

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: Ляшенко Д. Д.

Перевірив: Павловський В. І.

**Київ 2025**

**Мета роботи**

Метою даної роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

**Постановка задачі**

В рамках даної розрахунково-графічної роботи мають бути вирішені наступні задачі:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

**Опис предметної галузі**

Тема бази даних: Онлайн-платформа для здачі та оренди нерухомості. На заданій платформі людина (користувач) може одночасно виступати як орендарем, так і тим, хто здає нерухомість в оренду. Користувач може як бронювати, так і здавати в оренду нерухомість. Нерухомість може мати різні відгуки від користувачів. У даної моделі платформи передбачені умови, за які нерухомість може бути орендованою.

**При створенні даної бази даних було виділено такі сутності:**

1. **Користувач (User)** – представляє клієнтів платформи:

* Атрибути: ім'я, прізвище, електронна пошта, дата реєстрації.

1. **Нерухомість (Realty)** – представляє об’єкт нерухомості (квартира, будинок, тощо):

* Атрибути: тип нерухомості (квартира, будинок, тощо), назва міста де знаходиться нерухомість, вулиця місцезнаходження нерухомості, статус нерухомості (здається в оренду, не здається в оренду, орендується), мінімальний строк оренди, депозит(0, сума депозиту), дозволені умови (примітка від власника нерухомості), ціна, термін внесення цінової політики (місяць, доба).

1. **Бронювання (Booking)** – представляє можливість орендування нерухомості:

* Атрибути: дата початку, дата завершення, статус (очікує підтвердження, підтверджено, відхилено, завершено), сума угоди.

1. **Відгуки (Review)** – представляє об’єкт оцінки нерухомості:

* Атрибути: рейтинг (1-5).

1. **Власник нерухомості (Property owner)** – представляє собою користувача, що здає нерухомість в оренду:

* Атрибути: ім’я, прізвище, електронна пошта, дата реєстрації.

**Зв’язки:**

Зв’язок «Власник» - «Нерухомість»:

- Тип зв'язку: 1 до N (один власник може мати безліч нерухомостей; одна нерухомість відноситься до одного власника).

Зв’язок «Нерухомість» - «Відгук»:

- Тип зв'язку: 1 до N (одна нерухомість може мати безліч відгуків; один відгук відноситься до однієї нерухомості).

Зв’язок «Користувач» - «Відгук»:

- Тип зв'язку: 1 до N (один користувач може залишати безліч відгуків; один відгук належить одному користувачу).

Зв’язок «Власник» - «Нерухомість» - «Користувач»:

- Даний зв’язок є сутністю з ім’ям Бронювання та відповідними йому атрибутами. Даний зв’язок є неділимим. Сенс цього зв’язку існує лише у повному наборі сутностей, які в ньому беруть участь. Розглянемо 3 зв’язки між 2 сутностями, які ми можемо отримати в рамках даного зв’язку:

Зв’язок «Власник» - «Бронювання»:

- Тип зв’язку: 1 до N (один власник може мати багато бронювань; одне бронювання належить конкретному власнику).

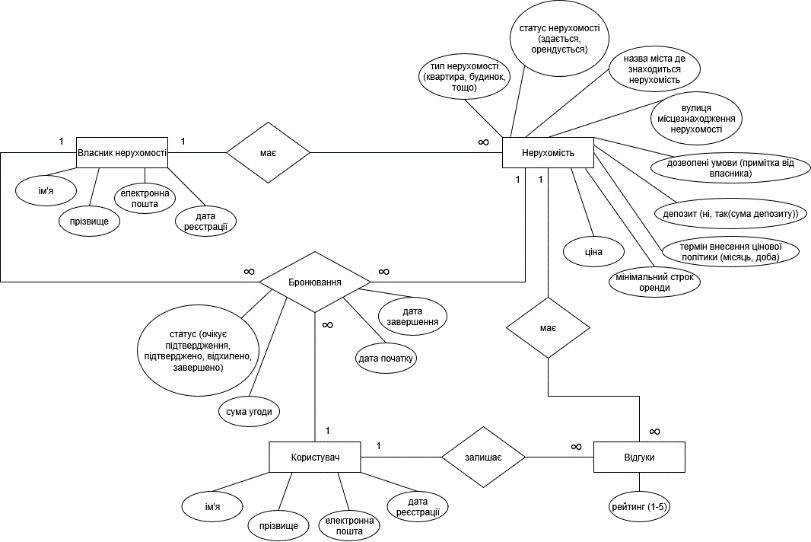
Зв’язок «Нерухомість» - «Бронювання»:

- Тип зв’язку: 1 до N (одна нерухомість може бути заброньованою багато разів; одне бронювання відноситься до однієї конкретної нерухомості).

Зв’язок «Користувач» - «Бронювання»:

- Тип зв’язку: 1 до N (один користувач може робити багато бронювань; одне бронювання відноситься до одного користувача).

Графічне подання концептуальної моделі «Сутність-зв’язок» (ER-діаграма, побудована за нотацією “Пташина лапка”) зображено на рисунку:



*Рисунок 1 - Концептуальна модель «Сутність-зв’язок»*

***Перетворення концептуальної моделі у логічну модель та схему бази даних***

Сутність "Користувач" перетворено в таблицю User з первинним ключем user\_id та атрибутами first\_name, last\_name, data\_registration, email.

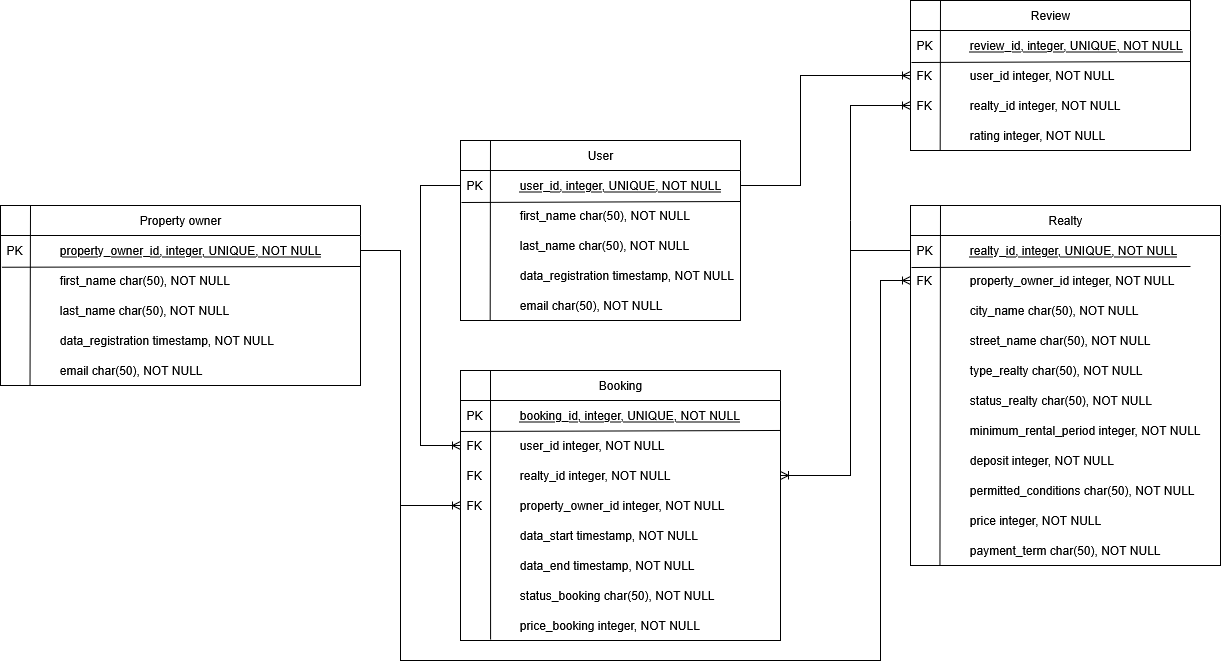
Сутність "Нерухомість" перетворено в таблицю Realty з первинним ключем realty\_ id та атрибутами type\_realty, status\_realty, city\_name, street\_name, minimum\_rental\_period, deposit, permitted\_conditions, price, payment\_term.. У таблиці є зовнішній ключ property\_owner\_id, який пов’язаний з таблицею Property owner відповідно.

