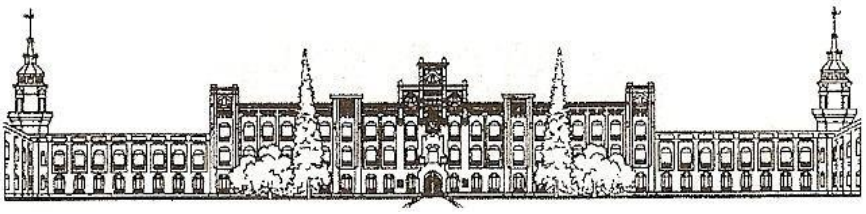

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Розрахунково-графічна робота

з дисципліни «Бази даних»

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

Виконав студент групи: КВ-33

ПІБ: Щербатюк Є. О.

Перевірив: Павловський В. І.

Київ 2025

Мета: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Виконання роботи:

Нижче наведені сутності предметної області «Система обліку автомобільного парку компанії» та зв'язки між ними.

Сутності предметної області:

1. **Автомобіль (Car)** – представляє транспортні засоби компанії, які використовуються для вантажних перевезень:
   - Атрибути: VIN (унікальний номер кузова), реєстраційний номер автомобіля, марка, вантажопідйомність.

2. **Водій (Driver)** – представляє працівників, які мають водійське посвідчення та керують транспортним засобом.
   - Атрибути: номер водійського посвідчення, ім'я, прізвище, категорія.

3. **Маршрут (Route) –** ця сутність описує шлях перевезення вантажу:
   - Атрибути: пункт відправлення, пункт призначення, відстань.

4. **Клієнт (Customer)** – ця сутність представляє собою компанію, яка замовляє вантажні перевезення:
   - Атрибути: назва компанії, телефон, email.
5. **Рейс (Trip) –** це окрема сутність, яка описує факт перевезення і через зовнішні ключі реалізує зв'язок між автомобілем, водієм, клієнтом і маршрутом.
   - Атрибути: дата виїзду, дата прибуття, дата повернення, опис вантажу, вага вантажу.
6. **Обслуговування (Service) –** ця сутність є записом про технічне обслуговування автомобілів.
   - Атрибут: дата обслуговування, опис виконаних робіт, вартість обслуговування.

Зв'язки між стуностями предметної області:

Центральною сутністю моделі є Рейс (Trip). Вона має власні атрибути (дати виїзду, прибуття та повернення, опис і вагу вантажу) і через зовнішні ключі реалізує зв'язок між чотирма іншими сутностями: Автомобіль (Car), Водій (Driver), Клієнт (Customer) та Маршрут (Route). Таким чином, Рейс виступає не набором окремих зв'язків, а єдиним комплексним зв'язком, що фіксує факт конкретного перевезення.

Додатково в моделі передбачено зв'язки 1:N:

- Car 1:N Service – один автомобіль може мати кілька записів про обслуговування.
- Customer 1:N Trip – один клієнт може замовити кілька рейсів.
- Driver 1:N Trip – один водій може виконати кілька рейсів.

- Route 1:N Trip – один маршрут може бути використаний у кількох рейсах.

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

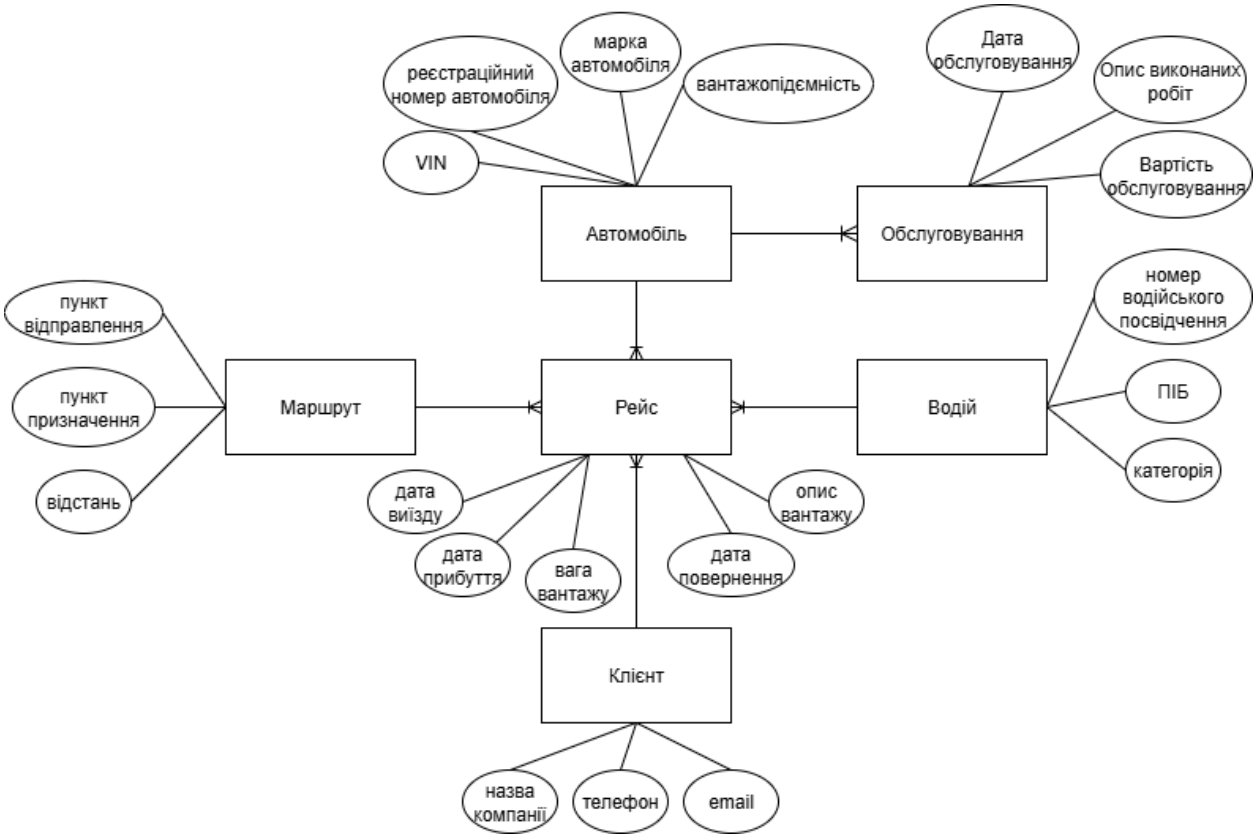

Рисунок 1 – ER-діаграма, побудована за нотацією” Пташина лапка”

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

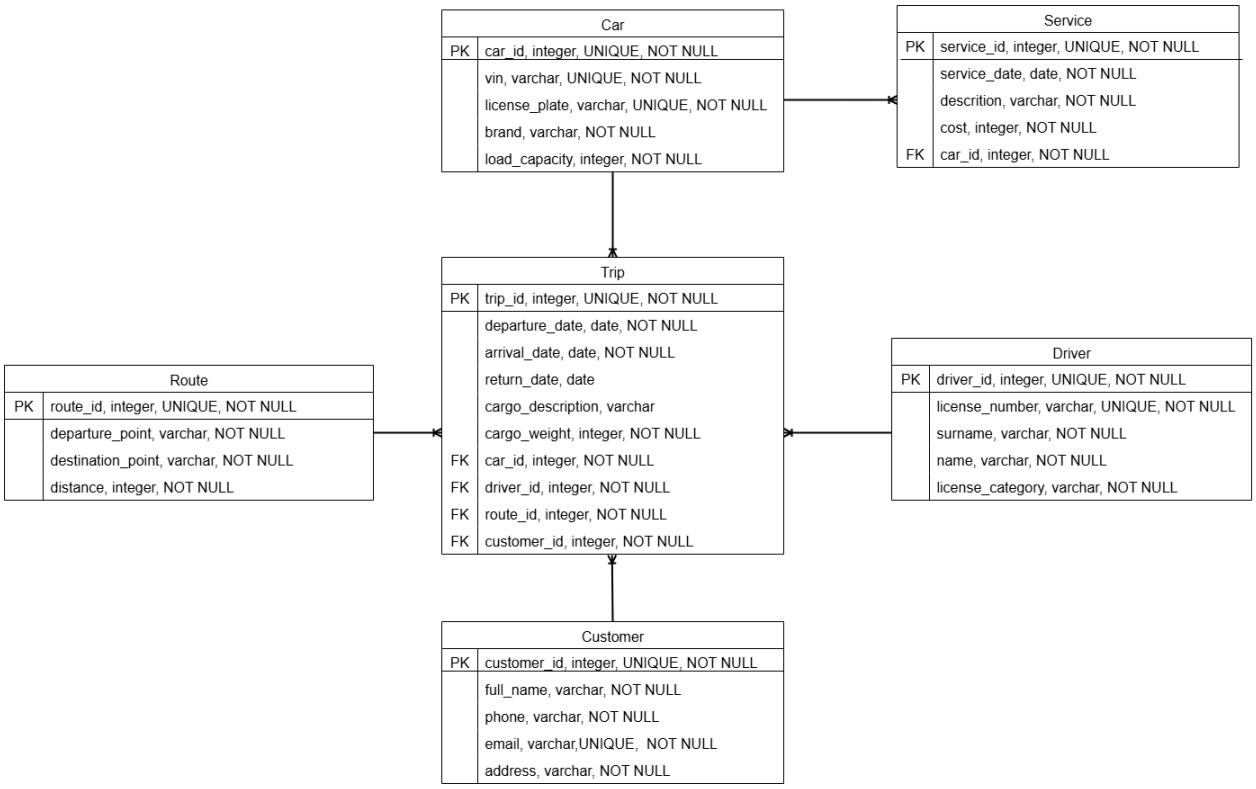

Рисунок 2 – Схема бази даних

Середовище та компоненти розробки

У процесі розробки була використана мова програмування Python, середовище розробки Pycharm, а також була використана бібліотека psycopg3, яка надає API для взаємодії з базою даних PostgreSQL.

