НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахунков- графічна робота №1

з дисципліни **«Бази даних і засоби управління»**

На тему: "Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL"

Виконав студент III курсу
групи КВ-22
Землянський Е.В.
(Телеграм: @whiskey18)
Перевірив: Павловський В.І.

Київ – 2024

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

Загальне завдання роботи полягає у наступному:

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).

Виконання роботи

Нижче наведено опис сутностей предметної області:

Опис сутностей предметної області

1. **User** – Користувач системи

Атрибути:

- user_id - Ідентифікатор користувача
- name - Ім’я користувача
- surname – Призвіще користувача

Призначення: Збереження даних про користувачів системою

2. **Wallet (Гаманець)** – Гаманець користувача системи, що підв’язаний до його профілю

Атрибути:

- wallet_id - Ідентифікатор
- status - Статус (закритий / відкритий)

Призначення: Збереження даних про гаманці

3. **Investment (Інвестиція)** – Інвестиція, придбана користувачем за рахунок фінансових активів у певній валюті із свого гаманця.

Атрибути:

- investment_id - Ідентифікатор
- date - Дата придбання
- annual_income - Річний дохід
- price - Ціна інвестиції
- seller - Компанія-продавець
- title – Назва інвестиційного проєкту

Призначення: Збереження даних про інвестиції

4. **Currency (Валюта)** – валюта, в якій знаходяться фінансові активи у гаманці користувача

Атрибути:

- currency_id - Ідентифікатор
- currency_name - Назва
- rate - Курс по USD
- quantity – кількість на балансі в певному гаманці

Призначення: Збереження даних про валюти і їх кількість на балансах гаманців

Опис зв'язків між сутностями

Зв'язок «Owner (User) - Wallet»:

1:N: Користувач може створити і мати більше одного гаманця, але гаманець не може належати більше ніж одному користувачу.

Зв'язок «Owner (User) - Investment»:

M:N: Інвестиційний пакет може бути придбаний більше ніж одним користувачем і в будь-якій кількості (доступній, наданій компанією-продавцем)

Зв'язок «Wallet - Currency»:

1:N: Фінанси можуть знаходитись на балансі у гаманці в будь-яких доступних валютах, але будь яка валюта відповідає одному запису зі списку балансів гаманця.

Зв’язок «Investment - Wallet»:

N:1: Інвестиційні пакети можливо придбати у певній кількості за рахунок одного і того ж гаманця

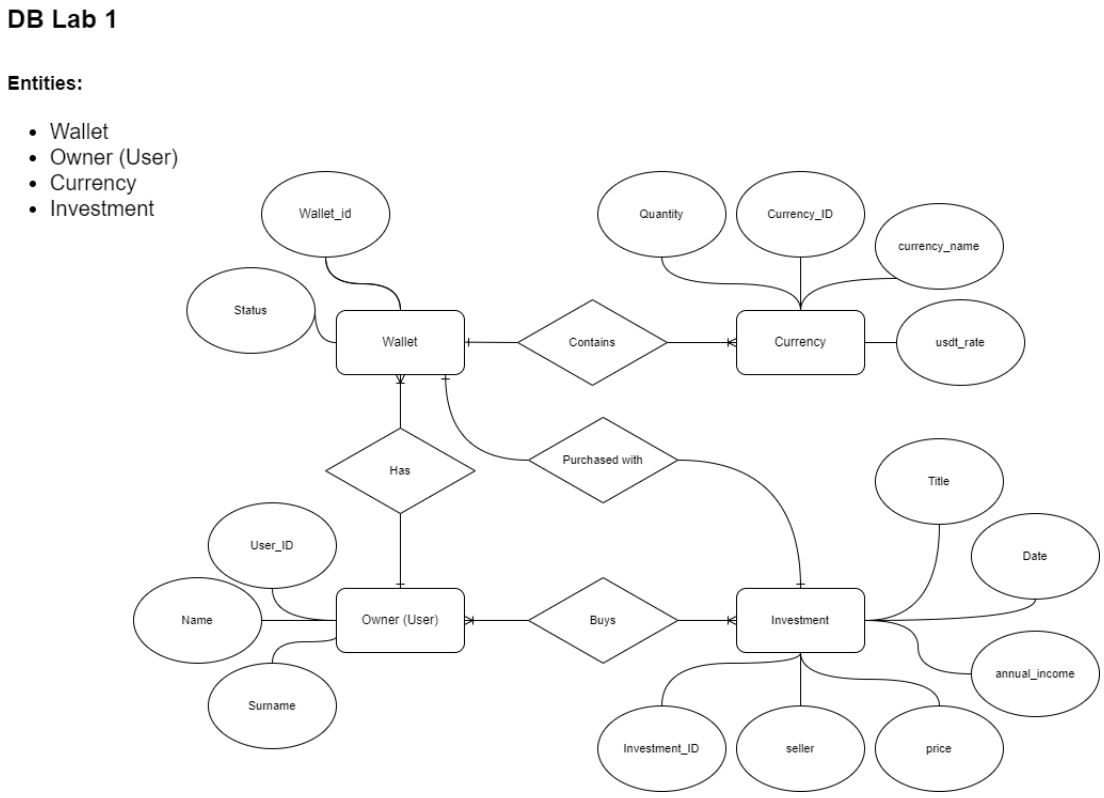


Рисунок 1 - Зображення ER – діаграми побудованої за нотацією Чена

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

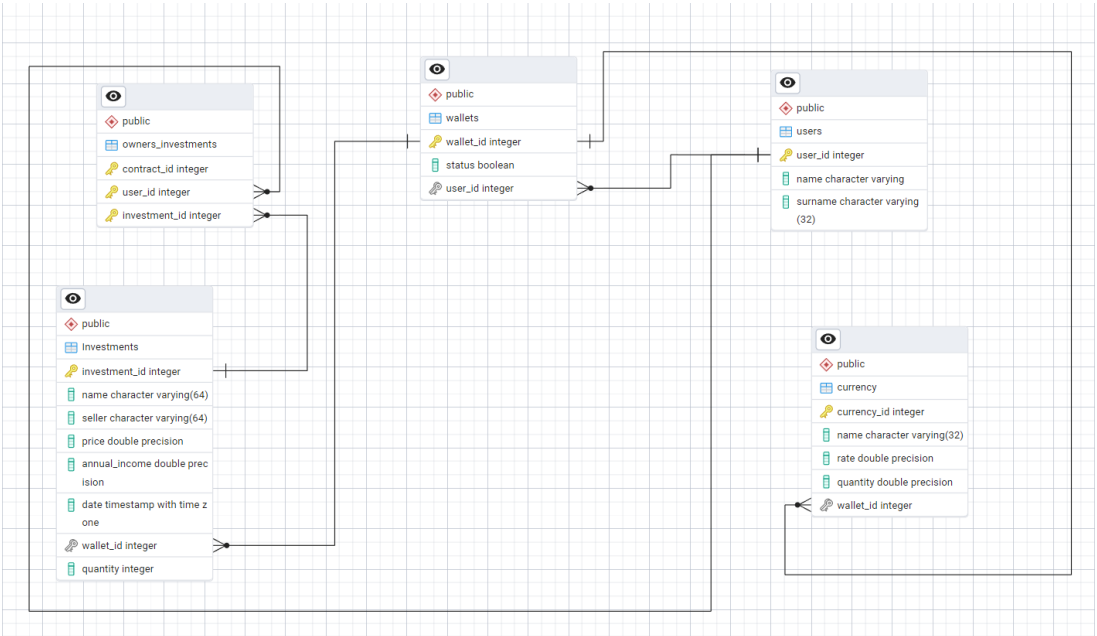


Рисунок 2 – Схема бази даних

Середовище та компоненти розробки

У процесі розробки була використана мова програмування Python, середовище розробки Visual Studio Code, 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.

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

Структура програми та її опис

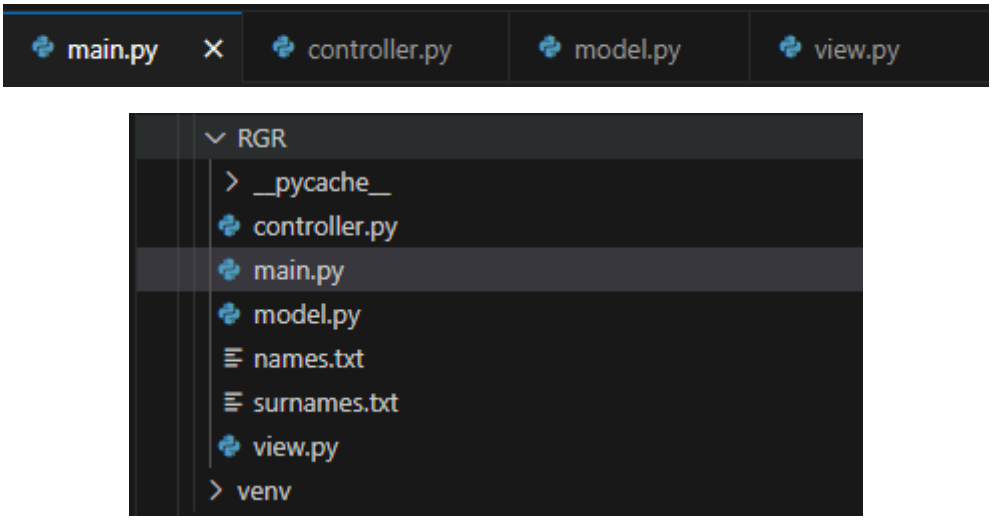


Рисунок 3 – Структура програми

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

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

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

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

Отже, структура програми відповідає патерну MVC.