Сутність "Бронювання" перетворено в таблицю Booking з первинним ключем booking\_id та атрибутами data\_start, data\_end, status\_booking, price\_booking. У таблиці є зовнішні ключі property\_owner\_id, user\_id, realty\_id, які пов’язані з таблицями Property owner, User, Realty відповідно.

Сутність “Відгуки” перетворено в таблицю Review з первинним ключем review\_id та атрибутами rating. У таблиці є зовнішні ключі user\_id та realty\_id, які пов’язані з таблицями User та Realty відповідно.

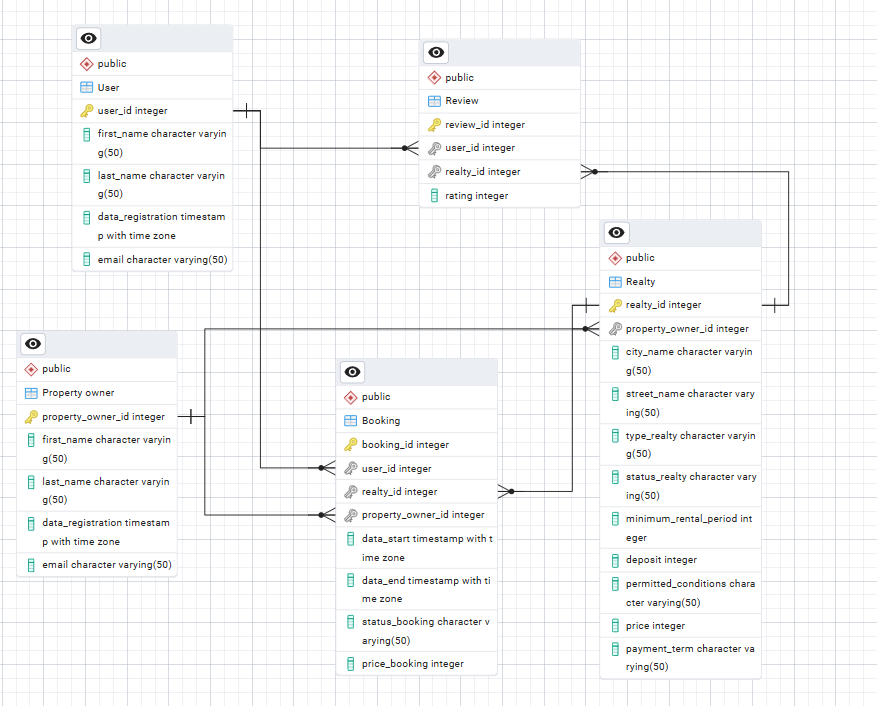
Сутність “Власник нерухомості” перетворено в таблицю Property owner з первинним ключем property\_owner\_id та атрибутами first\_name, last\_name, data\_registration, email.

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



*Рисунок 2 - Логічна модель «Сутність-зв’язок»*

Графічне подання схеми бази даних у pgAdmin 4 зображено на рисунку:



*Рисунок 3 - Схема бази даних у pgAdmin 4*

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки Visual Studio Code, стороння бібліотека для доступу до PostgreSQL – psycopg2, вбудована бібліотека, задля можливості використання таймера спрацювання функцій – time, а також стороння бібліотека, що надає можливість коректного виводу інформації таблиць – tabulate. Встановити бібліотеку tabulate можна шляхом написання наступної команди до консолі користувача: pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org tabulate

**Шаблон проектування**

MVC – Шаблон проектування, який був використаний при проектуванні застосунку.

Model – Представляє собою клас, що описує логіку використання даних, отриманих від користувача, а також тих даних, що початково закладені у саму програму.

View – Представляє собою клас комунікації із користувачем, а також виводу необхідної інформації.

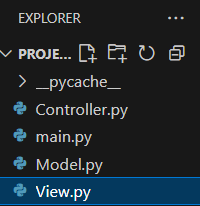
Controller – Представляє собою клас, що описує головну логіку взаємодії усіх інших класів та модулів.

**Структура програми та її опис**

Подання структури програми зображено на рисунках нижче:

****

*Рисунок 4 – Структура програми*



*Рисунок 5 – Структура програми*

Програма умовно поділена на 4 модулі:

1. Файл main.py;

2. Файл Model.py;

3. Файл View.py;

4. Файл Controller.py;

Файл main.py являє собою модуль початкової точки запуску усієї програми. Саме у цьому файлі відбувається підключення головного командного модуля MVC шаблону – Controller.py

Файл Controller.py являє собою модуль керування, що визначає порядок дій, а також порядок викликів необхідних функцій та методів інших модулів.

Файл View.py являє собою модуль комунікації із користувачем. Даний модуль виводить необхідну інформацію до користувача, отримує від користувача дані, після чого готує отримані дані до відправки в модуль Model.py.

Файл Model.py являє собою модуль взаємодії із базою даних. Даний модуль отримує дані від модуля View.py, після чого починає спілкування із базою даних, аналізує отримані дані, та надсилає їх до назад до модуля View.py, аби вивести результати до користувача.

**Структура меню програми**

Меню користувача складається із загальної інформації про структуру усіх наявних таблиць, їх стовпців та відповідних типів, а також із 5-ти пунктів, якими може скористатися користувач.

Консольне подання структури користувацького меню програми зображено на рисунку нижче:



*Рисунок 6 – Структура меню програми*

Нижче розглянемо код кожного модуля, проаналізуємо їх поведінку, можливі ситуації та результати роботи.