Шаблон проектування

Модель-представлення-контролер (MVC) – це шаблон проектування, що використовується у програмі. Кожен компонент відповідає за певну функціональну частину:

1. Модель (Model) – це клас, що відображає логіку роботи з даними, обробляє всі операції з даними, такі як додавання, оновлення, вилучення.
2. Представлення (View) – це клас, через який користувач взаємодіє з програмою. У даному випадку, консольний інтерфейс, який відображає дані для користувача та зчитує їх з екрану.
3. Контролер (Controller) – це клас, який відповідає за зв’язок між користувачем і системою. Він приймає введені користувачем дані та обробляє їх. В залежності від результатів, викликає відповідні дії з Model або View.

Структура програми та її опис

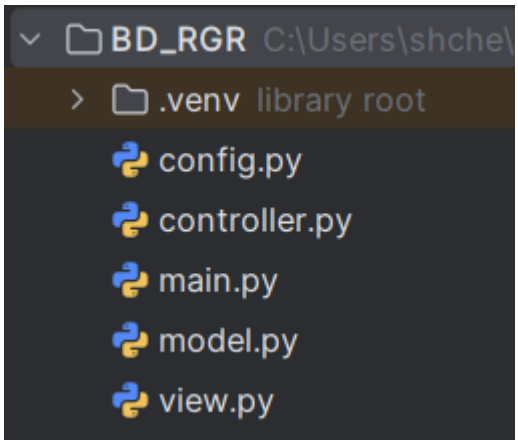

Рисунок 3 – Структура програми

Програма складається з п'яти основних файлів, що відповідають патерну MVC та конфігурації:

- config.py: Зберігає параметри підключення до бази даних (host, user, dbname, password).
- main.py: Точка входу в програму. Викликає контролер та передає йому управління.
- model.py: Описує клас Model, який відповідає за управління підключенням до бази даних і виконанням SQL-запитів.
- controller.py: Реалізовано інтерфейс взаємодії з користувачем, включаючи обробку вибору меню, а також інші дії, необхідні для взаємодії з моделлю та представленням.
- view.py: Описує функції, які відображають результати виконання різних дій користувача на екрані консолі.

Отже, структура програми відповідає патерну MVC.

Структура меню програми

Програма використовує ієрархічне меню для полегшення навігації. Користувач спочатку обирає категорію дії, а потім переходить у відповідне підменю.

```
--- ГОЛОВНЕ МЕНЮ ---
1.  Показати дані (Show data)
2.  Додати дані (Add data)
3.  Редагувати дані (Update data)
4.  Видалити дані (Delete data)
5.  Згенерувати дані (Generate data)
6.  Вихід (Quit)
Ваш вибір:
```

Рисунок 4 – Структура меню користувача

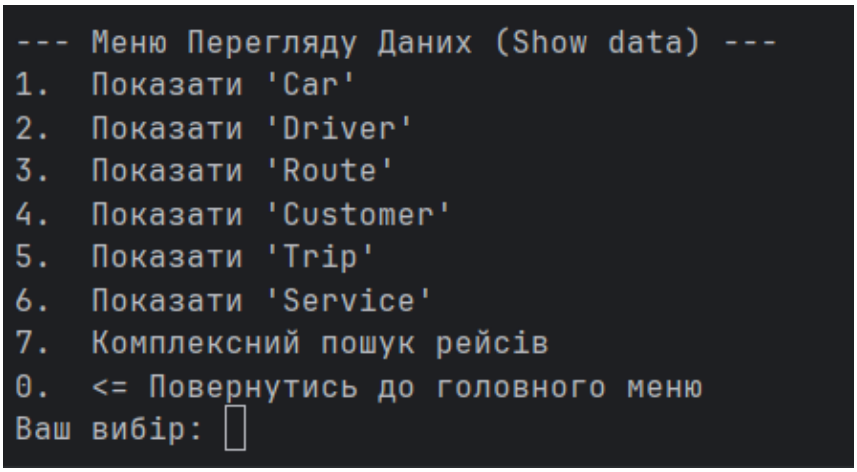

Рисунок 5 – Підменю перегляду

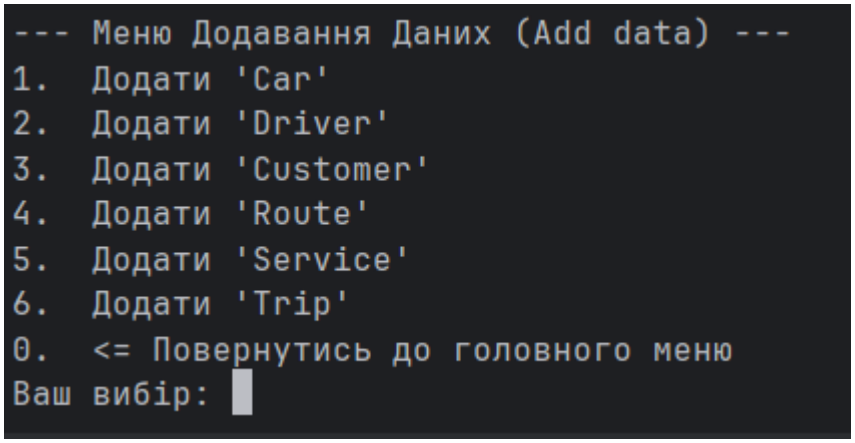

Рисунок 6 – Підменю додавання

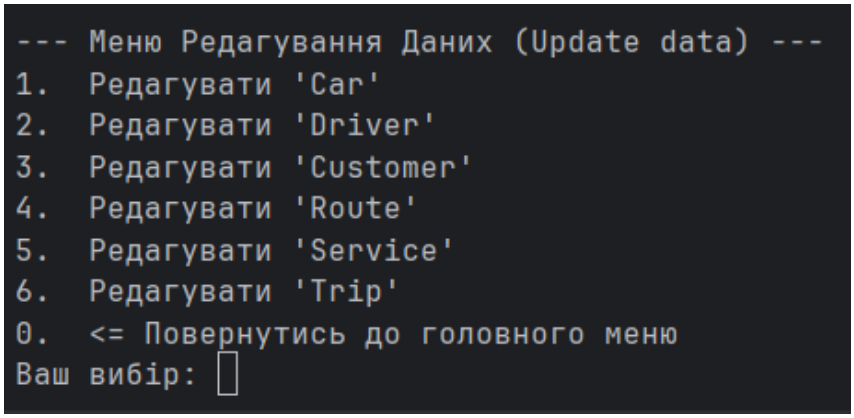

Рисунок 7 – Підменю оновлення

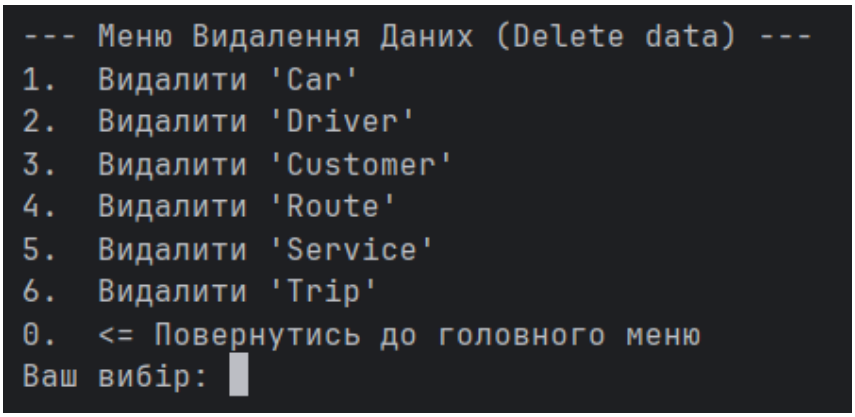

Рисунок 8 – Підменю видалення

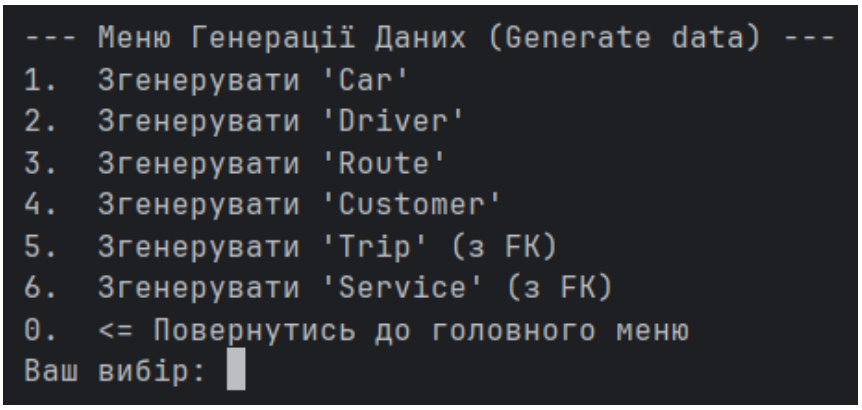

Рисунок 9 – Підменю генерації

Опис функціональності пунктів меню користувача

1. Показати дані (Show data): Відповідає за вивід даних з таблиць користувачу для перегляду. Має підменю для вибору конкретної таблиці та комплексного пошуку.
2. Додати дані (Add data): Відповідає за додавання нових даних у таблиці бази даних. Має підменю для вибору таблиці.
3. Редагувати дані (Update data): Відповідає за оновлення (зміну) даних у таблицях. Має підменю для вибору таблиці.
4. Видалити дані (Delete data): Відповідає за видалення даних з таблиць за їх унікальним ідентифікатором. Має підменю для вибору таблиці.
5. Згенерувати дані (Generate data): Відповідає за генерацію випадкових даних у відповідні таблиці бази даних. Має підменю для вибору таблиці.
6. Вихід (Quit): Відповідає за закриття з'єднання з БД та вихід з програми.

Тестування роботи функцій програми в різних ситуаціях

1. **Тестування створення**

Тест 1.1: Спроба додавання запису з неіснуючим зовнішнім ключем.