Структура меню програми

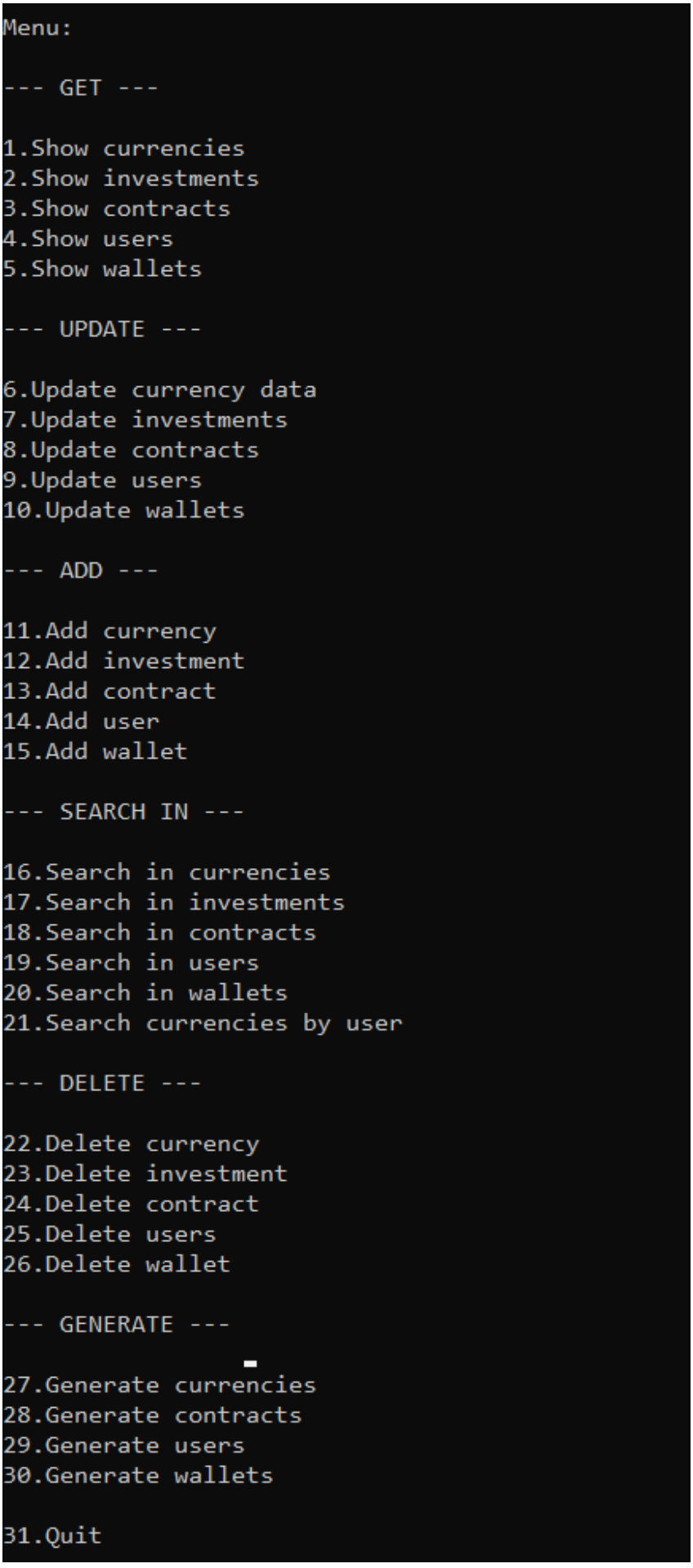


Рисунок 4 – Структура меню користувача

Опис функціональності пунктів меню користувача

Пункти 1 - 5 з відділу меню «GET» :

Відповідають за вивід даних з таблиць користувачу, для перегляду.

Пункти 6 – 10 з відділу меню «UPDATE» :

Відповідають за оновлення (зміну) даних у таблицях.

Пункти 11 – 15 з відділу меню «ADD» :

Відповідають за додавання нових даних у таблиці бази даних.

Пункти 16 – 20 з відділу меню «SEARCH» :

Відповідають за пошук серед даних у таблицях бази даних за різними полями і вивід даних, які відповідають критеріям пошуку, що задані користувачем.

Пункт 21 з відділу меню «SEARCH» :

Відповідає за пошук серед фінансових активів *(таблиці currency)* співставляючи дані з відповідними підв'язаними гаманцями *(з таблиці wallets)* і співставляючи із гаманацями відповідих користувачів (з таблиці users), пошук здійснюється за даними користувача *(user)*, на вибір користувача програми: за унікальним ідентифікатором *(user_id)* або за ім'ям *(name)* і фамілією *(surname)*, можливий неповний ввід даних у випадку імені та фамілії, буде здійснено пошук який містить введені дані.

Пункти 22 – 26 з відділу меню «DELETE» :

Відповідають за видалення даних з таблиць за їх унікальним ідентифікатором. Є можливість видаляти більше одного елементу вказуючи діапазон ідентифікаторів (наприклад з десятого по двадцятий).

Пункти 27 – 30 з відділу меню «GENERATE» :

Відповідають за генерацію даних у відповідні таблиці бази даних з автоматичним створенням унікального ідентифікатора для кожного згенерованого об'єкту.

Пункт 31:

Відповідає за вихід з програми.

Тестування роботи функцій програми в різних ситуаціях

Вилучення

Спроба провести операцію вилучення запису батьківського запису таблиці:

```
Enter your choice: 25

Enter starting user ID to remove: 1

Enter finishing user ID to remove: 1
An error occurred: ПОМИЛКА:  update або delete в таблиці "users" порушує обмеження зовнішнього ключа "wallet_owner" таблиці "wallets"
DETAIL:  На ключ (user_id)=(1) все ще є посилання в таблиці "wallets".

Function 'delete_users_in_range' executed in 46.1082 milliseconds
```

Призводить до помилки, яку в програмі перехоплено і виведено користувачу, після чого програма повертається до головного меню і готова до подальшої роботи

```
Enter your choice: 4

Enter starting ID to show from: 1

Enter ending currency ID to show: 1
Function 'get_users' executed in 6.0036 milliseconds
Users:

User ID: 1
Name: Edward
Surname: Zemlyanski
```

Спробувавши вивести дані користувача, якого щойно намагались вивести, бачимо, що він не зник.

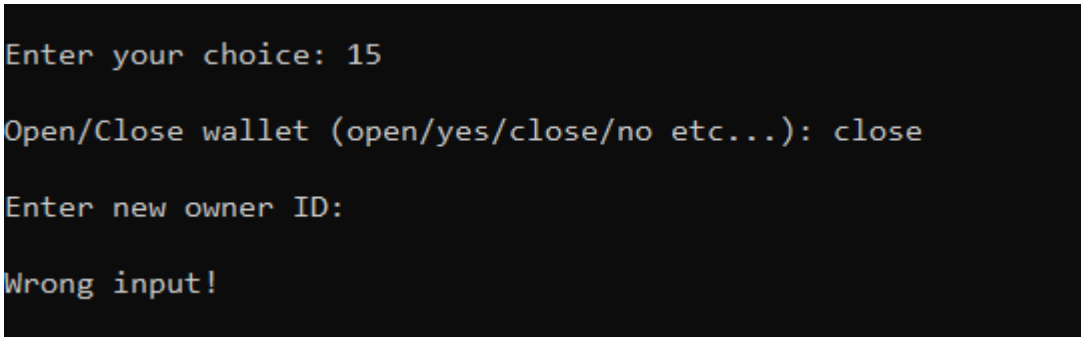
Створення

При спробі створити новий запис у таблиці, який посилається через зовнішній ключ на інші записи інших таблиць користувач програми забов’язанний ввести ідентифіктор запису з іншої таблиці на який посилатиметься новостворений запис:

```
Enter your choice: 15

Open/Close wallet (open/yes/close/no etc...): open

Enter new owner ID: 2

1 wallets added.

Function 'add_wallet' executed in 5.8138 milliseconds
```

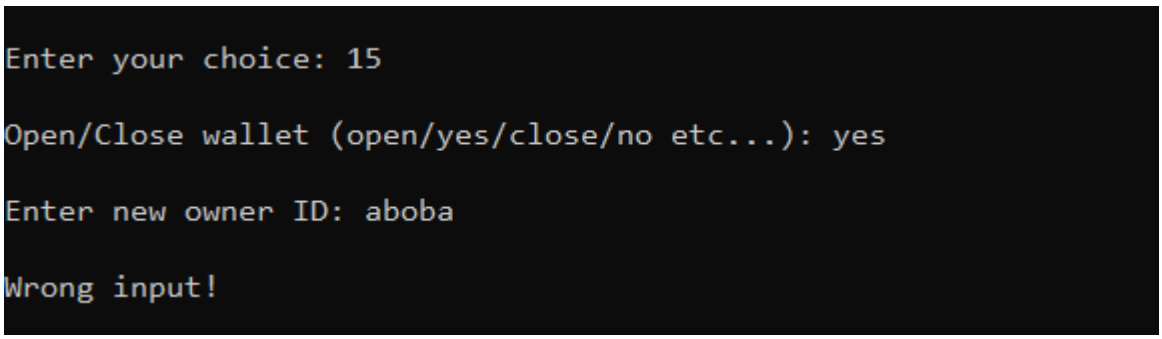
В такому випадку запис буде успішно створено.