**Модуль main.py**

Даний модуль визначає початкову точку запуску програми

# Стартова точка роботи програми

from Controller import Controller

def main():

    controller = Controller()

    controller.run()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Модуль Controller.py**

Даний модуль визначає порядок дій, порядок виклику необхідних методів та функцій у результаті внесених даних користувача та поведінки бази даних.

# Логіка спілкування з користувачем

from View import View

from Model import Model

from time import sleep

class Controller:

    def \_\_init\_\_ (self):

        self.view = View()

        self.model = Model()

        self.main\_input = ""

        self.action\_input = ""

    def run(self):

        try:

            self.model.start\_script\_method()

        except:

            print("database is already created\n")

            sleep(3)

        while True:

            try:

                self.view.clear\_console()

                self.view.show\_base\_info(self.model.get\_tables\_info())

                self.view.print\_action\_menu()

                self.action\_input = self.view.get\_input()

                match self.action\_input:

                    case "1":

                        self.model.get\_table\_data(\*self.view.get\_table\_description())

                        self.view.clear\_console()

                        self.view.show\_table\_data(\*self.model.analysis\_table\_data())

                    case "2":

                        self.model.add\_table\_data(\*self.view.get\_ADD\_description())

                    case "3":

                        self.model.delete\_table\_data(\*self.view.get\_DELETE\_description())

                    case "4":

                        self.model.update\_table\_data(\*self.view.get\_UPDATE\_description())

                    case "5":

                        self.model.generate\_data(self.view.get\_generate\_info())

            except:

                print("\n# - Syntax ERROR . . .")

                sleep(2)

                self.model.connect.rollback()

**Модуль View.py**

Даний модуль виконує вивід необхідної інформації до користувача, а також отримує дані від самого користувача, після чого перенаправляє їх до модуля Model.py, задля наступного аналізу.

Ініціалізація роботи

def \_\_init\_\_ (self):

        self.name\_table = ""

        self.columns = ""

        self.conditions = ""

        self.order\_by = ""

        self.number\_rows = ""

        self.values = ""

Отримання обраної опції

def get\_input(self):

        return input("Choose your option: ")

Очищення консолі

def clear\_console(self):

        print("\033[2J\033[H", end="")

Аналіз отриманих даних від модуля Model.py

    def get\_types(self, number\_table: int, info\_list: list):

        return [(types[1] for types in info\_list[number\_table])]

    def get\_names(self, number\_table: int, info\_list: list):

        return [names[0] for names in info\_list[number\_table]]

Вивід меню користувача

def print\_action\_menu(self):

        print("\n----- MENU -----")

        print("1. Get table data")

        print("2. Add new data")

        print("3. Delete data")

        print("4. Update data")

        print("5. Generate random data")

Отримання опису таблиці задля виводу її витримки

def get\_table\_description(self):

        self.name\_table = input("\nFROM 'table\_name': ").strip()

        self.columns = input("SELECT 'columns\_names' or 'ALL': ").strip()

        self.conditions = input("WHERE 'conditions' or 'NONE': ").strip()

        self.order\_by = input("ORDER BY 'columns\_names' or 'NONE': ").strip()

        self.number\_rows = input("LIMIT 'number\_rows' or 'ALL': ").strip()

        return self.name\_table, self.columns, self.number\_rows, self.order\_by, self.conditions

Отримання кількості згенерованих даних

def get\_generate\_info(self):

        return input("Enter number of new data: ").strip()

Отримання опису додавання, видалення, оновлення даних відповідно

def get\_ADD\_description(self):

        self.name\_table = input("\nINSERT INTO 'table\_name': ").strip()

        self.values = input("VALUES 'value\_1,value\_2,...;': ").strip()

        return self.name\_table, self.values

    def get\_DELETE\_description(self):

        self.name\_table = input("FROM 'table\_name': ").strip()

        self.conditions = input("WHERE 'conditions': ").strip()

        return self.name\_table, self.conditions

    def get\_UPDATE\_description(self):

        self.name\_table = input("UPDATE 'table\_name': ").strip()

        self.columns = input("SET 'column\_1, column\_2, column\_3, ...': ").strip()

        self.values = input("VALUES 'value\_1, value\_2, value\_3, ...': ").strip()

        self.conditions = input("WHERE 'conditions': ").strip()

        return self.name\_table, self.columns, self.values, self.conditions

Вивід загальної інформації про структуру таблиць та типів до відповідних колонок

def show\_base\_info(self, info\_list: list):

        print("### --- You can choose out of these tables:")

        print("### --- TABLE:  Users")

        print(tabulate(self.get\_types(0, info\_list), headers = self.get\_names(0, info\_list), tablefmt = "psql"))

        print("### --- TABLE: Review")

        print(tabulate(self.get\_types(1, info\_list), headers = self.get\_names(1, info\_list), tablefmt = "psql"))

        print("### --- TABLE: Realty")

        print(tabulate(self.get\_types(2, info\_list), headers = self.get\_names(2, info\_list), tablefmt = "psql"))

        print("### --- TABLE: Property owner")

        print(tabulate(self.get\_types(3, info\_list), headers = self.get\_names(3, info\_list), tablefmt = "psql"))

        print("### --- TABLE: Booking")

        print(tabulate(self.get\_types(4, info\_list), headers = self.get\_names(4, info\_list), tablefmt = "psql"))

Вивід конкретних даних таблиці за бажанням користувача

def show\_table\_data(self, columns\_names: list, data\_row: list):

        print(f'### --- TABLE: {self.name\_table}')

        print(tabulate(data\_row, headers = columns\_names, tablefmt = "psql"))

        input()

**Модуль Model.py**

Даний модуль відповідає за головний аналіз отриманих даних від користувача, комунікацію із базою даних, а також надсилання результатів аналізу та спілкування із базою даних до модуля виводу інформації.

Ініціалізація

def \_\_init\_\_(self):

        self.connect = psycopg.connect(

            dbname="postgres",

            user="postgres",

            password="11111",

            host="localhost",

            port="5432"

        )

        self.cursor = self.connect.cursor()

        self.types\_dict = {

            23: "INT",          # integer

            1043: "CHAR VAR",   # character varying

            1184: "TS WITH TZ"  # timestamp with timezone

        }

        self.search\_fk\_query = """

        SELECT

            kcu.column\_name AS foreign\_key\_column,

            ccu.table\_name AS parent\_table,

            ccu.column\_name AS parent\_column

        FROM

            information\_schema.key\_column\_usage AS kcu

        JOIN

            information\_schema.referential\_constraints AS rc

                ON kcu.constraint\_name = rc.constraint\_name

        JOIN

            information\_schema.constraint\_column\_usage AS ccu

                ON rc.unique\_constraint\_name = ccu.constraint\_name

        WHERE

            kcu.table\_schema = 'public'

            AND kcu.table\_name = %s;

