59 | 5134 грн | Дніпро -> Берлін |
| 42651 | 42651 | 28A | Іван Коваль | +380891190247 | Gold | Train | 09:22 | 2079 грн | Львів -> Прага |
| 42652 | 42652 | 17A | Олександр Коваль | +380994246000 | Standard | Bus | 07:18 | 1725 грн | Київ -> Прага |
| 42653 | 42653 | 12A | Марія Коваль | +380989865946 | Nemas | Bus | 01:02 | 432 грн | Львів -> Варшава |
| 42654 | 42654 | 26A | Марія Бойко | +380889439153 | Standard | Train | 10:26 | 2427 грн | Одеса -> Лондон |
| 42655 | 42655 | 32A | Анна Коваль | +380964036852 | Nemas | Train | 12:32 | 2892 грн | Вінниця -> Берлін |
| 42656 | 42656 | 42A | Анна Шевченко | +380824918520 | Silver | Plane | 16:42 | 3716 грн | Київ -> Варшава |
| 42657 | 42657 | 52A | Марія Бойко | +380992266268 | Standard | Bus | 05:12 | 1245 грн | Львів -> Берлін |
| 42658 | 42658 | 60A | Дмитро Бойко | +380900970544 | Gold | Train | 09:24 | 2205 грн | Дніпро -> Варшава |
| 42659 | 42659 | 23A | Дмитро Бойко | +380823864435 | Nemas | Bus | 02:05 | 636 грн | Харків -> Лондон |
| 42660 | 42660 | 54A | Дмитро Шевченко | +380846962734 | Nemas | Train | 15:38 | 3418 грн | Одеса -> Варшава |
| 42661 | 42661 | 9A | Анна Коваль | +380981780876 | Nemas | Plane | 23:58 | 5103 грн | Львів -> Лондон |

Total rows: 100000 | Query complete 00:00:00.693 | CRLF | Ln 8, Col 46

SQL-запити, що ілюструють генерацію при визначеннях вхідних параметрах:

```
TRUNCATE "Ticket", "PassengerCard", "BonusCard", "Trip", "Route",
"Passenger", "Station" RESTART IDENTITY CASCADE;

        INSERT INTO public."BonusCard" (level) VALUES ('Silver'),
('Gold'), ('Standard');

        INSERT INTO public."Station" (name, city, type)
        SELECT 'Вокзал ' || (ARRAY['Центральний', 'Південний',
'Sхідний']) [floor(random()*3+1)],
        (ARRAY['Київ', 'Львів', 'Одеса', 'Харків', 'Дніпро',
'Вінниця']) [floor(random()*6+1)],

(enum_range(NULL::station_type_enum)) [floor(random()*3 + 1)]
        FROM generate_series(1, 50);

        INSERT INTO public."Route" (station_id, destination,
distance)
        SELECT s.station_id, (ARRAY['Варшава', 'Берлін', 'Прага',
'Лондон']) [floor(random()*4 + 1)], floor(random() * 2000 + 100)::int
        FROM public."Station" s CROSS JOIN generate_series(1, 4)
ORDER BY random() LIMIT 200;
```

```

        INSERT INTO public."Trip" (route_id, transport, dep_time,
price)
            SELECT r.route_id,
(enum_range(NULL::transport_enum)) [floor(random()*3 + 1)],
                lpad(floor(random()*24)::text, 2, '0') || ':' ||
lpad(floor(random()*60)::text, 2, '0'), floor(random() * 5000 +
200)::int
                    FROM public."Route" r CROSS JOIN generate_series(1, 5)
ORDER BY random();

        INSERT INTO public."Passenger" (full_name, phone)
            SELECT (ARRAY['Олександр', 'Іван', 'Дмитро', 'Анна',
'Mарія']) [floor(random()*5+1)] || ' ' || (ARRAY['Шевченко', 'Бойко',
'Коваль']) [floor(random()*3+1)],
'+380' || floor(random()*(999999999-660000000) +
660000000)::text
                FROM generate_series(1, {count_passengers});

        INSERT INTO public."PassengerCard" (passenger_id, card_id)
            SELECT p.passenger_id, c.card_id FROM public."Passenger" p
            CROSS JOIN LATERAL (SELECT card_id FROM public."BonusCard"
ORDER BY random() + (p.passenger_id * 0) LIMIT 1) c
                WHERE random() < 0.4;

        INSERT INTO public."Ticket" (trip_id, passenger_id,
seat_number)
            SELECT t.trip_id, p.passenger_id, floor(random()*60 +
1)::text || 'A'
                FROM public."Passenger" p
                CROSS JOIN LATERAL (SELECT trip_id FROM public."Trip" ORDER
BY random() + (p.passenger_id * 0) LIMIT 1) t;