При спробі створити новий запис без введеня ідентифікатора запису з іншої таблиці на який має посилатись запис, що створюється, буде виведено повідомлення про помилковий ввід даних користувачем:



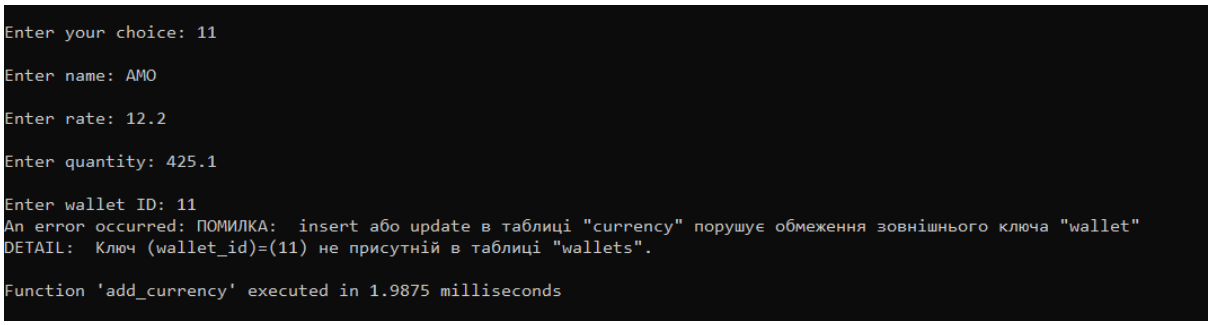
В такому випадку запис не буде створено і програма повернеться у головне меню, будучи готовою до подільшої роботи.

При спробі створити новий запис уввівши замість очікуваного ідентифікатора запису з іншої таблиці на який має посилатись запис, що створюється, аналогічно буде виведено повідомлення про помилковий ввід даних користувачем:



В такому випадку запис не буде створено і програма повернеться у головне меню, будучи готовою до подільшої роботи.

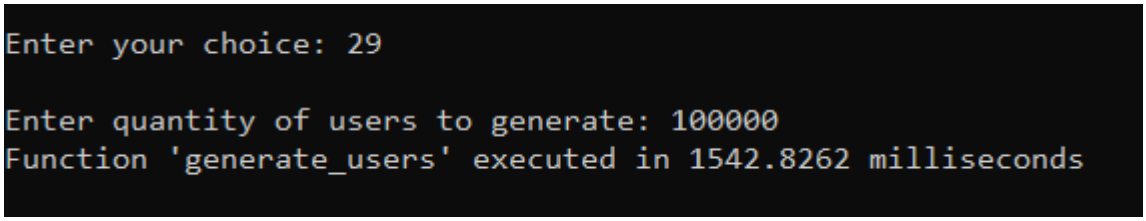
При спробі створити новий запис уввівши неіснуючий ідентифікатор запису з іншої таблиці на який має посилатись запис, що створюється, буде виведено повідомлення про помилковий ввід даних користувачем:



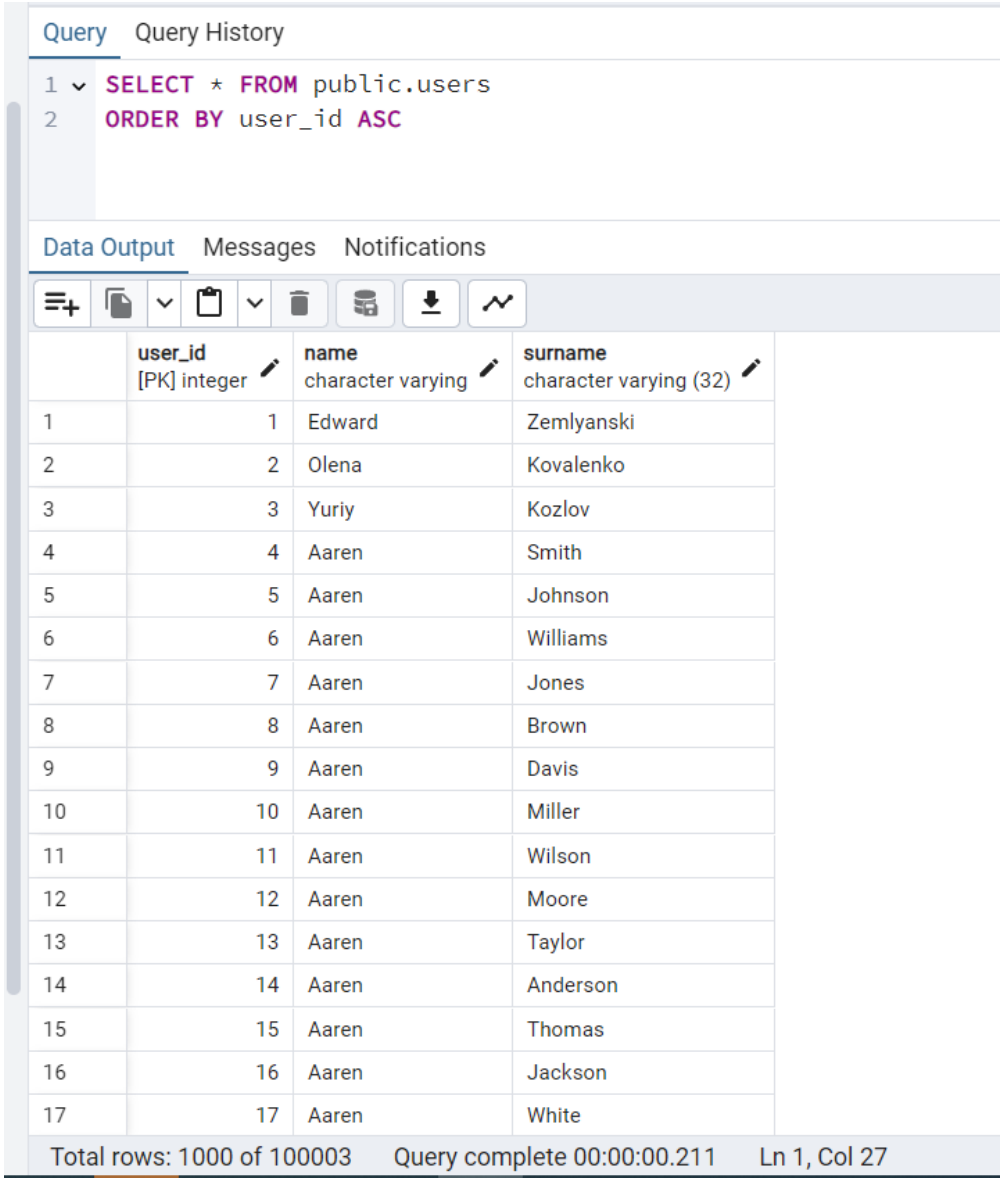
В такому випадку запис не буде створено і програма повернеться у головне меню, будучи готовою до подільшої роботи.

Генерація

Генерація 100.000 користувачів:



Програма повідомила про успішне виконання операції та час, який убло витрачено на виконання цієї операції – 1542 мілісекунди, переглянемо результати роботи:



PgAdmin повідомляє що тепер в таблиці знаходиться 100.003 записів, 3 старих записи і 100.000 нових, щойно згенерованих.

Переглянемо декілька останніх записів через консольний додаток:

```
Enter your choice: 4

Enter starting ID to show from: 99995

Enter ending currency ID to show: 100003
Function 'get_users' executed in 3.0026 milliseconds
Users:

User ID: 99995
Name: Annice
Surname: Kennedy

User ID: 99996
Name: Annice
Surname: Warren

User ID: 99997
Name: Annice
Surname: Dixon

User ID: 99998
Name: Annice
Surname: Ramos

User ID: 99999
Name: Annice
Surname: Reyes

User ID: 100000
Name: Annice
Surname: Burns

User ID: 100001
Name: Annice
Surname: Gordon

User ID: 100002
Name: Annice
Surname: Shaw

User ID: 100003
Name: Annice
Surname: Holmes
```

Приклад генерації гаманців:

```
Enter your choice: 30

Enter quantity of wallets to generate: 1000
Function 'generate_wallets' executed in 55.9628 milliseconds
```

Результати виконання операції:

Query | Query History

```
1 SELECT * FROM public.wallets
2 ORDER BY wallet_id ASC
```

Data Output | Messages | Notifications

	wallet_id [PK] integer	status boolean	user_id integer
1	1	false	1
2	2	true	2
3	3	true	1
4	4	true	54362
5	5	false	11155
6	6	true	98595
7	7	true	82127
8	8	true	89604
9	9	true	74246
10	10	true	31427
11	11	true	62302
12	12	false	64040
13	13	true	20908
14	14	true	99566
15	15	true	76182
16	16	false	41601
17	17	true	34272
18	18	false	80060

Total rows: 1000 of 1003 Query complete 00:00:00.148 Ln 2, Col 24

PgAdmin повідомляє що тепер в таблиці знаходиться 1003 записів, 3 старих записи і 1000 нових, щойно згенерованих. Ідентифікатори користувачів, на яких посилаються згенеровані записи обираютсья випадковим чином серед існуючих. Статус гаманця (закритий чи відкритий) теж обирається випадковим чином

Аналогічним чином працюють й інші функції для генерування даних: Унікальний ідентифікатор автоматично на один більше від ідентифікатору минулого запису у таблиці, зовнішні ключі обираються випадковим чином

серед існуючих, числові значення – випадкові серед певного діапазону значень, імена і фамілії комбінації із списків, в які парсяться текстові файли у потрібній кількості, зчитуючи файл не повністю, а певну, потрібну частину, назва валюти – випадкові 3 літери.