        """

        self.start\_script = """

        BEGIN;

        CREATE TABLE IF NOT EXISTS public."Property owner"

        (

        property\_owner\_id integer NOT NULL,

        first\_name character varying(50) NOT NULL,

        last\_name character varying(50) NOT NULL,

        data\_registration timestamp with time zone NOT NULL,

        email character varying(50) NOT NULL,

        CONSTRAINT property\_owner\_id\_pk PRIMARY KEY (property\_owner\_id)

        );

        CREATE TABLE IF NOT EXISTS public."Users"

        (

        user\_id integer NOT NULL,

        first\_name character varying(50) NOT NULL,

        last\_name character varying(50) NOT NULL,

        data\_registration timestamp with time zone NOT NULL,

        email character varying(50) NOT NULL,

        CONSTRAINT user\_id\_pk PRIMARY KEY (user\_id)

        );

        CREATE TABLE IF NOT EXISTS public."Realty"

        (

        realty\_id integer NOT NULL,

        property\_owner\_id integer NOT NULL,

        city\_name character varying(50) NOT NULL,

        street\_name character varying(50) NOT NULL,

        type\_realty character varying(50) NOT NULL,

        status\_realty character varying(50) NOT NULL,

        minimum\_rental\_period integer NOT NULL,

        deposit integer NOT NULL,

        permitted\_conditions character varying(50) NOT NULL,

        price integer NOT NULL,

        payment\_term character varying(50) NOT NULL,

        CONSTRAINT realty\_id\_pk PRIMARY KEY (realty\_id)

        );

        CREATE TABLE IF NOT EXISTS public."Review"

        (

        review\_id integer NOT NULL,

        user\_id integer NOT NULL,

        realty\_id integer NOT NULL,

        rating integer NOT NULL,

        CONSTRAINT review\_id\_pk PRIMARY KEY (review\_id)

        );

        CREATE TABLE IF NOT EXISTS public."Booking"

        (

        booking\_id integer NOT NULL,

        user\_id integer NOT NULL,

        realty\_id integer NOT NULL,

        property\_owner\_id integer NOT NULL,

        data\_start timestamp with time zone NOT NULL,

        data\_end timestamp with time zone NOT NULL,

        status\_booking character varying(50) NOT NULL,

        price\_booking integer NOT NULL,

        CONSTRAINT booking\_id\_pk PRIMARY KEY (booking\_id)

        );

        ALTER TABLE IF EXISTS public."Realty"

        ADD CONSTRAINT property\_owner\_fk FOREIGN KEY (property\_owner\_id)

        REFERENCES public."Property owner" (property\_owner\_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        NOT VALID;

        ALTER TABLE IF EXISTS public."Review"

        ADD CONSTRAINT user\_id\_fk FOREIGN KEY (user\_id)

        REFERENCES public."Users" (user\_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        NOT VALID;

        ALTER TABLE IF EXISTS public."Review"

        ADD CONSTRAINT realty\_id\_fk FOREIGN KEY (realty\_id)

        REFERENCES public."Realty" (realty\_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        NOT VALID;

        ALTER TABLE IF EXISTS public."Booking"

        ADD CONSTRAINT user\_id\_fk FOREIGN KEY (user\_id)

        REFERENCES public."Users" (user\_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        NOT VALID;

        ALTER TABLE IF EXISTS public."Booking"

        ADD CONSTRAINT realty\_id\_fk FOREIGN KEY (realty\_id)

        REFERENCES public."Realty" (realty\_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        NOT VALID;

        ALTER TABLE IF EXISTS public."Booking"

        ADD CONSTRAINT property\_owner\_id\_fk FOREIGN KEY (property\_owner\_id)

        REFERENCES public."Property owner" (property\_owner\_id) MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE NO ACTION

        NOT VALID;

        END;

        """

Запуск створення таблиць, за умови їх відсутності

def start\_script\_method(self):

        self.cursor.execute(self.start\_script)

        self.connect.commit()

Метод генерування запиту отримання інформації про структуру таблиці

def get\_table\_schema\_info(self, table\_name: str) -> list:

        command = f'SELECT \* FROM "{table\_name}" LIMIT 0;'

        self.cursor.execute(command)

        columns\_names = [column\_info[0] for column\_info in self.cursor.description]

        types\_row = [self.types\_dict.get(column\_info[1], "UNKNOWN") for column\_info in self.cursor.description]

        return list(zip(columns\_names, types\_row))

Запуск попереднього методу

def get\_tables\_info(self) -> list:

        table\_names = ["Users", "Review", "Realty", "Property owner", "Booking"]

        return [self.get\_table\_schema\_info(name) for name in table\_names]

Модуль генерування даних

def generate\_data(self, number\_of\_rows\_str: str):

        try:

            num = int(number\_of\_rows\_str)

            if num <= 0:

                print("\n# -> Кількість рядків повинна бути додатним числом.")

                input("Натисніть Enter для продовження...")

                return

        except (ValueError, TypeError):

            print("\n# -> ПОМИЛКА: Введіть коректне число.")

            input("Натисніть Enter для продовження...")

            return

        users\_query = """

        INSERT INTO "Users" (user\_id, first\_name, last\_name, data\_registration, email)

        SELECT

            row\_number() OVER () + (SELECT COALESCE(MAX(user\_id), 0) FROM "Users"),

            fname,

            lname,

            NOW() - (random() \* interval '365 days'),

            lower(fname || '.' || lname || (random()\*1000)::int) || '@generated.com'

        FROM

            (SELECT unnest(array['John', 'Jane', 'Peter', 'Alice', 'Chris', 'Eva']) as fname) f,

            (SELECT unnest(array['Smith', 'Doe', 'Jones', 'Williams', 'Brown', 'Davis']) as lname) l

        ORDER BY random() LIMIT %s;

        """

        property\_owner\_query = """

        INSERT INTO "Property owner" (property\_owner\_id, first\_name, last\_name, data\_registration, email)

        SELECT

            row\_number() OVER () + (SELECT COALESCE(MAX(property\_owner\_id), 0) FROM "Property owner"),

            fname,

            lname,

            NOW() - (random() \* interval '365 days'),

            lower(fname || '.' || lname || (random()\*1000)::int) || '@owner.com'

        FROM

            (SELECT unnest(array['Michael', 'Sarah', 'David', 'Laura', 'James', 'Linda']) as fname) f,

            (SELECT unnest(array['Miller', 'Wilson', 'Moore', 'Taylor', 'Anderson', 'Thomas']) as lname) l

        ORDER BY random() LIMIT %s;

        """

        realty\_query = """

        INSERT INTO "Realty" (realty\_id, property\_owner\_id, city\_name, street\_name, type\_realty, status\_realty,

                              minimum\_rental\_period, deposit, permitted\_conditions, price, payment\_term)

