

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Розрахунково-графічна робота**

з дисципліни

**«Бази даних та засоби управління»**

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

Виконав: студент групи КВ-12

Дмитрієвцев М. В.

Telegram: @muxasuk\_demitresku

Київ – 2023

*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).

Посилання на репозиторій y github: <https://github.com/Dmitriievtsev-Mykhail/BD.git>

Графічне подання концептуальної моделі зображено на рисунку 1

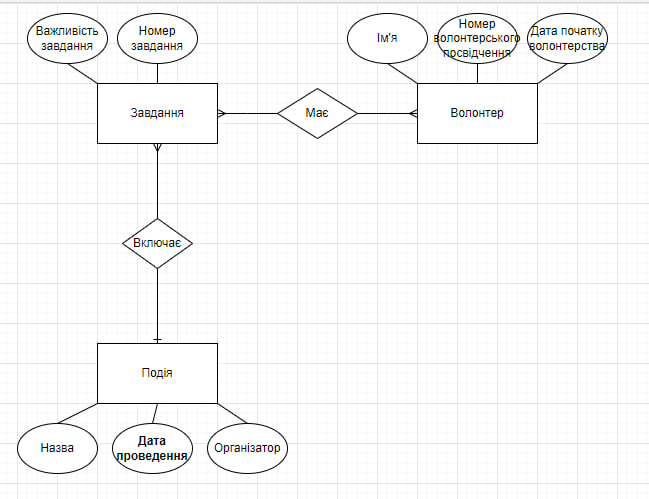
****

Рисунок 1 – ER-діаграма, побудована за нотацією «Crow`s foot»

**Сутності з описом призначення**

Предметна галузь «Система управління волонтерами на подіях» включає в себе 3 сутності, кожна сутність містить декілька атрибутів:

1. Event (event\_id, title, organizer, date\_of\_event).

2. Task (task\_id, importance, event\_id).

3. Volunteer (volunteer\_id, name, start\_date\_of\_volunteering).

Сутність Event описує події, які мають бути проведені волонтерами. Кожна подія має свій ідентифікатор, а також містить інформацію про свою назву, дату проведення та організатора.

Сутність Task описує завдання, яке потрібно виконати на певній події. Також кожне завдання містить свій ідентифікатор і певну задачу, яку повинні виконати волонтери.

Сутність Volunteer описує волонтера, який виконує певне завдання під час якоїсь волонтерської події. Кожен волонтер має свій ідентифікатор, а також ім’я та дату початку волонтерства.

**Опис зв’язків між сутностями предметної області**

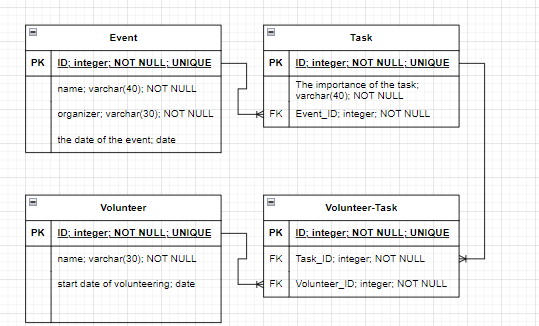
*Зв’язок між Task та Event*:

Кожна подія має завдання, яке потрібно виконати. Зв'язок 1:N – до однієї події може належати багато різних завдань, а певне завдання може відноситися лише до однієї події.

*Зв'язок між Task та Volunteer*:

Кожне завдання має волонтерів, які виконують їх. Зв'язок M:N – кожен волонтер може виконувати декілька завдань на певній події, також одне завдання можуть виконувати кілька волонтерів.

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

Рисунок 2 – Схема бази даних у графічному вигляді

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.10.

Середовище розробки програмного забезпечення – PyCharm Community Edition.