Функції, що відповідають за генерацію даних:

Генерація користувачів

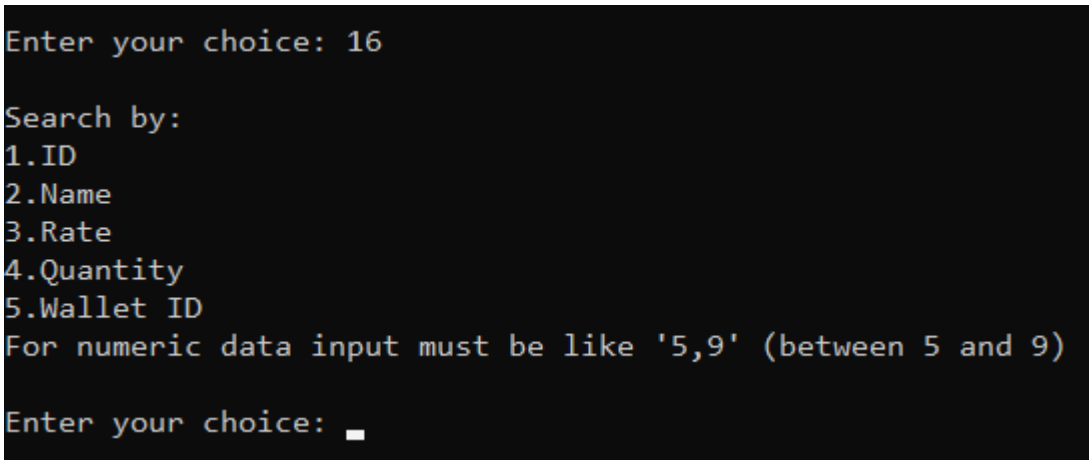
```python
    # GENERATE DATA

    @timeit
    def generate_users(self, quantity):
        # Get the current max user_id
        c = self.conn.cursor()
        c.execute("SELECT COALESCE(MAX(user_id), 0) FROM users")  # Use 0 if there are no users
        max_user_id = c.fetchone()[0]
        next_user_id = max_user_id + 1

        # Prepare name list
        name_list = ""

        with open('names.txt', 'r') as names:
            for counter in range(ceil(sqrt(quantity))):
                name = names.readline()
                if not name: break
                name_list += "'" + name.strip() + "', "

        name_list = name_list[:-2] # Deleting " ," in the end

        # Prepare surname list
        surname_list = ""
        with open('surnames.txt', 'r') as surnames:
            for counter in range(ceil(sqrt(quantity))):
                surname = surnames.readline()
                if not surname: break
                surname_list += "'" + surname.strip().capitalize() + "', "

        surname_list = surname_list[:-2]
```

```python

        # Insert users with incremented user_id and unique name-surname pairs
        try:
            c.execute(f"""WITH max_id AS (
                        SELECT COALESCE(MAX(user_id), 0) AS current_max_id FROM users),
                        generated_users AS (
                        SELECT
                        (ROW_NUMBER() OVER () + (SELECT current_max_id FROM max_id)) AS user_id,
                        first_name AS name,
                        last_name AS surname
                        FROM unnest(array[{name_list}]) AS first_name
                        CROSS JOIN unnest(array[{surname_list}]) AS last_name
                        LIMIT {quantity})
                        INSERT INTO users (user_id, name, surname)
                        SELECT user_id, name, surname
                        FROM generated_users;
                        """)
            self.conn.commit()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
```

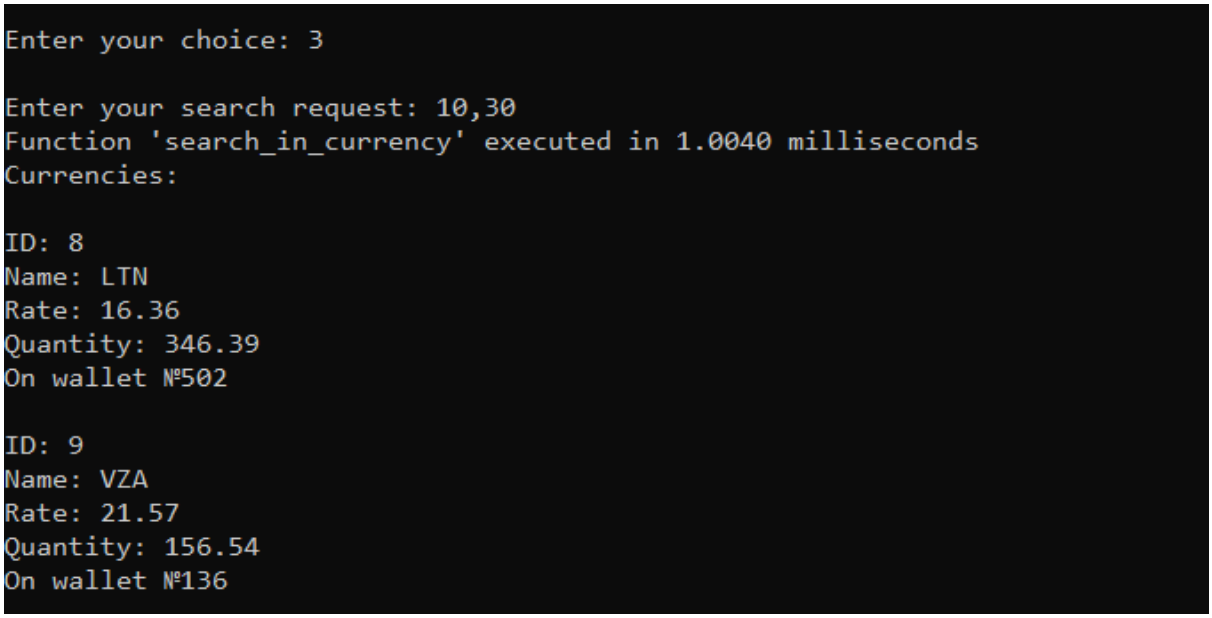
Пошук

Пошук серед даних реалізовано декількома способами для різноманіття:

Наприклад для пошуку по валютам *(записам із таблиці currency)*



Скористуємось для прикладом пошуком по значенню поля rate (курс валюти)



Отримали результати, які відповідають запиту (мають значення поля rate в діапазоні від 10 до 30, згідно із запитом користувача). Пошук по значенню поля quantity реалізовано аналогічним чином.

Пошук по полю name (назви валюти) виконується перевіркою, чи містить назва валюти запит користувача, тобто можливе часткове введення назви:

```
Enter your choice: 16

Search by:
1.ID
2.Name
3.Rate
4.Quantity
5.Wallet ID
For numeric data input must be like '5,9' (between 5 and 9)

Enter your choice: 2

Enter your search request: US
Function 'search_in_currency' executed in 1.0009 milliseconds
Currencies:

ID: 1
Name: USD
Rate: 47.86
Quantity: 123.2
On wallet №1
```

Запит користувача був «US», результати пошуку містять валюту з назвою «USD» й інші дані запису.

Пошук по декільком таблицям реалізовано наступним чином:

```
Enter your choice: 21

Enter user name (partially possible or nothing): Ed

Enter user surname (partially possible or nothing): Zem
Function 'search_users_currencies' executed in 5.0032 milliseconds

Edward Zemlyanski has 123.2 USD

Edward Zemlyanski has 0.00312 BTC
```

Під час формування відповіді на пошуковий запит серед фінансових активів *(таблиці currency)* співставляються дані з відповідними підв'язаними гаманцями *(з таблиці wallets)* і відповіді користувачі (з таблиці users) через зовнішній ключ *wallet_id*, пошук здійснюється за даними користувача *(user)*, на вибір користувача програми: за унікальним ідентифікатором *(user_id)* або за ім'ям *(name)* і фамілією *(surname)*, можливий неповний ввід даних у випадку імені та фамілії, буде здійснено пошук який містить введені дані.

Фрагменти коду:

Нижче наведені деякі приклади що ілюструють фукнціональні можливості додатку.

Перегляд даних

```
def get_currencies(self, start_id, end_id):
        c = self.conn.cursor()
        c.execute(f'SELECT * FROM currency WHERE currency_id BETWEEN
{start_id} AND {end_id} ORDER BY currency_id ASC')
        return c.fetchall()
```

Параметри: start_id і end_id — ідентифікатори, що визначають початок і кінець діапазону для вибору записів.

Запит: Вибирає всі записи з таблиці currency, де currency_id знаходиться між значеннями start_id та end_id. Записи впорядковуються за зростанням currency_id.

Результат: Повертає всі записи, що відповідають критеріям, як список кортежів.

Оновлення даних

```
def update_currency(self, name, rate, quantity, wallet_id, currency_id):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE currency SET name=\'%s\', rate=%f, quantity=%f,
wallet_id=%d WHERE currency_id=%d' % (name, rate, quantity, wallet_id,
currency_id))
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} currencies updated.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
            c.close()
```

Ця функція update_investment оновлює інформацію про інвестицію в таблиці investments на основі переданого investment_id.

Параметри:

name, seller, price, annual_income, date, wallet_id, quantity — нові значення для відповідних полів інвестиції.

investment_id — ідентифікатор інвестиції, яку потрібно оновити.

Запит: Оновлює запис у таблиці investments, встановлюючи нові значення для зазначених полів, де investment_id відповідає переданому.

Результат:

- Функція фіксує зміни в базі даних.
- Виводить кількість оновлених рядків.
- Обробляє можливі помилки і закриває курсор після завершення операції.

Створення і внесення даних

```
def add_contract(self, user_id, investment_id):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(contract_id) FROM owners_investments")
            latest_id = c.fetchone()[0]
            c.execute(f'INSERT INTO owners_investments (contract_id, user_id,
investment_id) VALUES ({latest_id + 1}, {user_id}, {investment_id})')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} contracts added.\n")
        except Error as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
        finally:
            c.close()
```

Ця функція add_contract додає новий контракт у таблицю owners_investments з переданими user_id та investment_id.

Параметри:

- user_id — ідентифікатор користувача, пов'язаного з контрактом.
- investment_id — ідентифікатор інвестиції, пов'язаної з контрактом.

Запити:

- Визначає максимальний contract_id в таблиці owners_investments.
- Вставляє новий запис з contract_id, що на 1 більший за максимальний, разом із переданими user_id та investment_id.