        SELECT

            n + (SELECT COALESCE(MAX(realty\_id), 0) FROM "Realty"),

            (SELECT property\_owner\_id FROM "Property owner" ORDER BY random() LIMIT 1),

            (SELECT city FROM unnest(array['Kyiv', 'Lviv', 'Odesa', 'Kharkiv']) city ORDER BY random() LIMIT 1),

            (SELECT street FROM unnest(array['Main St', 'Oak Ave', 'Pine Ln', 'Maple Dr']) street ORDER BY random() LIMIT 1),

            (SELECT type FROM unnest(array['Apartment', 'House', 'Cottage']) type ORDER BY random() LIMIT 1),

            (SELECT status FROM unnest(array['Available', 'Rented', 'Under repair']) status ORDER BY random() LIMIT 1),

            (1 + random() \* 11)::int,

            (1000 + random() \* 4000)::int,

            (SELECT cond FROM unnest(array['Pets allowed', 'No smoking', 'Families only']) cond ORDER BY random() LIMIT 1),

            (5000 + random() \* 15000)::int,

            (SELECT term FROM unnest(array['Monthly', 'Quarterly']) term ORDER BY random() LIMIT 1)

        FROM generate\_series(1, %s) as s(n);

        """

        review\_query = """

        INSERT INTO "Review" (review\_id, user\_id, realty\_id, rating)

        SELECT

            n + (SELECT COALESCE(MAX(review\_id), 0) FROM "Review"),

            (SELECT user\_id FROM "Users" ORDER BY random() LIMIT 1),

            (SELECT realty\_id FROM "Realty" ORDER BY random() LIMIT 1),

            (1 + random() \* 4)::int

        FROM generate\_series(1, %s) as s(n);

        """

        booking\_query = """

        INSERT INTO "Booking" (booking\_id, user\_id, realty\_id, property\_owner\_id, data\_start, data\_end,

                               status\_booking, price\_booking)

        SELECT

            row\_number() OVER () + (SELECT COALESCE(MAX(booking\_id), 0) FROM "Booking"),

            (SELECT user\_id FROM "Users" ORDER BY random() LIMIT 1),

            realty.realty\_id,

            realty.property\_owner\_id,

            NOW() + (random() \* interval '10 days'),

            NOW() + (random() \* interval '20 days' + interval '11 days'),

            (SELECT status FROM unnest(array['Confirmed', 'Pending', 'Cancelled']) status ORDER BY random() LIMIT 1),

            (5000 + random() \* 15000)::int

        FROM "Realty" as realty

        ORDER BY random() LIMIT %s;

        """

        queries\_in\_order = [

            ("Users", users\_query),

            ("Property owner", property\_owner\_query),

            ("Realty", realty\_query),

            ("Review", review\_query),

            ("Booking", booking\_query)

        ]

        try:

            print("\n# -> Починаю генерацію даних...")

            for table\_name, query in queries\_in\_order:

                if table\_name in ["Realty", "Review", "Booking"]:

                    parent\_check = {

                        "Realty": ('SELECT 1 FROM "Property owner" LIMIT 1', "Property owner"),

                        "Review": ('SELECT 1 FROM "Users" LIMIT 1', "Users"),

                        "Booking": ('SELECT 1 FROM "Realty" LIMIT 1', "Realty")

                    }

                    if table\_name in parent\_check:

                        check\_q, parent\_name = parent\_check[table\_name]

                        self.cursor.execute(check\_q)

                        if self.cursor.fetchone() is None:

                            print(f"# -> Пропускаю \"{table\_name}\", оскільки батьківська таблиця \"{parent\_name}\" порожня.")

                            continue

                print(f"# -> Заповнюю таблицю \"{table\_name}\"...")

                self.cursor.execute(query, (num,))

            self.connect.commit()

            print("\n# -> Успішно згенеровано та додано дані до всіх таблиць!")

        except psycopg.Error as e:

            print(f"\n# -> СТАЛАСЯ ПОМИЛКА: {e.diag.message\_primary}")

            print("# -> Відкочую всі зміни, щоб зберегти цілісність бази даних.")

            self.connect.rollback()

        input("Натисніть Enter для продовження...")

Метод формування та перевірки коректного занесення даних до таблиці

def \_insert\_data\_safely(self, table\_name: str, data: list):

        schema\_info = self.get\_table\_schema\_info(table\_name)

        columns = [col[0] for col in schema\_info]

        cols\_str = ", ".join([f'"{c}"' for c in columns])

        placeholders = ", ".join(["%s"] \* len(columns))

        insert\_query = f'INSERT INTO "{table\_name}" ({cols\_str}) VALUES ({placeholders})'

        try:

            self.cursor.executemany(insert\_query, data)

            self.connect.commit()

            print("\n# -> Дані успішно додано!")

        except Exception as e:

            print(f"\n# -> Помилка під час вставки: {e}")

            self.connect.rollback()

        input("Натисніть Enter для продовження...")

Метод занесення даних до таблиці

def add\_table\_data(self, name\_table: str, values: str):

        schema\_info = self.get\_table\_schema\_info(name\_table)

        columns\_names = [col[0] for col in schema\_info]

        # Конвертуємо вхідний рядок у список кортежів

        try:

            data\_to\_insert = [tuple(row.split(',')) for row in values.strip(';').split(';')]

        except ValueError:

            print("\n# -> Помилка формату вхідних даних.")

            input("Натисніть Enter для продовження...")

            return

        # Знаходимо всі зовнішні ключі для цієї таблиці

        self.cursor.execute(self.search\_fk\_query, (name\_table,))

        foreign\_keys = self.cursor.fetchall()

        if not foreign\_keys:

            # Якщо зовнішніх ключів немає, просто вставляємо дані

            self.\_insert\_data\_safely(name\_table, data\_to\_insert)

            return

        valid\_rows\_to\_insert = []

        for row\_data in data\_to\_insert:

            is\_row\_valid = True

            for fk in foreign\_keys:

                fk\_column\_name, parent\_table, parent\_column = fk

                try:

                    # Знаходимо індекс стовпця, що є зовнішнім ключем

                    fk\_index = columns\_names.index(fk\_column\_name)

                    value\_to\_check = row\_data[fk\_index]

                    # Безпечний запит для перевірки існування батьківського запису

                    check\_query = f'SELECT 1 FROM "{parent\_table}" WHERE "{parent\_column}" = %s LIMIT 1;'

                    self.cursor.execute(check\_query, (value\_to\_check,))

                    if self.cursor.fetchone() is None:

                        print(f'\n# -> ПОМИЛКА: Значення "{value\_to\_check}" для стовпця "{fk\_column\_name}"')

                        print(f'# -> не існує в батьківській таблиці "{parent\_table}". Рядок не буде додано.')