При спробі додати новий Рейс (таблиця trip) з car_id = 9999 (який не існує в батьківській таблиці car), програма коректно перехоплює помилку бази даних (psycopg.errors.ForeignKeyViolation). Користувачу

виводиться повідомлення про помилку цілісності, а не системний збій.

```
--- Додавання 'Trip' ---
Дата виїзду (YYYY-MM-DD): 2025.10.11
Дата прибуття (YYYY-MM-DD): 2025.11.11
Дата повернення (YYYY-MM-DD): 2025.11.12
Опис вантажу: Ліки
Вага (кг): 2000
ID Автомобіля (Car ID): 9999
ID Водія (Driver ID): 1
ID Маршруту (Route ID): 2
ID Клієнта (Customer ID): 1

*** Помилка цілісності (ForeignKeyViolation): insert or update on table "trip" violates foreign key constraint "trip_car_id_fkey"
DETAIL:  Key (car_id)=(9999) is not present in table "car". ***
```

Тест 1.2: Спроба додавання запису з некоректним типом даних.

При спробі додати Автомобіль і введенні тексту ("abc") у поле Вантажопідйомність (яке очікує integer), програма перехоплює помилку ValueError на стороні Python і виводить користувачу повідомлення про помилку валідації.

```
--- Додавання 'Trip' ---
Дата виїзду (YYYY-MM-DD): 2025.10.11
Дата прибуття (YYYY-MM-DD): 2025.11.11
Дата повернення (YYYY-MM-DD): 2025.11.12
Опис вантажу: Ліки
Вага (кг): abc
ID Автомобіля (Car ID): 1
ID Водія (Driver ID): 1
ID Маршруту (Route ID): 1
ID Клієнта (Customer ID): 1

*** Помилка: ID та вага мають бути числами. ***
```

Тест 1.3: Успішне додавання запису.

Була перевірена функція додавання нового запису в табличку trip.

Табличка до додавання:

	trip_id [PK] integer	departure_date date	arrival_date date	return_date date	cargo_description character varying (200)	cargo_weight integer	car_id integer	driver_id integer	route_id integer	customer_id integer
1	1	2025-09-01	2025-09-02	2025-09-03	Будівельні матеріали	15000	1	1	1	1
2	2	2025-09-05	2025-09-06	2025-09-07	Металопрокат	17000	2	2	2	2
3	3	2025-09-10	2025-09-10	2025-09-11	Сільськогосподарська техні…	20000	3	3	3	3

Додамо новий запис:

```
--- Додавання 'Trip' ---
Дата виїзду (YYYY-MM-DD): 2025-10-15
Дата прибуття (YYYY-MM-DD): 2025-10-20
Дата повернення (YYYY-MM-DD): 2025-10-21
Опис вантажу: Хімікати
Вага (кг): 20000
ID Автомобіля (Car ID): 3
ID Водія (Driver ID): 2
ID Маршруту (Route ID): 3
ID Клієнта (Customer ID): 1

*** Успішно додано 1 рейс. ***
```

Табличка після додавання:

	trip_id [PK] integer	departure_date date	arrival_date date	return_date date	cargo_description character varying (200)	cargo_weight integer	car_id integer	driver_id integer	route_id integer	customer_id integer
1	1	2025-09-01	2025-09-02	2025-09-03	Будівельні матеріали	15000	1	1	1	1
2	2	2025-09-05	2025-09-06	2025-09-07	Металопрокат	17000	2	2	2	2
3	3	2025-09-10	2025-09-10	2025-09-11	Сільськогосподарська техні...	20000	3	3	3	3
4	5	2025-10-15	2025-10-20	2025-10-21	Хімікати	20000	3	2	3	1

2. Тестування вилучення

Тест 2.1: Спроба вилучення неіснуючого запису.

При спробі видалити Автомобіль з car_id = 9999 (якого немає), програма коректно обробляє ситуацію і повідомляє, що запис не знайдено.

```
Введіть ID запису з 'car' для видалення: 999

*** 0 rows affected. (Запис з car_id = 999 не знайдено). ***
```

Тест 2.2: Спроба вилучення батьківського запису (Помилка Foreign Key).

Був створений Рейс (в trip), що посилається на Автомобіль з car_id = 1. При спробі видалити Автомобіль з car_id = 1 (який тепер є батьківським записом), програма перехоплює помилку ForeignKeyViolation. Запис не видаляється, цілісність даних збережена.

```
Введіть ID запису з 'car' для видалення: 1

*** ПОМИЛКА: update або delete в таблиці "car" порушує обмеження зовнішнього ключа. ***
```

Тест 2.3: Успішне вилучення.

При видаленні запису Автомобіль, на який немає посилань в інших таблицях, операція проходить успішно.

Табличка до видалення:

	car_id [PK] integer	vin character varying (17)	license_plate character varying (20)	brand character varying (50)	load_capacity integer
1	1	1HGBH41JXMN109186	AA1234BC	MAN	20000
2	2	2HGCM82633A004352	BC5678CC	Volvo	18000
3	3	INTAB7AP9GY256789	KA4331DC	Scania	22000
4	6	3JMT3G1C8JN5LEN0Q	TW2524GM	Mercedes	24100

Видалимо запис

```
Введіть ID запису з 'car' для видалення: 6

*** Deleted successfully! 1 рядків видалено. ***
```

Табличка після видалення

	car_id [PK] integer	vin character varying (17)	license_plate character varying (20)	brand character varying (50)	load_capacity integer
1	1	1HGBH41JXMN109186	AA1234BC	MAN	20000
2	2	2HGCM82633A004352	BC5678CC	Volvo	18000
3	3	INTAB7AP9GY256789	KA4331DC	Scania	22000

3. Тестування редагування

Тест 3.1: Успішне оновлення запису.

Була обрана таблиця Car, введено існуючий car_id. Для полів, які потрібно оновити (Марка, Вантажопідйомність), були введені нові значення. Для полів, які не потрібно змінювати, користувач просто натиснув Enter. Програма виконала UPDATE і повідомила про успішне оновлення.

Табличка до оновлення

	car_id [PK] integer	vin character varying (17)	license_plate character varying (20)	brand character varying (50)	load_capacity integer
1	1	1HGBH41JXMN109186	AA1234BC	MAN	20000
2	2	2HGCM82633A004352	BC5678CC	Volvo	18000
3	3	INTAB7AP9GY256789	KA4331DC	Scania	22000

Оновлення запису

```
--- Редагування 'Car' ---
ID автомобіля для оновлення: 1
Нова Марка (enter, щоб пропустити): Mercedes
Нова Вантажопідйомність (enter, щоб пропустити): 22000

*** Запис (ID: 1) успішно оновлено. ***
```

Табличка після редагування

	car_id [PK] integer	vin character varying (17)	license_plate character varying (20)	brand character varying (50)	load_capacity integer
1	2	2HGCM82633A004352	BC5678CC	Volvo	18000
2	3	INTAB7AP9GY256789	KA4331DC	Scania	22000
3	1	1HGBH41JXMN109186	AA1234BC	Mercedes	22000

Тест 3.2: Спроба оновлення неіснуючого запису.

При спробі оновити Автомобіль з car_id = 9999 (якого немає), програма коректно обробляє ситуацію і повідомляє, що запис не знайдено.

```
--- Редагування 'Car' ---
ID автомобіля для оновлення: 999
Нова Марка (enter, щоб пропустити): Scania
Нова Вантажопідйомність (enter, щоб пропустити): 10000

*** Запис з ID 999 не знайдено. ***
```

Тест 3.3: Спроба оновлення з некоректним типом даних.

При спробі оновити Автомобіль з існуючим car_id і введенні тексту ("abc") у поле Вантажопідйомність, програма перехоплює помилку ValueError і повідомляє про невірний тип даних.

```
--- Редагування 'Car' ---
ID автомобіля для оновлення: 1
Нова Марка (enter, щоб пропустити):
Нова Вантажопідйомність (enter, щоб пропустити): abc

*** Помилка: Вантажопідйомність має бути числом. ***
```

4. Тестування генерації даних

Протестуємо генерацію даних на прикладі таблички Car.

Табличка до генерації

	car_id [PK] integer	vin character varying (17)	license_plate character varying (20)	brand character varying (50)	load_capacity integer
1	2	2HGCM82633A004352	BC5678CC	Volvo	18000
2	3	INTAB7AP9GY256789	KA4331DC	Scania	22000
3	1	1HGBH41JXMN109186	AA1234BC	Mercedes	22000

Згенеруємо 100 записів

```
Ваш вибір: 1
Введіть кількість записів для генерації: 100
Генерація 100 записів в Python...
Вставка 100 записів у БД...

*** Успішно згенеровано та додано 100 унікальних записів. ***
```

Табличка після генерації

	car_id [PK] integer	vin character varying (17)	license_plate character varying (20)	brand character varying (50)	load_capacity integer
1	2	2HGCM82633A004352	BC5678CC	Volvo	18000
2	3	INTAB7AP9GY256789	KA4331DC	Scania	22000
3	1	1HGBH41JXMN109186	AA1234BC	Mercedes	22000
4	11	3DEF0AEA535E44CD9	NJ7702BX	MAN	11094
5	12	DE62150FB571456DA	KD4380MJ	Volvo	11337
6	13	AD7F256D6A8342E49	UH0220BS	Mercedes	14555
7	14	BBDAF8A5225847169	QH9099JR	DAF	23348
8	15	CBAE05E9EA434FC68	LI5128EV	Mercedes	22917
9	16	BB4D724EB87448CDA	RG4324AU	Mercedes	19388

Total rows: 103 | Query complete 00:00:00.196

Тест 4.2: Генерація даних з перевіркою зовнішніх ключів (FK).

Була протестована генерація для таблиць, що містять зовнішні ключі.