Результат:

- Фіксує зміни в базі даних.
- Виводить кількість доданих контрактів.
- У разі помилки виводить повідомлення про помилку та скасовує транзакцію.

Закриває курсор після завершення операції.

Пошук

```
def search_in_users(self, request, choice):
        c = self.conn.cursor()
        try:
            if choice == 1: # ID
                c.execute(f'SELECT * FROM users WHERE user_id = {request}')
            elif choice == 2: # Name
                c.execute(f'SELECT * FROM users WHERE name LIKE
\'%{request}%\' ORDER BY user_id ASC')
            elif choice == 3: # Surname
                c.execute(f'SELECT * FROM users WHERE surname LIKE
\'%{request}%\' ORDER BY user_id ASC')
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None

        c.close()
        return results
```

Функція search_in_users виконує пошук у таблиці users на основі заданого запиту request і типу пошуку choice.

Параметри:

- request — значення, яке використовується для пошуку.
- choice — визначає тип пошуку:
    - 1: Пошук за user_id.

- 2: Пошук за іменем (name), з використанням шаблону %request%.
- 3: Пошук за прізвищем (surname), також з використанням шаблону %request%.

Логіка:

- Виконує SQL-запит відповідно до вибору choice.
- Якщо пошук успішний, повертає всі результати.
- У разі помилки виводить повідомлення про помилку і скасовує транзакцію.

Результат:

- Повертає список знайдених записів або None, якщо сталася помилка.

Видалення даних

```
def delete_users_in_range(self, start_id, end_id):
        c = self.conn.cursor()
        try:
            c.execute("DELETE FROM users WHERE user_id BETWEEN %s AND %s",
(start_id, end_id))
            self.conn.commit()
            print("\nDeleted successfully! ")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
```

Функція delete_users_in_range видаляє користувачів з таблиці users, чиї user_id знаходяться в заданому діапазоні.

Параметри:

- start_id — початковий user_id для діапазону.
- end_id — кінцевий user_id для діапазону.

Запит:

- Видаляє всі записи з таблиці users, де user_id знаходиться між start_id і end_id включно.

Результат:

– Фіксує зміни в базі даних.
– Виводить повідомлення про успішне видалення.
– У разі помилки виводить повідомлення про помилку і скасовує транзакцію.

Генерація даних

```
def generate_users(self, quantity):
        # Get the current max user_id
        c = self.conn.cursor()
        c.execute("SELECT COALESCE(MAX(user_id), 0) FROM users")  # Use 0 if
there are no users
        max_user_id = c.fetchone()[0]
        next_user_id = max_user_id + 1

        # Prepare name list
        name_list = ""

        with open('names.txt', 'r') as names:
            for counter in range(ceil(sqrt(quantity))):
                name = names.readline()
                if not name: break
                name_list += "'" + name.strip() + "', "

        name_list = name_list[:-2] # Deleting " ," in the end

        # Prepare surname list
        surname_list = ""
        with open('surnames.txt', 'r') as surnames:
            for counter in range(ceil(sqrt(quantity))):
                surname = surnames.readline()
                if not surname: break
                surname_list += "'" + surname.strip().capitalize() + "', "

        surname_list = surname_list[:-2]

        # Insert users with incremented user_id and unique name-surname pairs
        try:
            c.execute(f"""WITH max_id AS (
                      SELECT COALESCE(MAX(user_id), 0) AS current_max_id FROM
users),
                      generated_users AS (
                      SELECT
                      (ROW_NUMBER() OVER () + (SELECT current_max_id FROM
max_id)) AS user_id,
                      first_name AS name,
                      last_name AS surname
                      FROM unnest(array[{name_list}]) AS first_name
                      CROSS JOIN unnest(array[{surname_list}]) AS last_name
                      LIMIT {quantity})
                      INSERT INTO users (user_id, name, surname)
```

```
                SELECT user_id, name, surname
                FROM generated_users;
                """)
            self.conn.commit()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
```

Функція generate_users генерує нових користувачів і додає їх до таблиці users, використовуючи комбінації імен та прізвищ із зовнішніх файлів.

Отримання максимального user_id:

- Виконує запит, щоб отримати максимальний user_id із таблиці users. Якщо таблиця порожня, повертає 0.
- Визначає наступний user_id, який буде використаний для нових записів.

Завантаження імен та прізвищ:

- Читає імена з файлу names.txt і прізвища з surnames.txt.
- Створює списки імен та прізвищ, обмежуючи кількість елементів до квадратного кореня від quantity (щоб забезпечити достатню кількість комбінацій).

Генерація користувачів:

- Використовує CTE (WITH запит) для створення нових записів користувачів із комбінацій імен та прізвищ.
- Використовує ROW_NUMBER() для автоматичного генерування user_id, починаючи з максимального поточного значення.

Додавання користувачів:

- Вставляє згенеровані дані в таблицю users.
- Фіксує зміни в базі даних.

Обробка помилок:

- У разі помилки транзакція скасовується, і виводиться повідомлення про помилку.

Повний код програми

main.py

```
from controller import Controller

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

controller.py

```
from model import Model
from view import View

class Controller:

    def __init__(self):
        self.model = Model()
        self.view = View()

    def run(self):
        while True:
            choice = self.show_menu()

            if choice == 1:
                self.show_currencies()
            elif choice == 2:
                self.show_investments()
            elif choice == 3:
                self.show_owners_investments()
            elif choice == 4:
                self.show_users()
            elif choice == 5:
                self.show_wallets()

            elif choice == 6:
                self.update_currency()
            elif choice == 7:
                self.update_investment()
            elif choice == 8:
                self.update_contract()
            elif choice == 8:
                self.update_contract()
            elif choice == 9:
                self.update_user()

            elif choice == 11:
                self.add_currency()
            elif choice == 12:
                self.add_investment()
            elif choice == 13:
                self.add_contract()
```

```
            elif choice == 14:
                self.add_user()
            elif choice == 15:
                self.add_wallet()

            elif choice == 16:
                self.search_in_currency()
            elif choice == 17:
                self.search_in_investments()
            elif choice == 18:
                self.search_in_contracts()
            elif choice == 19:
                self.search_in_users()
            elif choice == 20:
                self.search_in_wallets()
            elif choice == 21:
                self.search_users_currencies()

            elif choice == 22:
                self.delete_currency()
            elif choice == 23:
                self.delete_investment()
            elif choice == 24:
                self.delete_contract()
            elif choice == 25:
                self.delete_users()
            elif choice == 26:
                self.delete_wallet()

            elif choice == 27:
                self.generate_currency()
            elif choice == 28:
                self.generate_contracts()
            elif choice == 29:
                self.generate_users()
            elif choice == 30:
                self.generate_wallets()

            elif choice == 31:
                break

    def show_menu(self):

        while True:

            self.view.show_message("\nMenu:")
            self.view.show_message("\n--- GET ---\n")
            self.view.show_message("1.Show currencies")
            self.view.show_message("2.Show investments")
            self.view.show_message("3.Show contracts")
            self.view.show_message("4.Show users")
            self.view.show_message("5.Show wallets")
            self.view.show_message("\n--- UPDATE ---\n")
            self.view.show_message("6.Update currency data")
```

```
            self.view.show_message("7.Update investments")
            self.view.show_message("8.Update contracts")
            self.view.show_message("9.Update users")
            self.view.show_message("10.Update wallets")
            self.view.show_message("\n--- ADD ---\n")
            self.view.show_message("11.Add currency")
            self.view.show_message("12.Add investment")
            self.view.show_message("13.Add contract")
            self.view.show_message("14.Add user")
            self.view.show_message("15.Add wallet")
            self.view.show_message("\n--- SEARCH IN ---\n")
            self.view.show_message("16.Search in currencies")
            self.view.show_message("17.Search in investments")
            self.view.show_message("18.Search in contracts")
            self.view.show_message("19.Search in users")
            self.view.show_message("20.Search in wallets")
            self.view.show_message("21.Search currencies by user")
            self.view.show_message("\n--- DELETE ---\n")
            self.view.show_message("22.Delete currency")
            self.view.show_message("23.Delete investment")
            self.view.show_message("24.Delete contract")
            self.view.show_message("25.Delete users")
            self.view.show_message("26.Delete wallet")
            self.view.show_message("\n--- GENERATE ---\n")
            self.view.show_message("27.Generate currencies")
            self.view.show_message("28.Generate contracts")
            self.view.show_message("29.Generate users")
            self.view.show_message("30.Generate wallets")

            self.view.show_message("\n31.Quit")

            try:
                choice = self.view.get_input("\nEnter your choice: ")
            except Exception as e:
                print("\nWrong input!\n")
                continue

            return choice

    # GET (SHOW)

    def show_currencies(self):
        try:
            start_id = self.view.get_input("\nEnter starting ID to show from:
")
            end_id = self.view.get_input("\nEnter ending currency ID to show:
")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
```

```
            currencies = self.model.get_currencies(start_id, end_id)
            self.view.show_currencies(currencies)

    def show_investments(self):
        try:
            start_id = self.view.get_input("\nEnter starting ID to show from:
")
            end_id = self.view.get_input("\nEnter ending currency ID to show:
")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            investments = self.model.get_investments(start_id, end_id)
            self.view.show_investments(investments)

    def show_owners_investments(self):
        try:
            start_id = self.view.get_input("\nEnter starting ID to show from:
")
            end_id = self.view.get_input("\nEnter ending currency ID to show:
")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            owners_investments = self.model.get_owners_investments(start_id,
end_id)
            self.view.show_owners_investments(owners_investments)

    def show_users(self):
        try:
            start_id = self.view.get_input("\nEnter starting ID to show from:
")
            end_id = self.view.get_input("\nEnter ending currency ID to show:
")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            users = self.model.get_users(start_id, end_id)
            self.view.show_users(users)

    def show_wallets(self):
        try:
```

```
            start_id = self.view.get_input("\nEnter starting ID to show from:
")
            end_id = self.view.get_input("\nEnter ending currency ID to show:
")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            wallets = self.model.get_wallets(start_id, end_id)
            self.view.show_wallets(wallets)