                        is\_row\_valid = False

                        break  # Перериваємо перевірку для поточного рядка

                except ValueError:

                    print(f"# -> КРИТИЧНА ПОМИЛКА: Стовпець {fk\_column\_name} не знайдено в таблиці {name\_table}")

                    is\_row\_valid = False

                    break

            if is\_row\_valid:

                valid\_rows\_to\_insert.append(row\_data)

        if valid\_rows\_to\_insert:

            self.\_insert\_data\_safely(name\_table, valid\_rows\_to\_insert)

        else:

            print("\n# -> Жодного рядка не було додано через помилки валідації.")

            input("Натисніть Enter для продовження...")

Метод видалення даних з таблиці

def delete\_table\_data(self, name\_table: str, conditions: str):

        # Перевіряємо, чи умови не порожні

        if not conditions.strip():

            print("\n# -> ПОМИЛКА: Умови для видалення не можуть бути порожніми!")

            input("Натисніть Enter для продовження...")

            return

        # Створюємо базову команду DELETE

        command = f'DELETE FROM "{name\_table}" WHERE {conditions}'

        try:

            # Особлива логіка ТІЛЬКИ для батьківської таблиці "Realty"

            if name\_table == "Realty":

                # 1. Знайти ID об'єктів нерухомості, які підпадають під умови видалення

                select\_ids\_query = f'SELECT "realty\_id" FROM "Realty" WHERE {conditions}'

                self.cursor.execute(select\_ids\_query)

                realty\_ids\_to\_delete = self.cursor.fetchall()

                if not realty\_ids\_to\_delete:

                    print("\n# -> Жодного запису не знайдено за вашими умовами.")

                    input("Натисніть Enter для продовження...")

                    return

                # Перетворюємо список кортежів [(1,), (2,)] у простий список [1, 2]

                ids\_list = [item[0] for item in realty\_ids\_to\_delete]

                # 2. Перевірити, чи існують посилання на ці ID в дочірніх таблицях

                # Перевірка в таблиці "Review"

                check\_review\_query = 'SELECT 1 FROM "Review" WHERE "realty\_id" = ANY(%s) LIMIT 1'

                self.cursor.execute(check\_review\_query, (ids\_list,))

                if self.cursor.fetchone():

                    print(f'\n# -> ПОМИЛКА: Неможливо видалити. Існують відгуки, пов\'язані з цією нерухомістю.')

                    self.connect.rollback()

                    input("Натисніть Enter для продовження...")

                    return

                # Перевірка в таблиці "Booking"

                check\_booking\_query = 'SELECT 1 FROM "Booking" WHERE "realty\_id" = ANY(%s) LIMIT 1'

                self.cursor.execute(check\_booking\_query, (ids\_list,))

                if self.cursor.fetchone():

                    print(f'\n# -> ПОМИЛКА: Неможливо видалити. Існують бронювання, пов\'язані з цією нерухомістю.')

                    self.connect.rollback()

                    input("Натисніть Enter для продовження...")

                    return

            # 3. Виконуємо видалення (для "Realty" - після перевірок, для інших - одразу)

            self.cursor.execute(command)

            # Перевіряємо, чи було щось видалено

            if self.cursor.rowcount > 0:

                print(f"\n# -> Успішно видалено {self.cursor.rowcount} рядків.")

                self.connect.commit() # Підтверджуємо транзакцію

            else:

                print("\n# -> Жодного рядка не було видалено (можливо, умови не знайшли збігів).")

                self.connect.rollback()

        except psycopg.Error as e:

            print(f"\n# -> Сталася помилка бази даних: {e}")

            self.connect.rollback() # Відкочуємо зміни у разі будь-якої помилки

        input("Натисніть Enter для продовження...")

Метод оновлення даних в таблиці

def update\_table\_data(self, name\_table: str, columns: str, values: str, conditions: str):

        # 1. Валідація та парсинг вхідних даних

        if not all([name\_table, columns, values, conditions]):

            print("\n# -> ПОМИЛКА: Всі поля (таблиця, стовпці, значення, умови) мають бути заповнені!")

            input("Натисніть Enter для продовження...")

            return

        cols\_list = [c.strip() for c in columns.split(',')]

        vals\_list = [v.strip() for v in values.split(',')]

        if len(cols\_list) != len(vals\_list):

            print("\n# -> ПОМИЛКА: Кількість стовпців не відповідає кількості значень!")

            input("Натисніть Enter для продовження...")

            return

        try:

            # 2. Перевірка цілісності даних ПЕРЕД виконанням запиту

            self.cursor.execute(self.search\_fk\_query, (name\_table,))

            foreign\_keys = {fk[0]: (fk[1], fk[2]) for fk in self.cursor.fetchall()}

            for i, col\_name in enumerate(cols\_list):

                if col\_name in foreign\_keys:

                    parent\_table, parent\_column = foreign\_keys[col\_name]

                    value\_to\_check = vals\_list[i]

                    # Перевіряємо, чи існує нове значення в батьківській таблиці

                    check\_query = f'SELECT 1 FROM "{parent\_table}" WHERE "{parent\_column}" = %s LIMIT 1;'

                    self.cursor.execute(check\_query, (value\_to\_check,))

                    if self.cursor.fetchone() is None:

                        print(f'\n# -> ПОМИЛКА: Неможливо оновити стовпець "{col\_name}".')

                        print(f'# -> Значення "{value\_to\_check}" не існує в батьківській таблиці "{parent\_table}".')

                        self.connect.rollback()

                        input("Натисніть Enter для продовження...")

                        return

            set\_clause = ", ".join([f'"{col}" = %s' for col in cols\_list])

            command = f'UPDATE "{name\_table}" SET {set\_clause} WHERE {conditions}'

            # Передаємо значення для SET як параметри, щоб уникнути ін'єкцій

            self.cursor.execute(command, tuple(vals\_list))

            if self.cursor.rowcount > 0:

                print(f"\n# -> Успішно оновлено {self.cursor.rowcount} рядків.")

                self.connect.commit()

            else:

                print("\n# -> Жодного рядка не було оновлено (умови не знайшли збігів).")

                self.connect.rollback()

        except psycopg.Error as e:

            self.connect.rollback()

            # Надаємо користувачу зрозуміле повідомлення про помилку

            if e.pgcode == '23503': # Код помилки для foreign\_key\_violation

                 print(f"\n# -> ПОМИЛКА ОНОВЛЕННЯ: Неможливо змінити цей запис,")

                 print(f"# -> оскільки на нього посилаються дані в інших таблицях (наприклад, бронювання або відгуки).")

            else:

                 print(f"\n# -> Сталася помилка бази даних: {e}")

        input("Натисніть Enter для продовження...")