1. Користувач обирає таблицю для генерації (наприклад, Trip).
2. Користувач вводить кількість записів, наприклад 100000.
3. Програма виконує SQL-запит, який перед генерацією випадкових даних для trip (дати, вага, опис) вибирає випадкові, але гарантовано існуючі car_id, driver_id, route_id та customer_id з відповідних батьківських таблиць за допомогою (SELECT ... FROM ... ORDER BY random() LIMIT 1).
4. Це гарантує, що при масовій вставці не виникне помилок порушення цілісності зовнішніх ключів (ForeignKeyViolation).
5. Програма повідомляє про успішну генерацію.

Табличка до генерації

	trip_id [PK] integer	departure_date date	arrival_date date	return_date date	cargo_description character varying (200)	cargo_weight integer	car_id integer	driver_id integer	route_id integer	customer_id integer
1	1	2025-09-01	2025-09-02	2025-09-03	Будівельні матеріали	15000	1	1	1	1
2	2	2025-09-05	2025-09-06	2025-09-07	Металопрокат	17000	2	2	2	2
3	3	2025-09-10	2025-09-10	2025-09-11	Сільськогосподарська техні...	20000	3	3	3	3
4	5	2025-10-15	2025-10-20	2025-10-21	Хімікати	20000	3	2	3	1

Згенеруємо 100000 записів

```
Введіть кількість записів для генерації: 100000
Отримання списків існуючих ID...
Генерація 100000 записів 'trip' в Python...
Вставка 100000 записів у БД...

*** Успішно згенеровано та додано 100000 записів 'trip'. ***
```

Табличка після генерації

	trip_id [PK] integer	departure_date date	arrival_date date	return_date date	cargo_description character varying (200)	cargo_weight integer	car_id integer	driver_id integer	route_id integer	customer_id integer
1	1	2025-09-01	2025-09-02	2025-09-03	Будівельні матеріали	15000	1	1	1	1
2	2	2025-09-05	2025-09-06	2025-09-07	Металопрокат	17000	2	2	2	2
3	3	2025-09-10	2025-09-10	2025-09-11	Сільськогосподарська техні...	20000	3	3	3	3
4	200006	2025-10-06	2025-10-08	2025-10-11	Металопрокат	22400	94	86	29	16
5	5	2025-10-15	2025-10-20	2025-10-21	Хімікати	20000	3	2	3	1
6	200007	2025-10-20	2025-10-25	2025-10-27	Техніка	13900	29	29	17	88
7	200008	2025-10-13	2025-10-22	2025-10-22	Техніка	17100	20	65	32	8
8	200009	2025-10-07	2025-10-08	2025-10-10	Будматеріали	11300	34	5	37	27
9	200010	2025-09-23	2025-09-24	2025-09-29	Продукти	12600	25	66	14	22

Total rows: 100004 | Query complete 00:00:00.231

5. Тестування комплексного пошуку

Був протестований комплексний пошук з меню "Показати дані".

1. Користувач обирає опцію "Комплексний пошук рейсів".
2. Користувач вводить параметри пошуку:
   - МІН вага вантажу: 15000
   - МАКС вага вантажу: 20000
   - Шаблон марки авто: Volv%
3. Програма виконує SQL-запит, що об'єднує (JOIN) таблиці trip, car та driver.
4. Результати пошуку виводяться у вигляді таблиці.
5. Згідно з вимогами, виводиться час виконання запиту у мілісекундах.

```
--- Комплексний пошук рейсів ---
Введіть МІН вагу вантажу (напр., 5000): 19900
Введіть МАКС вагу вантажу (напр., 20000): 19999
Введіть шаблон марки авто (напр., 'Volv%' або '%'): Scania

Trip ID | Вантаж       | Вага  | Марка  | Номер    | Водій
--------------------------------------------------------------------------
202433  | Продукти     | 19900 | Scania | YV3766IJ | Михайло Коваленко
205180  | Техніка      | 19900 | Scania | VD7818OL | Іван Петренко
205539  | Будматеріали | 19900 | Scania | VD7818OL | Сергій Петренко
206809  | Техніка      | 19900 | Scania | FQ9877OC | Іван Шевченко
207156  | Металопрокат | 19900 | Scania | SD1454GH | Сергій Сидоренко
209048  | Будматеріали | 19900 | Scania | WZ3803HQ | Іван Сидоренко
209371  | Будматеріали | 19900 | Scania | BZ9180SH | Олександр Сидоренко
209698  | Будматеріали | 19900 | Scania | KE0139ZN | Іван Петренко
210402  | Продукти     | 19900 | Scania | AV4777CU | Олександр Ковальчук
212869  | Техніка      | 19900 | Scania | AV4777CU | Михайло Сидоренко
213311  | Техніка      | 19900 | Scania | WZ3803HQ | Михайло Сидоренко
214407  | Техніка      | 19900 | Scania | GW3470CK | Сергій Шевченко
216597  | Техніка      | 19900 | Scania | NK1361WH | Сергій Петренко
217809  | Продукти     | 19900 | Scania | RM8741CB | Петро Ковальчук
218021  | Техніка      | 19900 | Scania | VD7818OL | Михайло Шевченко
218212  | Будматеріали | 19900 | Scania | YA0446PD | Петро Петренко
218593  | Техніка      | 19900 | Scania | XG7264YA | Олександр Петренко
218942  | Будматеріали | 19900 | Scania | FQ9877OC | Сергій Іваненко
219228  | Продукти     | 19900 | Scania | SD1454GH | Сергій Ковальчук
219679  | Продукти     | 19900 | Scania | SF1499WW | Петро Шевченко
220268  | Продукти     | 19900 | Scania | KE0139ZN | Михайло Іваненко
223092  | Продукти     | 19900 | Scania | VD7818OL | Іван Сидоренко
223355  | Металопрокат | 19900 | Scania | XG7264YA | Петро Сидоренко
223481  | Техніка      | 19900 | Scania | XG7264YA | Олександр Петренко
224315  | Техніка      | 19900 | Scania | XG7264YA | Олександр Петренко
224732  | Будматеріали | 19900 | Scania | BZ9180SH | Сергій Сидоренко
227209  | Продукти     | 19900 | Scania | FQ9877OC | Сергій Петренко
227527  | Будматеріали | 19900 | Scania | KE0139ZN | Сергій Петренко
```

```
255079  | Будматеріали | 19900 | Scania | YA0446PD | Іван Петренко
257575  | Техніка      | 19900 | Scania | RM8741CB | Олександр Іваненко
257958  | Продукти     | 19900 | Scania | SD1454GH | Олександр Шевченко
262627  | Будматеріали | 19900 | Scania | RM8741CB | Іван Шевченко
265760  | Металопрокат | 19900 | Scania | SF1499WW | Іван Іваненко
265821  | Металопрокат | 19900 | Scania | SD1454GH | Михайло Ковальчук
265909  | Техніка      | 19900 | Scania | KA4331DC | Сергій Шевченко
266771  | Техніка      | 19900 | Scania | VD7818OL | Олександр Шевченко
266914  | Будматеріали | 19900 | Scania | YV3766IJ | Петро Іваненко
270033  | Техніка      | 19900 | Scania | CI6120NJ | Олександр Шевченко
270170  | Металопрокат | 19900 | Scania | NK1361WH | Іван Ковальчук
271475  | Продукти     | 19900 | Scania | SF1499WW | Іван Шевченко
274042  | Продукти     | 19900 | Scania | YA0446PD | Петро Сидоренко
274055  | Техніка      | 19900 | Scania | KE0139ZN | Михайло Іваненко
274436  | Техніка      | 19900 | Scania | FQ9877OC | Михайло Ковальчук
275119  | Техніка      | 19900 | Scania | CI6120NJ | Михайло Шевченко
275588  | Продукти     | 19900 | Scania | GW3470CK | Олександр Петренко
275911  | Металопрокат | 19900 | Scania | WZ3803HQ | Олександр Ковальчук
276265  | Металопрокат | 19900 | Scania | VD7818OL | Сергій Шевченко
276332  | Металопрокат | 19900 | Scania | YA0446PD | Петро Сидоренко
277459  | Будматеріали | 19900 | Scania | RM8741CB | Олександр Шевченко
278606  | Продукти     | 19900 | Scania | NK1361WH | Петро Ковальчук
279044  | Будматеріали | 19900 | Scania | WZ3803HQ | Олександр Іваненко
279835  | Будматеріали | 19900 | Scania | AK5271EN | Сергій Сидоренко
281131  | Техніка      | 19900 | Scania | CI6120NJ | Олександр Шевченко
281563  | Техніка      | 19900 | Scania | RM8741CB | Іван Шевченко
283249  | Металопрокат | 19900 | Scania | SF1499WW | Сергій Петренко
287850  | Металопрокат | 19900 | Scania | BZ9180SH | Іван Іваненко
289337  | Будматеріали | 19900 | Scania | WZ3803HQ | Іван Ковальчук
293699  | Металопрокат | 19900 | Scania | AK5271EN | Іван Ковальчук
295553  | Будматеріали | 19900 | Scania | GW3470CK | Сергій Ковальчук
296932  | Будматеріали | 19900 | Scania | YA0446PD | Петро Сидоренко
296963  | Будматеріали | 19900 | Scania | YA0446PD | Михайло Іваненко
297961  | Будматеріали | 19900 | Scania | NK1361WH | Михайло Коваленко
298847  | Техніка      | 19900 | Scania | SD1454GH | Олександр Шевченко

*** Час виконання запиту: 6.21 мс. ***
```

Фрагменти коду

Нижче наведені приклади коду з модуля model.py, що ілюструють реалізацію основних функціональних можливостей додатку.

1. **Перегляд даних (Show data)**

   Функція get_all_data використовується для отримання перших 100 записів з будь-якої таблиці. Вона будує SQL-запит динамічно, але з базовим захистом від ін'єкцій для назви таблиці, і використовує _execute_query для безпечного виконання.

```python
def get_all_data(self, table_name):
    if not table_name.replace('_', '').isalnum():
        return None, "Помилка: Неприпустима назва таблиці."

    query = f"SELECT * FROM {table_name} LIMIT 100"
    return self._execute_query(query, fetch=True)
```

   Пояснення: Функція приймає назву таблиці як рядок, перевіряє її на допустимі символи, формує простий запит SELECT * ... LIMIT 100 і викликає _execute_query для його виконання та отримання результатів (fetch=True).