    # UPDATE

    def update_currency(self):
        currency_id = self.view.get_currency_id()
        if currency_id != None:
            name, rate, quantity, wallet_id = self.view.get_currency_input()
            if wallet_id != None:
                self.model.update_currency(name, rate, quantity, wallet_id,
currency_id)

    def update_investment(self):
        investment_id = self.view.get_investment_id()
        if investment_id != None:
            name, seller, price, annual_income, date, wallet_id, quantity =
self.view.get_investment_input()
            if wallet_id != None:
                self.model.update_investment(
                    name,
                    seller,
                    price,
                    annual_income,
                    date,
                    wallet_id,
                    quantity,
                    investment_id
                )

    def update_contract(self):
        contract_id = self.view.get_contract_id()
        if contract_id != None:
            user_id, investment_id = self.view.get_contract_input()
            if user_id != None:
                self.model.update_contract(contract_id, user_id,
investment_id)

    def update_user(self):
        user_id = self.view.get_user_id()
        if user_id != None:
            name, surname = self.view.get_user_input()
            if name != None:
```

```
                self.model.update_user(name, surname, user_id)

    def update_wallet(self):
        wallet_id = self.view.get_wallet_id()
        if wallet_id != None:
            status, user_id = self.view.get_wallet_input()
            if user_id != None:
                self.model.update_wallet(status, user_id, wallet_id)

    # ADD (CREATE)

    def add_currency(self):
        name, rate, quantity, wallet_id = self.view.get_currency_input()
        if name != None:
            self.model.add_currency(name, rate, quantity, wallet_id)

    def add_investment(self):
        name, seller, price, annual_income, date, wallet_id, quantity =
self.view.get_investment_input()
        if wallet_id != None:
            self.model.add_investment(name, seller, price, annual_income,
date, wallet_id, quantity)

    def add_contract(self):
        user_id, investment_id = self.view.get_contract_input()
        if user_id != None:
            self.model.add_contract(user_id, investment_id)

    def add_user(self):
        name, surname = self.view.get_user_input()
        if name != None:
            self.model.add_user(name, surname)

    def add_wallet(self):
        status, user_id = self.view.get_wallet_input()
        if user_id != None:
            self.model.add_wallet(status, user_id)

    # SEARCH

    def search_in_currency(self):
        choice = self.view.get_input("\nSearch
by:\n1.ID\n2.Name\n3.Rate\n4.Quantity\n5.Wallet ID\nFor numeric data input
must be like '5,9' (between 5 and 9)\n\nEnter your choice: ")
        request = self.view.get_request("\nEnter your search request: ")
        results = self.model.search_in_currency(request, choice)
        self.view.show_currencies(results)

    def search_in_investments(self):
        choice = self.view.get_input("\nSearch
by:\n1.ID\n2.Name\n3.Seller\n4.Wallet ID\n5.Price'\n6.Annual
income\n7.Quantity\nFor numeric data input must be like '5,9' (between 5 and
9)\n\nEnter your choice: ")
        request = self.view.get_request("\nEnter your search request: ")
```

```
        results = self.model.search_in_investments(request, choice)
        self.view.show_investments(results)

    def search_in_contracts(self):
        choice = self.view.get_input("\nSearch by:\n1.Contract ID\n2.User
ID\n3.Investment ID\n\nEnter your choice: ")
        request = self.view.get_request("\nEnter your search request: ")
        results = self.model.search_in_contracts(request, choice)
        self.view.show_owners_investments(results)

    def search_in_users(self):
        choice = self.view.get_input("\nSearch by:\n1.User
ID\n2.Name\n3.Surname\n\nEnter your choice: ")
        request = self.view.get_request("\nEnter your search request: ")
        results = self.model.search_in_users(request, choice)
        self.view.show_users(results)

    def search_in_wallets(self):
        choice = self.view.get_input("\nSearch by:\n1.Wallet
ID\n2.Status\n3.Owner ID\n\nEnter your choice: ")
        request = self.view.get_request("\nEnter your search request: ")
        order = self.view.get_request("\nEnter field to order by (wallet_id,
status, user_id): ")
        results = self.model.search_in_wallets(request, choice, order)
        self.view.show_wallets(results)

    def search_users_currencies(self):
        name, surname = self.view.get_users_currencies()
        if name != None:
            result = self.model.search_users_currencies(name, surname)
            self.view.show_users_currencies(result)

    # DELETE

    def delete_currency(self):
        try:
            start_id = self.view.get_input("\nEnter starting currency ID to
remove: ")
            end_id = self.view.get_input("\nEnter ending currency ID to
remove: ")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            self.model.delete_currency(start_id, end_id)

    def delete_investment(self):
        try:
            start_id = self.view.get_input("\nEnter starting currency ID to
remove: ")
```

```
            end_id = self.view.get_input("\nEnter ending currency ID to
remove: ")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            self.model.delete_investment(start_id, end_id)

    def delete_contract(self):
        try:
            start_id = self.view.get_input("\nEnter starting currency ID to
remove: ")
            end_id = self.view.get_input("\nEnter ending currency ID to
remove: ")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            self.model.delete_contract(start_id, end_id)

    def delete_users(self):
        try:
            start_id = self.view.get_input("\nEnter starting currency ID to
remove: ")
            end_id = self.view.get_input("\nEnter ending currency ID to
remove: ")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
            self.model.delete_users_in_range(start_id, end_id)

    def delete_wallet(self):
        try:
            start_id = self.view.get_input("\nEnter starting currency ID to
remove: ")
            end_id = self.view.get_input("\nEnter ending currency ID to
remove: ")

        except Exception as e:
            print(f"Error occured: {e}")
            start_id = None
            end_id = None

        if start_id != None:
```

```
            self.model.delete_wallet(start_id, end_id)

    # Generate users

    def generate_users(self):
        quantity = self.view.get_input("\nEnter quantity of users to generate:
")
        self.model.generate_users(quantity)

    def generate_currency(self):
        quantity = self.view.get_input("\nEnter quantity of currencies to
generate: ")
        self.model.generate_currency(quantity)

    def generate_wallets(self):
        quantity = self.view.get_input("\nEnter quantity of wallets to
generate: ")
        self.model.generate_wallets(quantity)

    def generate_contracts(self):
        quantity = self.view.get_input("\nEnter quantity of contracts to
generate: ")
        self.model.generate_contracts(quantity)
```

model.py

```
import psycopg2, random
from math import ceil, sqrt
from psycopg2 import Error

import time
from functools import wraps

def timeit(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()  # Record the start time
        result = func(*args, **kwargs)  # Call the original function
        end_time = time.time()  # Record the end time
        elapsed_time = (end_time - start_time) * 1000 # Calculate elapsed time
in milliseconds

        # Output the result and elapsed time
        print(f"Function '{func.__name__}' executed in {elapsed_time:.4f}
milliseconds")

        return result  # Return the result of the function
    return wrapper

class Model:

    def __init__(self):
        self.conn = psycopg2.connect(
```

```
            dbname =,
            user =,
            password = #',
            host =,
            port =
        )

    #  GET

    @timeit
    def get_currencies(self, start_id, end_id):
        c = self.conn.cursor()
        c.execute(f'SELECT * FROM currency WHERE currency_id BETWEEN
{start_id} AND {end_id} ORDER BY currency_id ASC')
        return c.fetchall()

    @timeit
    def get_investments(self, start_id, end_id):
        c = self.conn.cursor()
        c.execute(f'SELECT * FROM investments WHERE investment_id BETWEEN
{start_id} AND {end_id} ORDER BY investment_id ASC')
        return c.fetchall()

    @timeit
    def get_owners_investments(self, start_id, end_id):
        c = self.conn.cursor()
        c.execute(f'SELECT * FROM owners_investments WHERE contract_id BETWEEN
{start_id} AND {end_id} ORDER BY contract_id ASC')
        return c.fetchall()

    @timeit
    def get_users(self, start_id, end_id):
        c = self.conn.cursor()
        c.execute(f'SELECT * FROM users WHERE user_id BETWEEN {start_id} AND
{end_id} ORDER BY user_id ASC')
        return c.fetchall()

    @timeit
    def get_wallets(self, start_id, end_id):
        c = self.conn.cursor()
        c.execute(f'SELECT * FROM wallets WHERE wallet_id BETWEEN {start_id}
AND {end_id} ORDER BY wallet_id ASC')
        return c.fetchall()

    # UPDATE

    @timeit
    def update_currency(self, name, rate, quantity, wallet_id, currency_id):
        c = self.conn.cursor()
        try:
            c.execute('UPDATE currency SET name=\'%s\', rate=%f, quantity=%f,
wallet_id=%d WHERE currency_id=%d' % (name, rate, quantity, wallet_id,
currency_id))
            self.conn.commit()
```

```
            rows_updated = c.rowcount
            print(f"\n{rows_updated} currencies updated.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
            c.close()

    @timeit
    def update_investment(self, name, seller, price, annual_income, date,
wallet_id, quantity, investment_id):
        c = self.conn.cursor()
        try:
            c.execute(f'UPDATE investments SET name=\'{name}\',
seller=\'{seller}\', price={price}, annual_income={annual_income},
date=\'{date}\', wallet_id={wallet_id}, quantity={quantity} WHERE
investment_id={investment_id}')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} investments updated.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
            c.close()

    @timeit
    def update_contract(self, contract_id, user_id, investment_id):
        c = self.conn.cursor()
        try:
            c.execute(f'UPDATE owners_investments SET user_id={user_id},
investment_id={investment_id} WHERE contract_id={contract_id}')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} relations updated.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
            c.close()