```

Виконання пункту №3

Для аналізу даних реалізовано функцію складного пошуку, яка об'єднує дані з чотирьох таблиць (Ticket, Trip, Route, Station). Користувач вводить критерії фільтрації з клавіатури (мінімальна ціна та частина назви міста відправлення). Система виконує SQL-запит, який не лише фільтрує рядки, а й групуює їх для отримання статистики.

- ```
--- ЗАЛІЗНИЧНА КАСА (RGR) ---
1. Згенерувати базу даних (SQL Random)
2. Показати квитки (Top 50)
3. Додати станцію
4. Видалити станцію (з перевіркою зв'язків)
5. Пошук рейсів (з таймером)
6. Редагувати станцію (UPDATE)
7. Додати маршрут (Тест Foreign Key)
0. Вихід
```

Ваш вибір: 5

Мінімальна ціна квитка: 1000

Частина назви міста відправлення: Ки

| Напрямок | Транспорт | Продано квитків | Середня ціна |
|----------|-----------|-----------------|--------------|
|----------|-----------|-----------------|--------------|

|         |       |      |         |
|---------|-------|------|---------|
| Варшава | Plane | 1985 | 4470.62 |
| Прага   | Plane | 1447 | 4469.87 |
| Лондон  | Plane | 939  | 4443.84 |
| Берлін  | Plane | 1891 | 4337.85 |
| Варшава | Train | 1027 | 2784.98 |
| Лондон  | Train | 700  | 2710.52 |
| Берлін  | Train | 1636 | 2702.19 |
| Прага   | Train | 1513 | 2565.76 |
| Лондон  | Bus   | 321  | 1572.94 |
| Варшава | Bus   | 624  | 1538.42 |
| Прага   | Bus   | 502  | 1373.21 |
| Берлін  | Bus   | 630  | 1272.34 |

[INFO]: Час виконання запиту: 34.45 мс

**SQL-запит пошуку:** Згідно з вимогами, запит містить оператори WHERE (фільтрація) та GROUP BY (групування). Використовується агрегатна функція COUNT() для підрахунку кількості проданих квитків та AVG() для обчислення середньої ціни.

```

def search_complex(self, min_price, city_part):
 query = """
 SELECT
 r.destination AS "Напрямок",
 tr.transport AS "Транспорт",
 COUNT(t.ticket_id) AS "Продано квитків",
 ROUND(AVG(tr.price), 2) AS "Середня ціна"
 FROM public."Ticket" t
 JOIN public."Trip" tr ON t.trip_id = tr.trip_id
 JOIN public."Route" r ON tr.route_id = r.route_id
 JOIN public."Station" s ON r.station_id = s.station_id
 WHERE tr.price >= %s AND s.city ILIKE %s
 GROUP BY r.destination, tr.transport
 ORDER BY "Середня ціна" DESC
 """
 with self.conn.cursor() as cur:
 cur.execute(query, (min_price, f'%{city_part}%'))
 return cur.fetchall(), [desc[0] for desc in cur.description]

```

**Аналіз продуктивності:** Для оцінки швидкодії роботи бази даних на великих обсягах інформації (100 000 записів) у контролері реалізовано замір часу виконання запиту за допомогою модуля time. Результат вимірювання виводиться користувачеві відразу після таблиці з даними.