2. **Додавання даних (Add data)**

   Функція add_car демонструє додавання нового запису. Вона приймає дані від користувача, виконує перевірку типу для числових полів (load_capacity), формує параметризований запит INSERT і використовує _execute_query для його виконання.

```
def add_car(self, vin, license_plate, brand, load_capacity):
    query = "INSERT INTO car (vin, license_plate, brand,
load_capacity) VALUES (%s, %s, %s, %s)"
    try:
        load_capacity_int = int(load_capacity)
    except ValueError:
        return "Помилка: 'Вантажопідйомність' має бути числом.", False
    params = (vin, license_plate, brand, load_capacity_int)
    rowcount, message = self._execute_query(query, params)
    if rowcount is not None:
        return f"Успішно додано {rowcount} автомобіль.", True
    return message, False
```

Пояснення: Функція отримує дані для нового автомобіля, перевіряє коректність типу вантажопідйомності, готує кортеж параметрів params і передає SQL-запит та параметри до _execute_query. Це забезпечує захист від SQL-ін'єкцій.

3. Оновлення даних (Update data)

Функція update_car (і схожі на неї update_driver, update_customer тощо) реалізує оновлення записів. Вона динамічно будує частину SET SQL-запиту, включаючи лише ті поля, для яких користувач ввів нові значення. Це дозволяє оновлювати лише окремі атрибути запису. Використовується параметризований запит.

```
def update_car(self, car_id, brand, load_capacity):
    fields = ["brand", "load_capacity"]
    values = [brand, load_capacity]
    try:
        if load_capacity: values[1] = int(load_capacity)
    except ValueError:
        return "Помилка: Вантажопідйомність має бути числом."
    return self._update_record("car", "car_id", car_id, fields,
values)
```

Універсальна функція оновлення (_update_record)

Щоб уникнути дублювання коду в функціях оновлення (update_car, update_driver тощо), була створена приватна допоміжна функція _update_record. Вона приймає назву таблиці, назву поля ID, сам ID, список полів для можливого оновлення та список відповідних нових значень. Вона динамічно будує та виконує SQL-запит UPDATE, включаючи лише ті поля, для яких користувач надав нове значення.

```
def _update_record(self, table_name, id_field, record_id,
fields_to_update, values):
    if not record_id.isdigit():
        return "Помилка: ID має бути числом."

    updates = []
    params = []

    for i, value in enumerate(values):
        if value:
            field_name = fields_to_update[i]
```

```
            updates.append(f"{field_name} = %s")
            params.append(value)

    if not updates:
        return "Немає даних для оновлення."

    params.append(int(record_id))
    query = f"UPDATE {table_name} SET {', '.join(updates)} WHERE
{id_field} = %s"

    rowcount, message = self._execute_query(query, params)

    if rowcount == 0:
        return f"Запис з ID {record_id} не знайдено."
    elif rowcount is not None:
        return f"Запис (ID: {record_id}) успішно оновлено."
    else:
        return message
```

Пояснення: Ця функція значно спрощує код. Вона ітерує по наданих значеннях values. Якщо значення не порожнє, відповідне поле з fields_to_update додається до списку updates (частина SET ...), а саме значення — до списку params. Потім формується динамічний, але безпечний (завдяки параметризації %s) SQL-запит UPDATE, який оновлює лише вказані поля для запису з заданим record_id. Виклик _execute_query забезпечує обробку помилок та управління транзакціями.

4. **Видалення даних (Delete data)**

Функція delete_data_dynamic реалізує видалення записів з будь-якої таблиці за вказаним полем та значенням. Вона використовує _execute_query для безпечного виконання та обробки помилок, включаючи помилку видалення батьківського запису, на який є посилання.

```
def delete_data_dynamic(self, table_name, field, value):
    if not table_name.replace('_', '').isalnum() or not
field.replace('_', '').isalnum():
        return "Помилка: Неприпустима назва таблиці або поля."

    query = f"DELETE FROM {table_name} WHERE {field} = %s"

    try:
        param = int(value)
    except ValueError:
        param = str(value)

    rowcount, message = self._execute_query(query, (param,))

    if rowcount is None:
        if "ForeignKeyViolation" in message:
            return f"ПОМИЛКА: update або delete в таблиці
\"{table_name}\" порушує обмеження зовнішнього ключа."
        return message
    elif rowcount == 0:
        return f"0 rows affected. (Запис з {field} = {value} не
```

```
знайдено)."
    else:
        return f"Deleted successfully! {rowcount} рядків видалено."
```

Пояснення: Функція приймає назву таблиці, поле та значення як рядки. Вона виконує базову перевірку назв на допустимі символи та використовує параметризований запит (%s) для безпечної передачі значення value у WHERE. Обробляє помилки ForeignKeyViolation окремо для надання зрозумілого повідомлення згідно з вимогами звіту.

5. **Генерація даних (Generate data)**

Генерація даних реалізована для всіх таблиць. Для "батьківських" таблиць використовуються SQL-запити з generate_series та random(). Для таблиць з зовнішніми ключами (trip, service) використовується підхід генерації даних у Python з вибіркою існуючих ID з батьківських таблиць для гарантування цілісності.

```
def generate_trips(self, count):

        if not self.conn:
            return "Помилка: Немає з'єднання з БД."

        print("Отримання списків існуючих ID...")
        car_ids = self._get_existing_ids("car", "car_id")
        driver_ids = self._get_existing_ids("driver", "driver_id")
        route_ids = self._get_existing_ids("route", "route_id")
        customer_ids = self._get_existing_ids("customer",
"customer_id")

        if not all([car_ids, driver_ids, route_ids, customer_ids]):
            return ("Помилка: Неможливо згенерувати 'trip'. "
                    "Одна або декілька батьківських таблиць порожні.")

        generated_data = []
        cargo_options = ['Будматеріали', 'Металопрокат', 'Продукти',
'Техніка', 'Хімікати']
        print(f"Генерація {count} записів 'trip' в Python...")
        for _ in range(count):
            departure_date = datetime.date.today() -
datetime.timedelta(days=random.randint(0, 30))
            arrival_date = departure_date +
datetime.timedelta(days=random.randint(1, 10))
            return_date = arrival_date +
datetime.timedelta(days=random.randint(0, 5))
            cargo_desc = random.choice(cargo_options)
            cargo_weight = random.randint(50, 249) * 100

            car_id = random.choice(car_ids)
            driver_id = random.choice(driver_ids)
            route_id = random.choice(route_ids)
            customer_id = random.choice(customer_ids)

            generated_data.append((
                departure_date, arrival_date, return_date, cargo_desc,
cargo_weight,
                car_id, driver_id, route_id, customer_id
            ))
```

```
        query = """
                INSERT INTO trip (departure_date, arrival_date,
return_date, cargo_description, cargo_weight,
                                  car_id, driver_id, route_id,
customer_id)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
                """

        print(f"Вставка {len(generated_data)} записів у БД...")
        try:
            with self.conn.cursor() as cursor:
                cursor.executemany(query, generated_data)
                inserted_count = cursor.rowcount
                self.conn.commit()
                return f"Успішно згенеровано та додано
{inserted_count} записів 'trip'."

        except Exception as e:
            self.conn.rollback()
            return f"Помилка при пакетній вставці 'trip': {e}"
```

Пояснення: Функція спочатку отримує списки всіх дійсних ID з батьківських таблиць. Потім у циклі Python генерує дані для кожного нового рейсу, вибираючи зовнішні ключі випадковим чином з цих списків. Нарешті, всі згенеровані дані ефективно вставляються у базу даних за допомогою cursor.executemany. Це забезпечує цілісність даних.

6. Комплексний пошук

Реалізований у функції search_trips_complex, яка виконує запит з JOIN трьох таблиць (trip, car, driver) та фільтрацією за діапазоном (BETWEEN) і шаблоном (ILIKE). Також вимірюється час виконання запиту.

```
def search_trips_complex(self, min_weight, max_weight, brand_pattern):

    query = """
            SELECT t.trip_id, \
                   t.cargo_description, \
                   t.cargo_weight, \
                   c.brand, \
                   c.license_plate, \
                   d.name || ' ' || d.surname AS driver_full_name
            FROM trip AS t \
                     JOIN \
                 car AS c ON t.car_id = c.car_id \
                     JOIN \
                 driver AS d ON t.driver_id = d.driver_id
            WHERE t.cargo_weight BETWEEN %s AND %s
              AND c.brand ILIKE %s; \
            """

    if not self.conn:
        return None, 0, "Помилка: Немає з'єднання з БД."

    try:
```

```
        with self.conn.cursor() as cursor:
            start_time = time.time()
            cursor.execute(query, (min_weight, max_weight,
brand_pattern))
            end_time = time.time()
            duration_ms = (end_time - start_time) * 1000

            result = cursor.fetchall()
            self.conn.commit()
            return result, duration_ms, "Пошук успішний."

    except Exception as e:
        self.conn.rollback()
        return None, 0, f"Помилка пошуку: {e}"
```

Пояснення: Функція приймає параметри фільтрації від користувача. Вона використовує JOIN для об'єднання даних з пов'язаних таблиць trip, car та driver. Фільтрація здійснюється за допомогою WHERE з операторами BETWEEN для ваги та ILIKE (нечутливий до регістру LIKE) для марки автомобіля. Параметри передаються безпечно через cursor.execute. Вимірювання часу відбувається за допомогою модуля time.