    @timeit
    def update_user(self, name, surname, user_id):
        c = self.conn.cursor()
        try:
            c.execute(f'UPDATE users SET name=\'{name}\',
surname=\'{surname}\' WHERE user_id={user_id}')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} users updated.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
```

```
            c.close()

    @timeit
    def update_wallet(self, status, user_id, wallet_id):
        c = self.conn.cursor()
        try:
            c.execute(f'UPDATE wallets SET status={status}, user_id={user_id}
WHERE wallet_id={wallet_id}')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} wallets updated.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
            c.close()

    # ADD

    @timeit
    def add_currency(self, name, rate, quantity, wallet_id):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(currency_id) FROM currency")
            latest_id = c.fetchone()[0]
            c.execute(f'INSERT INTO currency (currency_id, name, rate,
quantity, wallet_id) VALUES ({latest_id + 1}, \'{name}\', {rate}, {quantity},
{wallet_id})')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} currency added.\n")
        except Error as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
        finally:
            c.close()

    @timeit
    def add_investment(self, name, seller, price, annual_income, date,
wallet_id, quantity):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(investment_id) FROM investments")
            latest_id = c.fetchone()[0]
            c.execute(f'INSERT INTO investments (investment_id, name, seller,
price, annual_income, date, wallet_id, quantity) VALUES ({latest_id + 1},
\'{name}\', \'{seller}\', {price}, {annual_income}, \'{date}\', {wallet_id},
{quantity})')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} investments added.\n")
        except Error as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
```

```
        finally:
            c.close()

    @timeit
    def add_contract(self, user_id, investment_id):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(contract_id) FROM owners_investments")
            latest_id = c.fetchone()[0]
            c.execute(f'INSERT INTO owners_investments (contract_id, user_id,
investment_id) VALUES ({latest_id + 1}, {user_id}, {investment_id})')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} contracts added.\n")
        except Error as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
        finally:
            c.close()

    @timeit
    def add_user(self, name, surname):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(user_id) FROM users")
            latest_id = c.fetchone()[0]
            c.execute(f'INSERT INTO users (user_id, name, surname) VALUES
({latest_id + 1}, \'{name}\', \'{surname}\')')
            self.conn.commit()
            rows_updated = c.rowcount
            print(f"\n{rows_updated} users added.\n")
        except Error as e:
            self.conn.rollback()
            print(f"An error occurred: {e}")
        finally:
            c.close()

    @timeit
    def add_wallet(self, status, user_id):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(wallet_id) FROM wallets")
            latest_id = c.fetchone()[0]
            c.execute(f'INSERT INTO wallets (wallet_id, status, user_id)
VALUES ({latest_id + 1}, {status}, {user_id})')
            self.conn.commit()
            rows_updated = c.rowcount
            self.conn.rollback()
            print(f"\n{rows_updated} wallets added.\n")
        except Error as e:
            print(f"An error occurred: {e}")

        c.close()
```

```
    # SEARCH

    @timeit
    def search_in_currency(self, request, choice):
        c = self.conn.cursor()
        try:
            if choice == 1: # id
                c.execute(f'SELECT * FROM currency WHERE currency_id =
{request}')
            elif choice == 2: # name
                c.execute(f'SELECT * FROM currency WHERE name LIKE
\'%{request}%\' ORDER BY currency_id ASC')
            elif choice == 3: # rate
                values = [item.strip() for item in request.split(',')]
                c.execute(f'SELECT * FROM currency WHERE rate BETWEEN
{values[0]} AND {values[1]} ORDER BY currency_id ASC')
            elif choice == 4: # quantity
                values = [item.strip() for item in request.split(',')]
                c.execute(f'SELECT * FROM currency WHERE quantity BETWEEN
{values[0]} AND {values[1]} ORDER BY currency_id ASC')
            if choice == 5: # wallet id
                c.execute(f'SELECT * FROM currency WHERE wallet_id = {request}
ORDER BY currency_id ASC')
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None

        c.close()
        return results

    @timeit
    def search_in_investments(self, request, choice):
        c = self.conn.cursor()
        try:
            if choice == 1: # ID
                c.execute(f'SELECT * FROM investments WHERE investment_id =
{request}')
            elif choice == 2: # Name
                c.execute(f'SELECT * FROM investments WHERE name LIKE
\'%{request}%\' ORDER BY investment_id ASC')
            elif choice == 3: # Seller
                c.execute(f'SELECT * FROM investments WHERE seller LIKE
\'%{request}%\' ORDER BY investment_id ASC')
            elif choice == 4: # Wallet ID
                c.execute(f'SELECT * FROM investments WHERE wallet_id =
{request} ORDER BY investment_id ASC')
            elif choice >= 5 and choice <= 7:
                values = [item.strip() for item in request.split(',')]
                if choice == 5: # price
                    c.execute(f'SELECT * FROM investments WHERE price BETWEEN
{values[0]} AND {values[1]} ORDER BY investment_id ASC')
                elif choice == 6: # annual income
```

```
                    c.execute(f'SELECT * FROM investments WHERE annual_income
BETWEEN {values[0]} AND {values[1]} ORDER BY investment_id ASC')
                elif choice == 7: # quantity
                    c.execute(f'SELECT * FROM investments WHERE quantity
BETWEEN {values[0]} AND {values[1]} ORDER BY investment_id ASC')
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None

        c.close()
        return results

    @timeit
    def search_in_contracts(self, request, choice):
        c = self.conn.cursor()
        try:
            if choice == 1: # ID
                c.execute(f'SELECT * FROM owners_investments WHERE contract_id
= {request}')
            elif choice == 2: # user id
                c.execute(f'SELECT * FROM owners_investments WHERE user_id =
{request} ORDER BY contract_id ASC')
            elif choice == 3: # investment id
                c.execute(f'SELECT * FROM owners_investments WHERE
investment_id = {request} ORDER BY contract_id ASC')
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None

        c.close()
        return results

    @timeit
    def search_in_users(self, request, choice):
        c = self.conn.cursor()
        try:
            if choice == 1: # ID
                c.execute(f'SELECT * FROM users WHERE user_id = {request}')
            elif choice == 2: # Name
                c.execute(f'SELECT * FROM users WHERE name LIKE
\'%{request}%\' ORDER BY user_id ASC')
            elif choice == 3: # Surname
                c.execute(f'SELECT * FROM users WHERE surname LIKE
\'%{request}%\' ORDER BY user_id ASC')
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None
```

```
        c.close()
        return results

    @timeit
    def search_in_wallets(self, request, choice, order):
        c = self.conn.cursor()
        try:
            if choice == 1: # ID
                c.execute(f'SELECT * FROM wallets WHERE wallet_id =
{request}')
            elif choice == 2: # status
                if request.lower() in ['opened', 'open', '1', 'yes', 'active',
'on', 'true']:
                    request = 'true'
                elif request.lower() in ['closed', 'close', '0', 'no', 'off',
'false']:
                    request = 'false'
                c.execute(f'SELECT * FROM wallets WHERE status = {request}
ORDER BY {order} ASC')
            elif choice == 3: # user_id
                c.execute(f'SELECT * FROM wallets WHERE user_id = {request}
ORDER BY {order} ASC')
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None
        c.close()
        return results

    # SEARCH IN users and users at the same time

    @timeit
    def search_users_currencies(self, name, surname):
        c = self.conn.cursor()
        try:
            c.execute(f"""SELECT currency.name, currency.quantity, users.name,
users.surname
            FROM currency
            JOIN wallets ON currency.wallet_id = wallets.wallet_id
            JOIN users ON users.user_id = wallets.user_id
            WHERE users.name LIKE \'%{name}%\' AND users.surname LIKE
\'%{surname}%\';""")
            results = c.fetchall()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
            results = None
        c.close()
        return results

    # DELETE

    @timeit
```

```
    def delete_currency(self, start_id, end_id):
        try:
            c = self.conn.cursor()
            c.execute(f'DELETE FROM currency WHERE currency_id BETWEEN
{start_id} AND {end_id}')
            self.conn.commit()
            print("\nDeleted successfully! ")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    @timeit
    def delete_investment(self, start_id, end_id):
        try:
            c = self.conn.cursor()
            c.execute(f'DELETE FROM investments WHERE investment_id BETWEEN
{start_id} AND {end_id}')
            self.conn.commit()
            print("\nDeleted successfully! ")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    @timeit
    def delete_contract(self, start_id, end_id):
        try:
            c = self.conn.cursor()
            c.execute(f'DELETE FROM owners_investments WHERE contract_id
BETWEEN {start_id} AND {end_id}')
            self.conn.commit()
            print("\nDeleted successfully! ")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    @timeit
    def delete_users_in_range(self, start_id, end_id):
        c = self.conn.cursor()
        try:
            c.execute("DELETE FROM users WHERE user_id BETWEEN %s AND %s",
(start_id, end_id))
            self.conn.commit()
            print("\nDeleted successfully! ")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    @timeit
    def delete_wallet(self, start_id, end_id):
        try:
            c = self.conn.cursor()
            c.execute(f'DELETE FROM wallets WHERE wallet_id BETWEEN {start_id}
AND {end_id}')
            self.conn.commit()
```

```
            print("\nDeleted successfully! ")
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    # GENERATE DATA

    @timeit
    def generate_users(self, quantity):
        # Get the current max user_id
        c = self.conn.cursor()
        c.execute("SELECT COALESCE(MAX(user_id), 0) FROM users")  # Use 0 if
there are no users
        max_user_id = c.fetchone()[0]
        next_user_id = max_user_id + 1