#### Виконання пункту №4:

Модуль model.py є ключовим компонентом архітектури MVC (Model-View-Controller) у розробленому додатку. Він відповідає за безпосередню взаємодію з системою управління базами даних PostgreSQL. У класі Model інкапсульовано всі SQL-запити, параметри підключення та логіку обробки транзакцій. Інші частини програми (View, Controller) не мають прямого доступу до бази даних, а звертаються до методів цього класу.

#### Лістинг коду модуля model.py:

```

import psycopg2

from db_config import db_params

```

```
class Model:

 def __init__(self):

 self.conn = None

 try:

 self.conn = psycopg2.connect(**db_params)

 self.conn.autocommit = True

 except Exception as e:

 print(f"Помилка підключення: {e}")

 def __del__(self):

 if self.conn:

 self.conn.close()

 def generate_data(self, count_passengers):

 sql = """\

 TRUNCATE "Ticket", "PassengerCard", "BonusCard", "Trip",
 "Route", "Passenger", "Station" RESTART IDENTITY CASCADE;

 INSERT INTO public."BonusCard" (level) VALUES ('Silver'),
 ('Gold'), ('Standard');

 INSERT INTO public."Station" (name, city, type)

 SELECT 'Вокзал' || (ARRAY['Центральний', 'Південний',
 'Східний']) [floor(random()*3+1)],

 (ARRAY['Київ', 'Львів', 'Одеса', 'Харків', 'Дніпро',
 'Вінниця']) [floor(random()*6+1)],

 '"""
```

```

(enum_range(NULL::station_type_enum)) [floor(random()*3 + 1)]

FROM generate_series(1, 50);

INSERT INTO public."Route" (station_id, destination,
distance)

SELECT s.station_id, (ARRAY['Варшава', 'Берлін', 'Прага',
'Лондон']) [floor(random()*4 + 1)], floor(random() * 2000 + 100)::int

FROM public."Station" s CROSS JOIN generate_series(1, 4)
ORDER BY random() LIMIT 200;

INSERT INTO public."Trip" (route_id, transport, dep_time,
price)

SELECT r.route_id,
(enum_range(NULL::transport_enum)) [floor(random()*3 + 1)],

lpad(floor(random()*24)::text, 2, '0') || ':' ||
lpad(floor(random()*60)::text, 2, '0'), floor(random() * 5000 +
200)::int

FROM public."Route" r CROSS JOIN generate_series(1, 5)
ORDER BY random();

INSERT INTO public."Passenger" (full_name, phone)

SELECT (ARRAY['Олександр', 'Іван', 'Дмитро', 'Анна',
'Марія']) [floor(random()*5+1)] || ' ' || (ARRAY['Шевченко', 'Бойко',
'Коваль']) [floor(random()*3+1)],

'+380' || floor(random()*(999999999-660000000) +
660000000)::text

FROM generate_series(1, {count_passengers});

INSERT INTO public."PassengerCard" (passenger_id, card_id)

```

```

SELECT p.passenger_id, c.card_id FROM public."Passenger" p
 CROSS JOIN LATERAL (SELECT card_id FROM public."BonusCard"
ORDER BY random() + (p.passenger_id * 0) LIMIT 1) c
 WHERE random() < 0.4;

INSERT INTO public."Ticket" (trip_id, passenger_id,
seat_number)
 SELECT t.trip_id, p.passenger_id, floor(random()*60 +
1)::text || 'A'
 FROM public."Passenger" p
 CROSS JOIN LATERAL (SELECT trip_id FROM public."Trip" ORDER
BY random() + (p.passenger_id * 0) LIMIT 1) t;
"""

with self.conn.cursor() as cur:
 cur.execute(sql)

def get_all_tickets(self, limit=50):
 query = """
 SELECT t.ticket_id, p.full_name, s.city, r.destination,
tr.price, tr.transport
 FROM public."Ticket" t
 JOIN public."Passenger" p ON t.passenger_id =
p.passenger_id
 JOIN public."Trip" tr ON t.trip_id = tr.trip_id
 JOIN public."Route" r ON tr.route_id = r.route_id
 JOIN public."Station" s ON r.station_id = s.station_id
 """