Повний код програми

main.py

```
import controller

if __name__ == "__main__":
    controller.run()
```

controller.py

```
from model import Model
import view


def run():
    model = Model()

    if model.conn is None:
        view.show_message("Не вдалося підключитися до БД. Робота
програми неможлива.")
        return

    while True:
        view.show_main_menu()
        choice = view.get_user_choice()

        if choice == '1':
            run_show_data_menu(model)
        elif choice == '2':
            run_add_data_menu(model)
        elif choice == '3':
            run_update_data_menu(model)
        elif choice == '4':
            run_delete_menu(model)
```

```
        elif choice == '5':
            run_generate_data_menu(model)
        elif choice == '6':
            model.close_connection()
            view.show_message("До побачення!")
            break
        else:
            view.show_message("Невірний вибір. Спробуйте ще раз.")


def run_show_data_menu(model):
    options = ["Показати 'Car'", "Показати 'Driver'", "Показати
'Route'",
               "Показати 'Customer'", "Показати 'Trip'", "Показати
'Service'",
               "Комплексний пошук рейсів"]

    while True:
        view.show_submenu("Меню Перегляду Даних (Show data)", options)
        choice = view.get_user_choice()

        if choice == '1':
            rows, msg = model.get_all_data("car")
            if rows: view.show_list(rows, ["car_id", "vin",
"license_plate", "brand", "load_capacity"])
            view.show_message(msg)
        elif choice == '2':
            rows, msg = model.get_all_data("driver")
            if rows: view.show_list(rows, ["driver_id",
"license_number", "surname", "name", "license_category"])
            view.show_message(msg)
        elif choice == '3':
            rows, msg = model.get_all_data("route")
            if rows: view.show_list(rows, ["route_id",
"departure_point", "destination_point", "distance_km"])
            view.show_message(msg)
        elif choice == '4':
            rows, msg = model.get_all_data("customer")
            if rows: view.show_list(rows, ["customer_id", "full_name",
"phone", "email", "address"])
            view.show_message(msg)
        elif choice == '5':
            rows, msg = model.get_all_data("trip")
            if rows: view.show_list(rows,["trip_id", "departure_date",
"arrival_date", "return_date", "cargo_description",
                                          "cargo_weight", "car_id",
"driver_id", "route_id", "customer_id"])
            view.show_message(msg)
        elif choice == '6':
            rows, msg = model.get_all_data("service")
            if rows: view.show_list(rows, ["service_id", "car_id",
"service_date", "description", "cost"])
            view.show_message(msg)
        elif choice == '7':
            handle_complex_search(model)
        elif choice == '0':
            break
        else:
```

```
            view.show_message("Невірний вибір.")


def handle_complex_search(model):
    try:
        min_w, max_w, pattern = view.get_search_params()
        min_w_int, max_w_int = int(min_w), int(max_w)
        rows, duration, message =
model.search_trips_complex(min_w_int, max_w_int, pattern)
        if rows is not None:
            view.show_search_results(rows, duration)
        view.show_message(message)
    except ValueError:
        view.show_message("Помилка: Вага має бути числом.")


def run_add_data_menu(model):
    options = ["Додати 'Car'", "Додати 'Driver'", "Додати 'Customer'",
               "Додати 'Route'", "Додати 'Service'",
               "Додати 'Trip'"]

    while True:
        view.show_submenu("Меню Додавання Даних (Add data)", options)
        choice = view.get_user_choice()
        try:
            if choice == '1':
                data = view.get_new_car_data()
                msg, success = model.add_car(*data)
                view.show_message(msg)
            elif choice == '2':
                data = view.get_new_driver_data()
                msg, success = model.add_driver(*data)
                view.show_message(msg)
            elif choice == '3':
                data = view.get_new_customer_data()
                msg, success = model.add_customer(*data)
                view.show_message(msg)
            elif choice == '4':
                data = view.get_new_route_data()
                msg, success = model.add_route(*data)
                view.show_message(msg)
            elif choice == '5':
                data = view.get_new_service_data()
                msg, success = model.add_service(*data)
                view.show_message(msg)
            elif choice == '6':
                data = view.get_new_trip_data()
                msg = model.add_trip(*data)
                view.show_message(msg)
            elif choice == '0':
                break
            else:
                view.show_message("Невірний вибір.")
        except Exception as e:
            view.show_message(f"Помилка вводу: {e}")


def run_update_data_menu(model):
```

```
    options = ["Редагувати 'Car'", "Редагувати 'Driver'", "Редагувати
'Customer'",
               "Редагувати 'Route'", "Редагувати 'Service'",
"Редагувати 'Trip'"]

    while True:
        view.show_submenu("Меню Редагування Даних (Update data)",
options)
        choice = view.get_user_choice()
        try:
            if choice == '1':
                data = view.get_update_car_data()
                msg = model.update_car(*data)
                view.show_message(msg)
            elif choice == '2':
                data = view.get_update_driver_data()
                msg = model.update_driver(*data)
                view.show_message(msg)
            elif choice == '3':
                data = view.get_update_customer_data()
                msg = model.update_customer(*data)
                view.show_message(msg)
            elif choice == '4':
                data = view.get_update_route_data()
                msg = model.update_route(*data)
                view.show_message(msg)
            elif choice == '5':
                data = view.get_update_service_data()
                msg = model.update_service(*data)
                view.show_message(msg)
            elif choice == '6':
                data = view.get_update_trip_data()
                msg = model.update_trip(*data)
                view.show_message(msg)
            elif choice == '0':
                break
            else:
                view.show_message("Невірний вибір.")
        except Exception as e:
            view.show_message(f"Помилка вводу: {e}")


def run_delete_menu(model):
    options = ["Видалити 'Car'", "Видалити 'Driver'", "Видалити
'Customer'",
               "Видалити 'Route'", "Видалити 'Service'",
               "Видалити 'Trip'"]

    while True:
        view.show_submenu("Меню Видалення Даних (Delete data)",
options)
        choice = view.get_user_choice()

        table_name = None
        id_field = None

        if choice == '1':
            table_name, id_field = 'car', 'car_id'
```

```
        elif choice == '2':
            table_name, id_field = 'driver', 'driver_id'
        elif choice == '3':
            table_name, id_field = 'customer', 'customer_id'
        elif choice == '4':
            table_name, id_field = 'route', 'route_id'
        elif choice == '5':
            table_name, id_field = 'service', 'service_id'
        elif choice == '6':
            table_name, id_field = 'trip', 'trip_id'
        elif choice == '0':
            break
        else:
            view.show_message("Невірний вибір.")
            continue

        try:
            value = view.get_id_input(f"ID запису з '{table_name}' для
видалення")
            message = model.delete_data_dynamic(table_name, id_field,
value)
            view.show_message(message)
        except Exception as e:
            view.show_message(f"Помилка: {e}")


def run_generate_data_menu(model):
    options = ["Згенерувати 'Car'", "Згенерувати 'Driver'",
"Згенерувати 'Route'",
               "Згенерувати 'Customer'", "Згенерувати 'Trip' (з FK)",
"Згенерувати 'Service' (з FK)"]

    while True:
        view.show_submenu("Меню Генерації Даних (Generate data)",
options)
        choice = view.get_user_choice()

        try:
            if choice == '0':
                break

            count_str = view.get_generation_count()
            count = int(count_str)
            if count <= 0: raise ValueError("Кількість > 0")

            if choice == '1':
                view.show_message(model.generate_cars(count))
            elif choice == '2':
                view.show_message(model.generate_drivers(count))
            elif choice == '3':
                view.show_message(model.generate_routes(count))
            elif choice == '4':
                view.show_message(model.generate_customers(count))
            elif choice == '5':
                view.show_message(model.generate_trips(count))
            elif choice == '6':
                view.show_message(model.generate_service(count))
            else:
```

```
                view.show_message("Невірний вибір.")

        except ValueError as e:
            view.show_message(f"Помилка: Кількість має бути додатнім
числом. {e}")
        except Exception as e:
            view.show_message(f"Неочікувана помилка: {e}")
```