Метод генерування запиту отримання конкретних даних з таблиці

def get\_table\_data(self, name\_table: str, columns: str, number\_rows: str, order: str, conditions: str):

        selected\_columns = columns if columns != "ALL" else "\*"

        command = f'SELECT {selected\_columns} FROM "{name\_table}"'

        if conditions != "NONE":

            command += f' WHERE {conditions}'

        if order != "NONE":

            command += f' ORDER BY {order}'

        if number\_rows != "ALL":

            if number\_rows.isdigit():

                command += f' LIMIT {number\_rows}'

        self.cursor.execute(command)

Метод аналізу та надсилання отриманих даних до модуля виводу View.py

def analysis\_table\_data(self) -> tuple:

        if not self.cursor.description:

            return [], []

        columns\_names = [column\_info[0] for column\_info in self.cursor.description]

        types\_row = [self.types\_dict.get(column\_info[1], "UNKNOWN") for column\_info in self.cursor.description]

        gotten\_data = self.cursor.fetchall()

        final\_data\_rows = [tuple(types\_row)] + gotten\_data

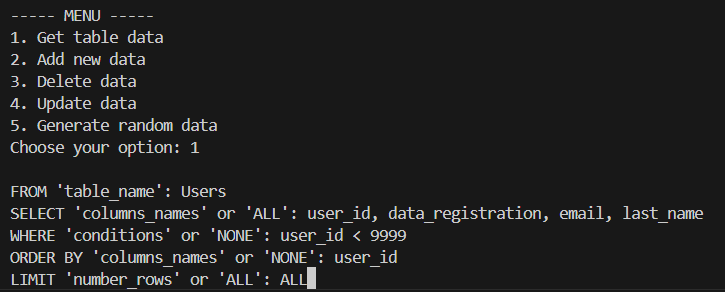
        return columns\_names, final\_data\_rows

**Тестування**

Задля підтвердження коректності виконання програми, необхідно провести тестування кожного з пунктів меню користувача.

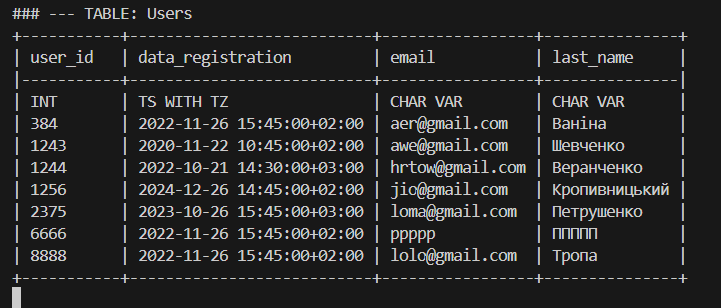
Проведемо тестування 1-го пункту меню. Занесемо необхідні параметри задля отримання конкретних даних таблиць.

Внесення тестових даних 1-го пункту користувацького меню зображено на рисунку нижче:



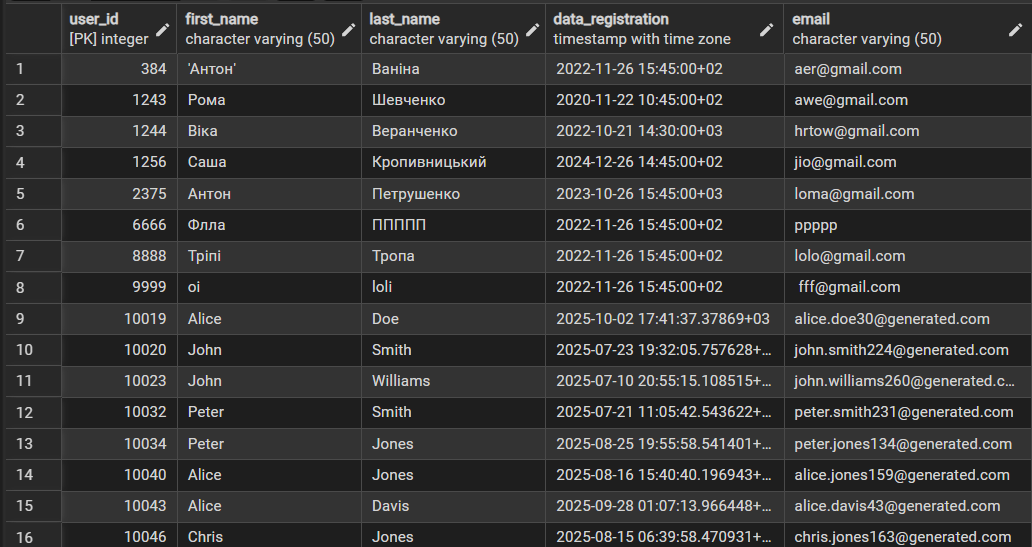
*Рисунок 7 – Внесення тестових даних 1-го пункту користувацького меню*

Результати роботи програми після внесення даних до 2-го пункту користувацького меню зображено на рисунку нижче:



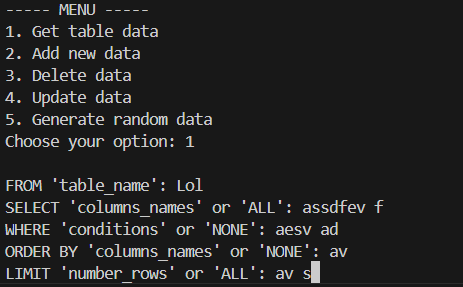
*Рисунок 8 – Результати роботи програми після внесення даних до 1-го пункту користувацького меню*

Порівняння результатів роботи програми та загального вигляду таблиці users у PostgreSQL зображено на рисунку нижче:



*Рисунок 9 – Порівняння результатів роботи програми та загального вигляду таблиці Users у PostgreSQL*

Внесення тестових даних 1-го пункту користувацького меню зображено на рисунку нижче:



*Рисунок 10 – Внесення тестових даних 1-го пункту користувацького меню*

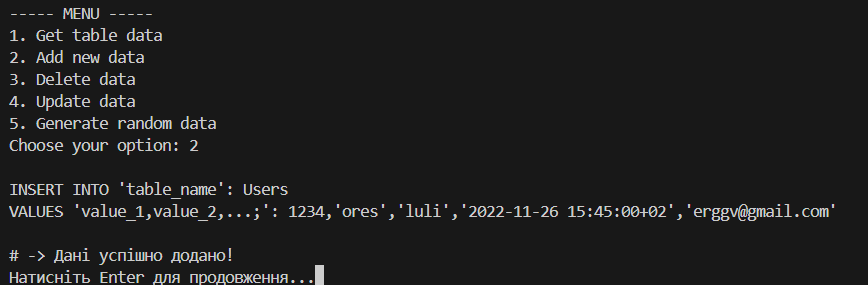
Результат роботи програми внаслідок введення некоректних даних зображено на рисунку нижче:



*Рисунок 11 – Результат роботи програми внаслідок введення некоректних даних*

Проведемо тестування 2-го пункту меню за подібним принципом

Внесення тестових даних 2-го пункту користувацького меню зображено на рисунку нижче:



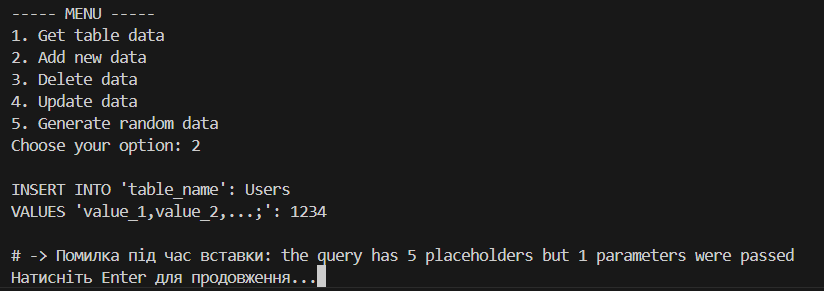
*Рисунок 12 – Внесення тестових даних 2-го пункту користувацького меню*

Результат роботи програми на прикладі отриманих даних PostgreSQL зображено на рисунку нижче:



*Рисунок 13 – Результат роботи програми на прикладі отриманих даних PostgreSQL*

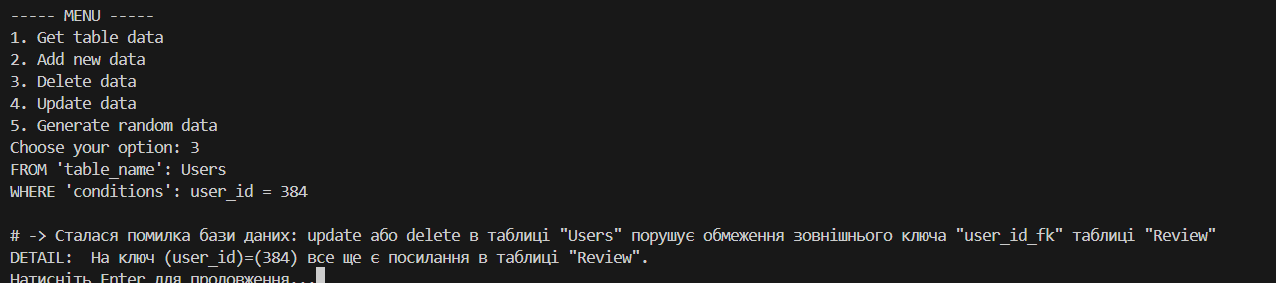
Внесення некоректних даних до 2-го пункту користувацького меню зображено на рисунку нижче:



*Рисунок 14 – Внесення некоректних даних до 2-го пункту користувацького меню*

Проведемо тестування 3-го пункту меню.

Внесення некоректних даних до 3-го пункту користувацького меню зображено на рисунку нижче:

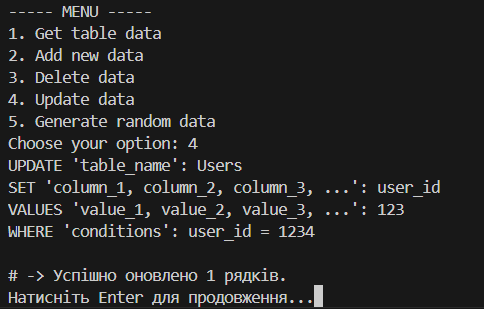


*Рисунок 15 – Внесення некоректних даних до 3-го пункту користувацького меню*

Аби видалити дані з батьківської таблиці, необхідно аби на ці дані не посилалися жодні дочірні таблиці. Таким чином зберігається цілісність даних.

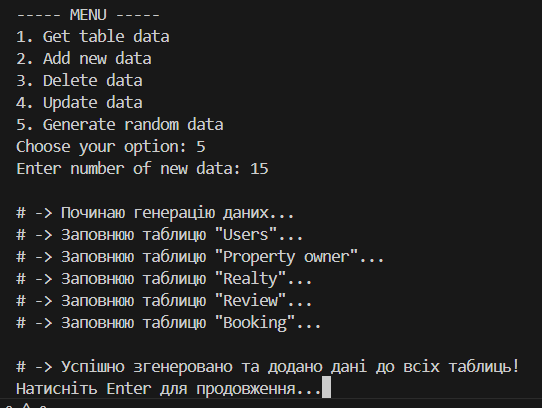
Проведемо тестування 4 та 5 пунктів

Внесення тестових даних до 4-го пункту користувацького меню зображено на рисунку нижче:



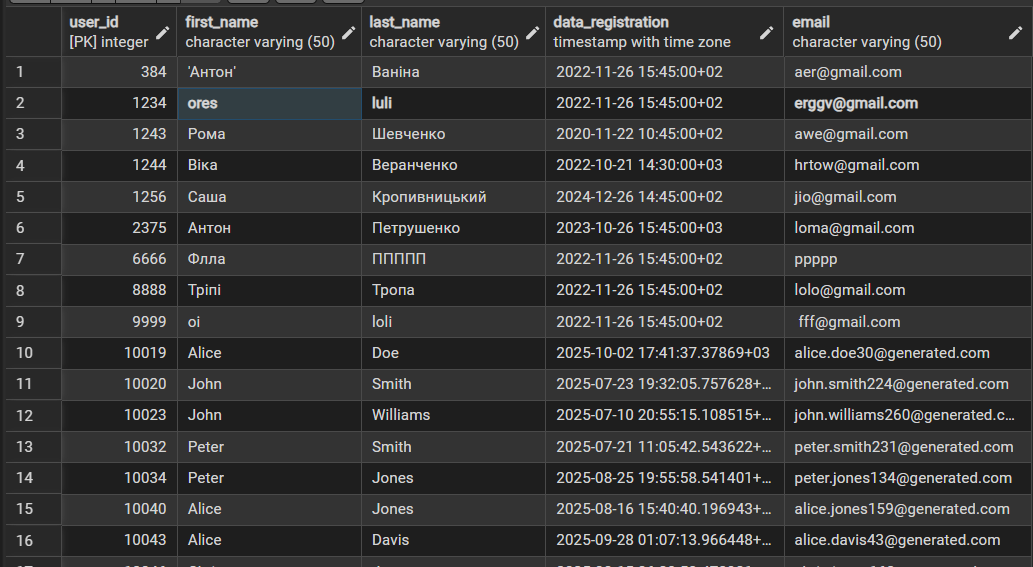
*Рисунок 16 – Внесення тестових даних до 4-го пункту користувацького меню*

Внесення тестових даних до 5-го пункту користувацького меню зображено на рисунку нижче:



*Рисунок 17 – Внесення тестових даних до 5-го пункту користувацького меню*

Результат роботи програми у PostgreSQL після внесення даних до 4-го та 5-го пунктів користувацького меню зображено на рисунку нижче:



*Рисунок 18 – Результат роботи програми у PostgreSQL після внесення даних до 4-го та 5-го пунктів користувацького меню*

Посилання на GitHub:

<https://github.com/liashenkodaniil/->

Контакти в телеграм:

Username: @daniczek

Nickname: Danchik

Електронна пошта:

kyvalich@gmail.com