```

```
LIMIT %s

"""

 with self.conn.cursor() as cur:

 cur.execute(query, (limit,))

 return cur.fetchall(), [desc[0] for desc in
cur.description]

def add_station(self, name, city, s_type):

 with self.conn.cursor() as cur:

 cur.execute('INSERT INTO public."Station" (name, city,
type) VALUES (%s, %s, %s)', (name, city, s_type))

def delete_station(self, station_id):

 with self.conn.cursor() as cur:

 cur.execute('SELECT count(*) FROM public."Route" WHERE
station_id = %s', (station_id,))

 if cur.fetchone()[0] > 0:

 raise Exception(f"Неможливо видалити станцію ID
{station_id}: існують пов'язані маршрути!")

 cur.execute('DELETE FROM public."Station" WHERE station_id
= %s', (station_id,))

def search_complex(self, min_price, city_part):
```

```
query = """

 SELECT

 r.destination AS "Напрямок",

 tr.transport AS "Транспорт",

 COUNT(t.ticket_id) AS "Продано квитків",

 ROUND(AVG(tr.price), 2) AS "Середня ціна"

 FROM public."Ticket" t

 JOIN public."Trip" tr ON t.trip_id = tr.trip_id

 JOIN public."Route" r ON tr.route_id = r.route_id

 JOIN public."Station" s ON r.station_id = s.station_id

 WHERE tr.price >= %s AND s.city ILIKE %

 GROUP BY r.destination, tr.transport

 ORDER BY "Середня ціна" DESC

"""

with self.conn.cursor() as cur:

 cur.execute(query, (min_price, f'%{city_part}%'))

 return cur.fetchall(), [desc[0] for desc in
cur.description]

def update_station(self, station_id, new_name, new_city, new_type):

 query = """

 UPDATE public."Station"

 SET name = %s, city = %s, type = %s

 WHERE station_id = %s

 """
```

```

"""
with self.conn.cursor() as cur:

 cur.execute(query, (new_name, new_city, new_type,
station_id))

if cur.rowcount == 0:

 raise Exception(f"Станції з ID {station_id} не
знайдено.")

def add_route(self, station_id, destination, distance):

 query = """

 INSERT INTO public."Route" (station_id, destination,
distance)

 VALUES (%s, %s, %s)

"""
with self.conn.cursor() as cur:

 cur.execute(query, (station_id, destination, distance))

```

## Опис функцій модуля:

- \_\_init\_\_**: Виконує підключення до бази даних PostgreSQL за допомогою бібліотеки psycopg2, використовуючи параметри з файлу конфігурації. Встановлює режим autocommit=True для автоматичного збереження транзакцій.
- generate\_data(count\_passengers)**: Реалізує пакетну генерацію тестових даних. Виконує SQL-скрипт, який очищає таблиці та наповнює їх псевдовипадковими даними з дотриманням логічної цілісності. Кількість пасажирів передається як аргумент.

3. **get\_all\_tickets(limit)**: Отримує список останніх проданих квитків. Використовує оператори JOIN для об'єднання даних з 5 таблиць та повертає результат для відображення у View.
4. **add\_station(...)**: Додає новий запис у таблицю Station.
5. **delete\_station(station\_id)**: Виконує видалення станції за її ID. Перед видаленням здійснює перевірку на наявність залежних записів у таблиці Route. Якщо такі записи існують, генерує виняток, забороняючи видалення.
6. **search\_complex(...)**: Виконує аналітичний запит для пошуку та групування даних. Фільтрує рейси за ціною та містом, групует їх за напрямком і транспортом, а також обчислює агрегатні значення (кількість продажів, середня ціна).
7. **update\_station(...)**: Оновлює дані існуючої станції (UPDATE). Перевіряє атрибут rowcount, щоб переконатися, що запис дійсно існував і був змінений.
8. **add\_route(...)**: Додає новий маршрут у дочірню таблицю. Якщо передано неіснуючий station\_id, база даних повертає помилку зовнішнього ключа, яка потім обробляється у контролері.