view.py

```
def show_main_menu():
    print("\n--- ГОЛОВНЕ МЕНЮ ---")
    print("1.  Показати дані (Show data)")
    print("2.  Додати дані (Add data)")
    print("3.  Редагувати дані (Update data)")
    print("4.  Видалити дані (Delete data)")
    print("5.  Згенерувати дані (Generate data)")
    print("6.  Вихід (Quit)")


def show_submenu(title, options):
    print(f"\n--- {title} ---")
    for i, option in enumerate(options, 1):
        print(f"{i}.  {option}")
    print("0.  <= Повернутись до головного меню")


def get_user_choice():
    return input("Ваш вибір: ").strip()


def show_message(message):
    print(f"\n*** {message} ***")


def show_list(rows, headers):
    if not rows:
        print("-> Немає даних для відображення.")
        return

    col_widths = [len(str(h)) for h in headers]
    for row in rows:
        for i, col in enumerate(row):
            col_widths[i] = max(col_widths[i], len(str(col)))

    header_line = " | ".join(f"{h:<{col_widths[i]}}" for i, h in
enumerate(headers))
    print("\n" + header_line)
    print("-" * len(header_line))

    for row in rows:
        row_line = " | ".join(f"{str(col):<{col_widths[i]}}" for i,
col in enumerate(row))
        print(row_line)


def get_id_input(prompt="ID"):
```

```
    return input(f"Введіть {prompt}: ").strip()


def get_generation_count():
    return input("Введіть кількість записів для генерації: ").strip()


def get_search_params():
    print("\n--- Комплексний пошук рейсів ---")
    min_weight = input("Введіть МІН вагу вантажу (напр., 5000):
").strip()
    max_weight = input("Введіть МАКС вагу вантажу (напр., 20000):
").strip()
    brand_pattern = input("Введіть шаблон марки авто (напр., 'Volv%'
або '%'): ").strip()

    if not min_weight: min_weight = '0'
    if not max_weight: max_weight = '1000000'
    if not brand_pattern: brand_pattern = '%'

    return min_weight, max_weight, brand_pattern


def show_search_results(rows, duration_ms):
    headers = ["Trip ID", "Вантаж", "Вага", "Марка", "Номер", "Водій"]
    show_list(rows, headers)
    show_message(f"Час виконання запиту: {duration_ms:.2f} мс.")


def get_new_car_data():
    print("\n--- Додавання 'Car' ---")
    vin = input("VIN: ").strip()
    license_plate = input("Номерний знак: ").strip()
    brand = input("Марка: ").strip()
    load_capacity = input("Вантажопідйомність (кг): ").strip()
    return vin, license_plate, brand, load_capacity


def get_new_driver_data():
    print("\n--- Додавання 'Driver' ---")
    license_num = input("Номер посвідчення: ").strip()
    surname = input("Прізвище: ").strip()
    name = input("Ім'я: ").strip()
    category = input("Категорія: ").strip()
    return license_num, surname, name, category


def get_new_customer_data():
    print("\n--- Додавання 'Customer' ---")
    full_name = input("Повна назва компанії: ").strip()
    phone = input("Телефон: ").strip()
    email = input("Email: ").strip()
    address = input("Адреса: ").strip()
    return full_name, phone, email, address


def get_new_route_data():
    print("\n--- Додавання 'Route' ---")
```

```
    departure = input("Пункт відправлення: ").strip()
    destination = input("Пункт призначення: ").strip()
    distance = input("Відстань (км): ").strip()
    return departure, destination, distance


def get_new_service_data():
    print("\n--- Додавання 'Service' ---")
    car_id = input("ID Автомобіля (Car ID): ").strip()
    service_date = input("Дата обслуговування (YYYY-MM-DD): ").strip()
    description = input("Опис робіт: ").strip()
    cost = input("Вартість (напр., 3500.00): ").strip()
    return car_id, service_date, description, cost


def get_new_trip_data():
    print("\n--- Додавання 'Trip' ---")
    departure = input("Дата виїзду (YYYY-MM-DD): ").strip()
    arrival = input("Дата прибуття (YYYY-MM-DD): ").strip()
    return_d = input("Дата повернення (YYYY-MM-DD): ").strip()
    cargo_desc = input("Опис вантажу: ").strip()
    cargo_weight = input("Вага (кг): ").strip()
    car_id = input("ID Автомобіля (Car ID): ").strip()
    driver_id = input("ID Водія (Driver ID): ").strip()
    route_id = input("ID Маршруту (Route ID): ").strip()
    customer_id = input("ID Клієнта (Customer ID): ").strip()
    return departure, arrival, return_d, cargo_desc, cargo_weight,
car_id, driver_id, route_id, customer_id


def get_update_car_data():
    print("\n--- Редагування 'Car' ---")
    car_id = input("ID автомобіля для оновлення: ").strip()
    brand = input(f"Нова Марка (enter, щоб пропустити): ").strip()
    load_capacity = input(f"Нова Вантажопідйомність (enter, щоб
пропустити): ").strip()
    return car_id, brand, load_capacity


def get_update_driver_data():
    print("\n--- Редагування 'Driver' ---")
    driver_id = input("ID водія для оновлення: ").strip()
    surname = input(f"Нове Прізвище (enter, щоб пропустити):
").strip()
    name = input(f"Нове Ім'я (enter, щоб пропустити): ").strip()
    category = input(f"Нова Категорія (enter, щоб пропустити):
").strip()
    return driver_id, surname, name, category


def get_update_customer_data():
    print("\n--- Редагування 'Customer' ---")
    customer_id = input("ID клієнта для оновлення: ").strip()
    phone = input(f"Новий Телефон (enter, щоб пропустити): ").strip()
    email = input(f"Новий Email (enter, щоб пропустити): ").strip()
    return customer_id, phone, email
```

```
def get_update_route_data():
    print("\n--- Редагування 'Route' ---")
    route_id = input("ID маршруту для оновлення: ").strip()
    distance = input(f"Нова Відстань (км) (enter, щоб пропустити):
").strip()
    return route_id, distance


def get_update_service_data():
    print("\n--- Редагування 'Service' ---")
    service_id = input("ID обслуговування для оновлення: ").strip()
    description = input(f"Новий Опис (enter, щоб пропустити):
").strip()
    cost = input(f"Нова Вартість (enter, щоб пропустити): ").strip()
    return service_id, description, cost


def get_update_trip_data():
    print("\n--- Редагування 'Trip' ---")
    trip_id = input("ID рейсу для оновлення: ").strip()
    cargo_desc = input(f"Новий Опис вантажу (enter, щоб пропустити):
").strip()
    cargo_weight = input(f"Нова Вага (кг) (enter, щоб пропустити):
").strip()
    return trip_id, cargo_desc, cargo_weight
```