        # Prepare name list
        name_list = ""

        with open('names.txt', 'r') as names:
            for counter in range(ceil(sqrt(quantity))):
                name = names.readline()
                if not name: break
                name_list += "'" + name.strip() + "', "

        name_list = name_list[:-2] # Deleting " ," in the end

        # Prepare surname list
        surname_list = ""
        with open('surnames.txt', 'r') as surnames:
            for counter in range(ceil(sqrt(quantity))):
                surname = surnames.readline()
                if not surname: break
                surname_list += "'" + surname.strip().capitalize() + "', "

        surname_list = surname_list[:-2]

        # Insert users with incremented user_id and unique name-surname pairs
        try:
            c.execute(f"""WITH max_id AS (
                      SELECT COALESCE(MAX(user_id), 0) AS current_max_id FROM
users),
                      generated_users AS (
                      SELECT
                      (ROW_NUMBER() OVER () + (SELECT current_max_id FROM
max_id)) AS user_id,
                      first_name AS name,
                      last_name AS surname
                      FROM unnest(array[{name_list}]) AS first_name
                      CROSS JOIN unnest(array[{surname_list}]) AS last_name
                      LIMIT {quantity})
                      INSERT INTO users (user_id, name, surname)
                      SELECT user_id, name, surname
                      FROM generated_users;
```

```
                    """)
            self.conn.commit()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    @timeit
    def generate_currency(self, quantity):
        c = self.conn.cursor()
        try:
            c.execute("SELECT MAX(wallet_id) FROM wallets")
            max_wallet_id = c.fetchone()[0]
            c.execute(f"""WITH max_id AS (
                    SELECT COALESCE(MAX(currency_id), 0) AS current_max_id
FROM currency
                ),
                generated_currency AS (
                    SELECT
                        (ROW_NUMBER() OVER () + (SELECT current_max_id FROM
max_id)) AS currency_id,
                        chr(trunc(65 + random() * 26)::int) ||
                        chr(trunc(65 + random() * 26)::int) ||
                        chr(trunc(65 + random() * 26)::int) AS name,
                        ROUND((random() * 100)::numeric, 2) AS rate,
                        ROUND((random() * 1000)::numeric, 2) AS quantity,
                        trunc(random() * {max_wallet_id} + 1)::int AS
wallet_id
                    FROM generate_series(1, {quantity})
                )
                INSERT INTO currency (currency_id, name, rate, quantity,
wallet_id)
                SELECT currency_id, name, rate, quantity, wallet_id
                FROM generated_currency
                LIMIT {quantity}
                """)
            self.conn.commit()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()

    @timeit
    def generate_wallets(self, quantity):
        c = self.conn.cursor()
        try:
            # Get the maximum wallet_id from the wallets table
            c.execute("SELECT COALESCE(MAX(wallet_id), 0) FROM wallets")
            max_wallet_id = c.fetchone()[0]

            # Generate wallets using a common table expression
            c.execute(f"""
                WITH max_id AS (
                    SELECT COALESCE(MAX(user_id), 0) AS current_max_id FROM
users
                ),
```

```
                generated_wallets AS (
                    SELECT
                        (ROW_NUMBER() OVER () + {max_wallet_id}) AS wallet_id,
                        (RANDOM() < 0.5) AS status,  -- Randomly generates
true or false
                        ((random()*(SELECT current_max_id FROM max_id))) AS
user_id
                    FROM generate_series(1, {quantity})
                )
                INSERT INTO wallets (wallet_id, status, user_id)
                SELECT wallet_id, status, user_id
                FROM generated_wallets
                LIMIT {quantity}
            """)
            self.conn.commit()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
        finally:
            c.close()

    @timeit
    def generate_contracts(self, quantity):
        c = self.conn.cursor()
        try:
            # Get the maximum contract_id from the contracts table
            c.execute("SELECT COALESCE(MAX(contract_id), 0) FROM
owners_investments")
            max_contract_id = c.fetchone()[0]

            # Get the maximum user_id from the users table
            c.execute("SELECT COALESCE(MAX(user_id), 0) FROM users")
            max_user_id = c.fetchone()[0]

            # Get the maximum investment_id from the investments table
            c.execute("SELECT COALESCE(MAX(investment_id), 0) FROM
investments")
            max_investment_id = c.fetchone()[0]

            # Generate contracts using a common table expression
            c.execute(f"""
                WITH generated_contracts AS (
                    SELECT
                        (ROW_NUMBER() OVER () + {max_contract_id}) AS
contract_id,
                        FLOOR(RANDOM() * {max_user_id} + 1)::int AS user_id,
                        FLOOR(RANDOM() * {max_investment_id} + 1)::int AS
investment_id
                    FROM generate_series(1, {quantity})
                )
                INSERT INTO owners_investments (contract_id, user_id,
investment_id)
                SELECT contract_id, user_id, investment_id
                FROM generated_contracts
```

```
                LIMIT {quantity}
            """)
            self.conn.commit()
        except Exception as e:
            print(f"An error occurred: {e}")
            self.conn.rollback()
        finally:
            c.close()  # Ensure the cursor is closed after the operation
```

view.py

```
class View:

    def show_currencies(self, currencies):
        if currencies != None:
            print("Currencies:")
            for currency in currencies:
                print(f"\nID: {currency[0]}\nName: {currency[1]}\nRate:
{currency[2]}\nQuantity: {currency[3]}\nOn wallet №{currency[4]}")

    def show_investments(self, investments):
        if investments != None:
            print("Investments:")
            for investment in investments:
                print(f"\nID: {investment[0]}\nName: {investment[1]}\nSeller:
{investment[2]}\nPrice: {investment[3]}\nAnnual income: {investment[4]}\nDate:
{investment[5]}\nBought from wallet no. {investment[6]}\nQuantity:
{investment[7]}")

    def show_owners_investments(self, owners_investments):
        if owners_investments != None:
            print("Owners and their investments:")
            for owners_investment in owners_investments:
                print(f"\nContract ID: {owners_investment[0]}\nUser with ID
{owners_investment[1]} owns investments with ID: {owners_investment[2]}")

    def show_users(self, users):
        if users != None:
            print("Users:")
            for user in users:
                print(f"\nUser ID: {user[0]}\nName: {user[1]}\nSurname:
{user[2]}")

    def show_wallets(self, wallets):
        if wallets != None:
            print("Wallets:")
            for wallet in wallets:
                print(f"\nWallet ID: {wallet[0]}\nOpen: {wallet[1]}, Owner ID:
{wallet[2]}")

    def get_currency_id(self):
        try:
            currency_id = int(input("\nEnter currency ID: "))
```

```
        except Exception as e:
            print("\nWrong input!")
            currency_id = None
        return currency_id

    def get_currency_input(self):
        try:
            name = input("\nEnter name: ")
            rate = float(input("\nEnter rate: "))
            quantity = float(input("\nEnter quantity: "))
            wallet_id = int(input("\nEnter wallet ID: "))
        except Exception as e:
            print("\nWrong input!")
            return None, None, None, None
        return name, rate, quantity, wallet_id

    def get_investment_id(self):
        try:
            investment_id = int(input("\nEnter investment ID: "))
        except Exception as e:
            print("\nWrong input!")
            investment_id = None
        return investment_id

    def get_investment_input(self):
        try:
            name = input("\nEnter name: ")
            seller = input("\nEnter seller company name: ")
            price = float(input("\nEnter investment price: "))
            annual_income = float(input("\nEnter annual income ($): "))
            date = input("\nEnter date (YEAR-MONTH-DAY
HOURS:MINUTES:SECONDS+TIME_ZONE): ")
            wallet_id = int(input("\nEnter wallet ID: "))
            quantity = int(input("\nEnter quantity: "))
        except Exception as e:
            print("\nWrong input!\n")
            return None, None, None, None, None, None, None
        return name, seller, price, annual_income, date, wallet_id, quantity

    def get_contract_id(self):
        try:
            contract_id = int(input("\nEnter contract ID: "))
        except Exception as e:
            print("\nWrong input!\n")
            contract_id = None
        return contract_id

    def get_user_id(self):
        try:
            user_id = int(input("\nEnter user ID: "))
        except Exception as e:
            print("\nWrong input!\n")
            return None
        return user_id
```

```
    def get_user_input(self):
        try:
            name = input("\nEnter name: ")
            surname = input("\nEnter surname: ")
        except Exception as e:
            print("\nWrong input!\n")
            return None, None
        return name, surname

    def get_contract_input(self):
        try:
            user_id = int(input("\nEnter user ID as a part of contract: "))
            ivestment_id = int(input("\nEnter investment ID as a part of
contract: "))
        except Exception as e:
            print("\nWrong input!\n")
            user_id = None
            ivestment_id = None
        return user_id, ivestment_id

    def get_wallet_id(self):
        try:
            wallet_id = int(input("\nEnter wallet ID: "))
        except Exception as e:
            print("\nWrong input!\n")
            wallet_id = None
        return wallet_id

    def get_wallet_input(self):
        try:
            status = input("\nOpen/Close wallet (open/yes/close/no etc...):
").lower() in ["open", "yes", "1", "true"]
            user_id = int(input("\nEnter new owner ID: "))
        except Exception as e:
            print("\nWrong input!\n")
            return None, None
        return status, user_id

    def get_users_currencies(self):
        try:
            name = input("\nEnter user name (partially possible or nothing):
")
            surname = input("\nEnter user surname (partially possible or
nothing): ")
        except Exception as e:
            print("\nWrong input!\n")
            name = None
            surname = None
        return name, surname

    def show_users_currencies(self, data):
        if data != None:
            for fetch in data:
```

```
            print(f"\n{fetch[2]} {fetch[3]} has {fetch[1]} {fetch[0]}")

    def show_message(self, message):
        print(message)

    def get_input(self, input_message):
        return int(input(input_message))

    def get_request(self, input_message):
        return input(input_message)
```