model.py

```
import datetime

import psycopg
from psycopg import errors
from config import DB_PARAMS
import time
import uuid
import random

class Model:
    def __init__(self):
        try:
            self.conn = psycopg.connect(**DB_PARAMS)
            self.conn.autocommit = False
        except Exception as e:
            self.conn = None
            print(f"Помилка підключення до БД: {e}")

    def close_connection(self):
        if self.conn:
            self.conn.close()

    def _execute_query(self, query, params=None, fetch=False):
        if not self.conn:
            return None, "Помилка: Немає з'єднання з БД."

        try:
            with self.conn.cursor() as cursor:
                cursor.execute(query, params)
```

```
                if fetch:
                    result = cursor.fetchall()
                    self.conn.commit()
                    return result, "Запит успішно виконано."
                else:
                    rowcount = cursor.rowcount
                    self.conn.commit()
                    return rowcount, "Запит успішно виконано."

        except errors.ForeignKeyViolation as e:
            self.conn.rollback()
            return None, f"Помилка цілісності (ForeignKeyViolation):
{e}"
        except Exception as e:
            self.conn.rollback()
            return None, f"Помилка при виконанні запиту: {e}"

    def get_all_data(self, table_name):
        if not table_name.replace('_', '').isalnum():
            return None, "Помилка: Неприпустима назва таблиці."

        query = f"SELECT * FROM {table_name} LIMIT 100"
        return self._execute_query(query, fetch=True)

    def add_car(self, vin, license_plate, brand, load_capacity):
        query = "INSERT INTO car (vin, license_plate, brand,
load_capacity) VALUES (%s, %s, %s, %s)"
        try:
            load_capacity_int = int(load_capacity)
        except ValueError:
            return "Помилка: 'Вантажопідйомність' має бути числом.",
False
        params = (vin, license_plate, brand, load_capacity_int)
        rowcount, message = self._execute_query(query, params)
        if rowcount is not None:
            return f"Успішно додано {rowcount} автомобіль.", True
        return message, False

    def add_driver(self, license_number, surname, name,
license_category):
        query = "INSERT INTO driver (license_number, surname, name,
license_category) VALUES (%s, %s, %s, %s)"
        params = (license_number, surname, name, license_category)
        rowcount, message = self._execute_query(query, params)
        if rowcount is not None:
            return f"Успішно додано {rowcount} водія.", True
        return message, False

    def add_customer(self, full_name, phone, email, address):
        query = "INSERT INTO customer (full_name, phone, email,
address) VALUES (%s, %s, %s, %s)"
        params = (full_name, phone, email, address)
        rowcount, message = self._execute_query(query, params)
        if rowcount is not None:
            return f"Успішно додано {rowcount} клієнта.", True
        return message, False

    def add_route(self, departure, destination, distance):
```

```
        query = "INSERT INTO route (departure_point,
destination_point, distance_km) VALUES (%s, %s, %s)"
        try:
            params = (departure, destination, int(distance))
        except ValueError:
            return "Помилка: 'Відстань' має бути числом.", False
        rowcount, message = self._execute_query(query, params)
        if rowcount is not None:
            return f"Успішно додано {rowcount} маршрут.", True
        return message, False

    def add_service(self, car_id, service_date, description, cost):
        query = "INSERT INTO service (car_id, service_date,
description, cost) VALUES (%s, %s, %s, %s)"
        try:
            params = (int(car_id), service_date, description,
float(cost))
        except ValueError:
            return "Помилка: ID має бути числом, а вартість - числом
(напр., 3500.00).", False
        rowcount, message = self._execute_query(query, params)
        if rowcount is not None:
            return f"Успішно додано {rowcount} запис про
обслуговування.", True
        return message, False

    def add_trip(self, departure, arrival, return_d, cargo_desc,
cargo_weight, car_id, driver_id, route_id,
                 customer_id):
        query = "INSERT INTO trip (departure_date, arrival_date,
return_date, cargo_description, cargo_weight, car_id, driver_id,
route_id, customer_id) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        try:
            params = (departure, arrival, return_d, cargo_desc,
int(cargo_weight), int(car_id), int(driver_id),
                      int(route_id), int(customer_id))
        except ValueError:
            return "Помилка: ID та вага мають бути числами."
        rowcount, message = self._execute_query(query, params)
        if rowcount is None:
            return message
        return f"Успішно додано {rowcount} рейс."

    def _update_record(self, table_name, id_field, record_id,
fields_to_update, values):
        if not record_id.isdigit():
            return "Помилка: ID має бути числом."

        updates = []
        params = []

        for i, value in enumerate(values):
            if value:
                field_name = fields_to_update[i]
                updates.append(f"{field_name} = %s")
                params.append(value)

        if not updates:
```

```
            return "Немає даних для оновлення."

        params.append(int(record_id))
        query = f"UPDATE {table_name} SET {', '.join(updates)} WHERE
{id_field} = %s"

        rowcount, message = self._execute_query(query, params)

        if rowcount == 0:
            return f"Запис з ID {record_id} не знайдено."
        elif rowcount is not None:
            return f"Запис (ID: {record_id}) успішно оновлено."
        else:
            return message

    def update_car(self, car_id, brand, load_capacity):
        fields = ["brand", "load_capacity"]
        values = [brand, load_capacity]
        try:
            if load_capacity: values[1] = int(load_capacity)
        except ValueError:
            return "Помилка: Вантажопідйомність має бути числом."
        return self._update_record("car", "car_id", car_id, fields,
values)

    def update_driver(self, driver_id, surname, name, category):
        fields = ["surname", "name", "license_category"]
        values = [surname, name, category]
        return self._update_record("driver", "driver_id", driver_id,
fields, values)

    def update_customer(self, customer_id, phone, email):
        fields = ["phone", "email"]
        values = [phone, email]
        return self._update_record("customer", "customer_id",
customer_id, fields, values)

    def update_route(self, route_id, distance):
        fields = ["distance_km"]
        values = [distance]
        try:
            if distance: values[0] = int(distance)
        except ValueError:
            return "Помилка: Відстань має бути числом."
        return self._update_record("route", "route_id", route_id,
fields, values)

    def update_service(self, service_id, description, cost):
        fields = ["description", "cost"]
        values = [description, cost]
        try:
            if cost: values[1] = float(cost)
        except ValueError:
            return "Помилка: Вартість має бути числом."
        return self._update_record("service", "service_id",
service_id, fields, values)

    def update_trip(self, trip_id, cargo_desc, cargo_weight):
```

```
        fields = ["cargo_description", "cargo_weight"]
        values = [cargo_desc, cargo_weight]
        try:
            if cargo_weight: values[1] = int(cargo_weight)
        except ValueError:
            return "Помилка: Вага має бути числом."
        return self._update_record("trip", "trip_id", trip_id, fields,
values)

    def delete_data_dynamic(self, table_name, field, value):
        if not table_name.replace('_', '').isalnum() or not
field.replace('_', '').isalnum():
            return "Помилка: Неприпустима назва таблиці або поля."

        query = f"DELETE FROM {table_name} WHERE {field} = %s"

        try:
            param = int(value)
        except ValueError:
            param = str(value)

        rowcount, message = self._execute_query(query, (param,))

        if rowcount is None:
            if "ForeignKeyViolation" in message:
                return f"ПОМИЛКА: update або delete в таблиці
\"{table_name}\" порушує обмеження зовнішнього ключа."
            return message
        elif rowcount == 0:
            return f"0 rows affected. (Запис з {field} = {value} не
знайдено)."
        else:
            return f"Deleted successfully! {rowcount} рядків
видалено."


    def _generate_data(self, query, count):
        rowcount, message = self._execute_query(query, (count,))
        if rowcount is None:
            return message
        return f"Успішно згенеровано {rowcount} записів."



    def generate_cars(self, count):

            if not self.conn:
                return "Помилка: Немає з'єднання з БД."

            generated_data = []
            brands = ['Volvo', 'MAN', 'Scania', 'Mercedes', 'DAF']

            print(f"Генерація {count} записів в Python...")
            for _ in range(count):
                raw_uuid = uuid.uuid4()
                vin = str(raw_uuid).replace('-', '').upper()[:17]

                plate = (
```

```
                        random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ') +
                        random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ') +
                        f"{random.randint(0, 9999):04d}" +
                        random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ') +
                        random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
                )

                brand = random.choice(brands)
                load_capacity = random.randint(10000, 25000)

                generated_data.append((vin, plate, brand,
load_capacity))

            query = """
                    INSERT INTO car (vin, license_plate, brand,
load_capacity)
                    VALUES (%s, %s, %s, %s)
                    """

            print(f"Вставка {len(generated_data)} записів у БД...")
            try:
                with self.conn.cursor() as cursor:
                    cursor.executemany(query, generated_data)
                    inserted_count = cursor.rowcount
                    self.conn.commit()
                    if inserted_count < count:
                        return f"УВАГА: Змогли додати лише
{inserted_count} унікальних записів з {count} запитаних (через
дублікати VIN)."
                    else:
                        return f"Успішно згенеровано та додано
{inserted_count} унікальних записів."

            except Exception as e:
                self.conn.rollback()
                return f"Помилка при пакетній вставці: {e}"

    def generate_drivers(self, count):
        query = """
                INSERT INTO driver (license_number, surname, name,
license_category)
                SELECT 'DR' || trunc(100000 + random() *
900000)::text, \
                       (ARRAY ['Іваненко', 'Петренко', 'Сидоренко',
'Ковальчук', 'Шевченко'])[trunc(random() * 5) + 1], \
                       (ARRAY ['Петро', 'Олександр', 'Михайло',
'Іван', 'Сергій'])[trunc(random() * 5) + 1], \
                       (ARRAY ['B', 'C', 'CE'])[trunc(random() * 3) +
1]
                FROM generate_series(1, %s) \
                """
        return self._generate_data(query, count)

    def generate_routes(self, count):
        query = """
                INSERT INTO route (departure_point, destination_point,
distance_km)
                SELECT (ARRAY ['Київ', 'Львів', 'Одеса', 'Харків',
```

```
'Дніпро'])[trunc(random() * 5) + 1], \
                        (ARRAY ['Варшава', 'Берлін', 'Прага', 'Відень',
'Краків'])[trunc(random() * 5) + 1], \
                        trunc(300 + random() * 1200)::int
                    FROM generate_series(1, %s) \
                    """
            return self._generate_data(query, count)

        def generate_customers(self, count):
            query = """
                    INSERT INTO customer (full_name, phone, email,
address)
                    SELECT 'ТОВ ' || chr(trunc(65 + random() * 25)::int)
|| chr(trunc(65 + random() * 25)::int) || \
                        chr(trunc(65 + random() * 25)::int), \
                        '+380' || trunc(100000000 + random() *
900000000)::text, \
                        LEFT(MD5(random()::text), 10) || '@gmail.com',
\
                        'м. Київ, вул. ' || (ARRAY ['Хрещатик',
'Сумська', 'Дерибасівська'])[trunc(random() * 3) + 1] || \
                        ', ' || trunc(1 + random() * 100)::text
                    FROM generate_series(1, %s) \
                    """
            return self._generate_data(query, count)


        def generate_trips(self, count):

                if not self.conn:
                    return "Помилка: Немає з'єднання з БД."

                print("Отримання списків існуючих ID...")
                car_ids = self._get_existing_ids("car", "car_id")
                driver_ids = self._get_existing_ids("driver", "driver_id")
                route_ids = self._get_existing_ids("route", "route_id")
                customer_ids = self._get_existing_ids("customer",
"customer_id")

                if not all([car_ids, driver_ids, route_ids,
customer_ids]):
                    return ("Помилка: Неможливо згенерувати 'trip'. "
                            "Одна або декілька батьківських таблиць
порожні.")

                generated_data = []
                cargo_options = ['Будматеріали', 'Металопрокат',
'Продукти', 'Техніка', 'Хімікати']
                print(f"Генерація {count} записів 'trip' в Python...")
                for _ in range(count):
                    departure_date = datetime.date.today() -
datetime.timedelta(days=random.randint(0, 30))
                    arrival_date = departure_date +
datetime.timedelta(days=random.randint(1, 10))
                    return_date = arrival_date +
datetime.timedelta(days=random.randint(0, 5))
                    cargo_desc = random.choice(cargo_options)
                    cargo_weight = random.randint(50, 249) * 100
```

```
                car_id = random.choice(car_ids)
                driver_id = random.choice(driver_ids)
                route_id = random.choice(route_ids)
                customer_id = random.choice(customer_ids)

                generated_data.append((
                    departure_date, arrival_date, return_date,
cargo_desc, cargo_weight,
                    car_id, driver_id, route_id, customer_id
                ))

            query = """
                    INSERT INTO trip (departure_date, arrival_date,
return_date, cargo_description, cargo_weight,
                                      car_id, driver_id, route_id,
customer_id)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
                    """

            print(f"Вставка {len(generated_data)} записів у БД...")
            try:
                with self.conn.cursor() as cursor:
                    cursor.executemany(query, generated_data)
                    inserted_count = cursor.rowcount
                    self.conn.commit()
                    return f"Успішно згенеровано та додано
{inserted_count} записів 'trip'."

            except Exception as e:
                self.conn.rollback()
                return f"Помилка при пакетній вставці 'trip': {e}"

    def generate_service(self, count):
        query = """
                INSERT INTO service (car_id, service_date,
description, cost)
                SELECT (SELECT car_id FROM car ORDER BY random() LIMIT
1), \
                       CURRENT_DATE - (random() * 90)::int, \
                       (ARRAY ['Планове ТО', 'Ремонт двигуна', 'Заміна
шин', 'Ремонт гальм'])[trunc(random() * 4) + 1], \
                       trunc(1000 + random() * 15000)::numeric(10, 2)
                FROM generate_series(1, %s) \
                """
        return self._generate_data(query, count)


    def search_trips_complex(self, min_weight, max_weight,
brand_pattern):

        query = """
                SELECT t.trip_id, \
                       t.cargo_description, \
                       t.cargo_weight, \
                       c.brand, \
                       c.license_plate, \
                       d.name || ' ' || d.surname AS driver_full_name
```

```
                FROM trip AS t \
                        JOIN \
                    car AS c ON t.car_id = c.car_id \
                        JOIN \
                    driver AS d ON t.driver_id = d.driver_id
                WHERE t.cargo_weight BETWEEN %s AND %s
                  AND c.brand ILIKE %s; \
                """

        if not self.conn:
            return None, 0, "Помилка: Немає з'єднання з БД."

        try:
            with self.conn.cursor() as cursor:
                start_time = time.time()
                cursor.execute(query, (min_weight, max_weight,
brand_pattern))
                end_time = time.time()
                duration_ms = (end_time - start_time) * 1000

                result = cursor.fetchall()
                self.conn.commit()
                return result, duration_ms, "Пошук успішний."

        except Exception as e:
            self.conn.rollback()
            return None, 0, f"Помилка пошуку: {e}"

    def _get_existing_ids(self, table_name, id_column):
        ids = []
        if not self.conn:
            return ids
        try:
            with self.conn.cursor() as cursor:
                cursor.execute(f"SELECT {id_column} FROM
{table_name}")
                ids = [row[0] for row in cursor.fetchall()]
        except Exception as e:
            print(f"Помилка отримання ID для {table_name}: {e}")
        return ids
```

config.py

```
DB_PARAMS = {
    "dbname": "logistic",
    "user": "postgres",
    "password": "1111",
    "host": "localhost",
    "port": "5432"
}
```

Контакти:

Репозиторій GIT: https://github.com/OverdrafT/kpi_db_and_managment

Telegram: @S_Eugene_S