Бібліотека взаємодії з PostgreSQL - psycopg2 (рисунок 3 та 4)

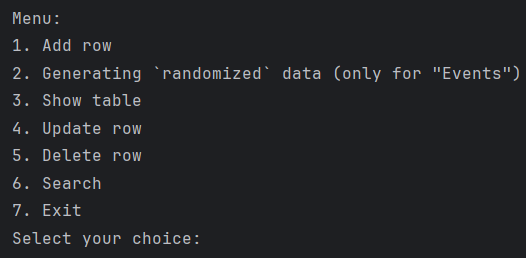


Рисунок 3 – Структура меню користувача

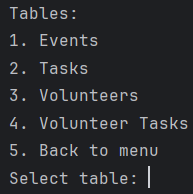


Рисунок 4 – Елементи таблиці, з якими можна працювати

Меню складається з 7 пунктів:

1. **Add row** служить для додавання рядка в таблицю. Після вибору цього пункту, потрібно обрати таблицю, для якої буде виконана ця операція, після чого, треба ввести дані для кожного атрибуту таблиці, щоб додати новий рядок.
2. **Generating `randomized` data**. Для цього пункту було обрано таблицю Events. Цей пункт створений для додавання «рандомізованих» даних. Потрібно ввести число полів, яке ми хочемо додати.
3. **Show table** служить для показу таблиці. Перед виведенням, користувач обирає, яку саме таблицю потрібно вивести. Після цього на екрані виводяться всі поля обраної таблиці БД.
4. **Update row** використовується для редагування полів по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися, після чого потрібно ввести id поля, яке потрібно змінити. Залишається ввести нові дані для кожного атрибуту таблиці.
5. **Delete row** служить для видалення рядку по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення рядка, після чого користувач вводить id рядка, який потрібно видалити.
6. **Search** створений для пошуку за атрибутами з декількох таблиць. Пропонується 4 варіанти вибору (рисунок 5):

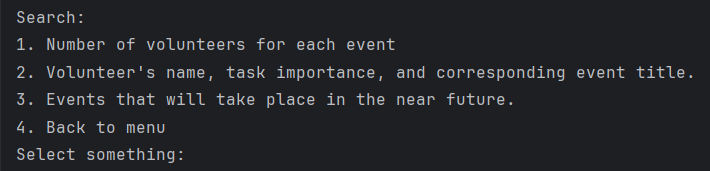


Рисунок 5 – Варіанти запитів

1. Number of volunteers for each event.

Запит для отримання загальної кількості волонтерів для кожної події. На виході отримуємо таблицю із id події, її назви та кількість волонтерів, що беруть у ній участь.

1. Volunteer's name, task importance, and corresponding event title. Цей запит виведе ім'я волонтера, важливість завдання та відповідну назву події. На виході отримуємо таблицю з іменем волонтера, важливістю завдання та назвою заходу.
2. Events that will take place in the near future. Цей запит для отримання заходів, які відбудуться в найближчому майбутньому. На виході отримуємо таблицю з п’ятьма найближчими заходами. Тобто табличка, що містить назву заходу та дату його проведення.
3. Back to menu. Повернення до основного меню.

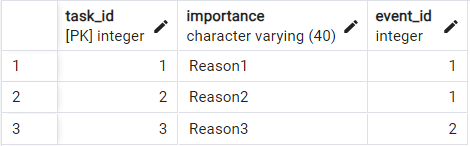
Також можна побачити час виконання запиту у мілісекундах, після виведення даних у кожному пункті.

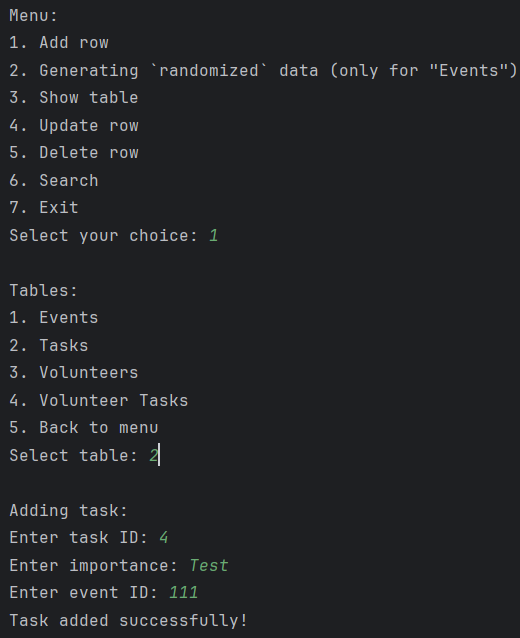
1. **Exit** служить для виходу з програми.

**Завдання 1**

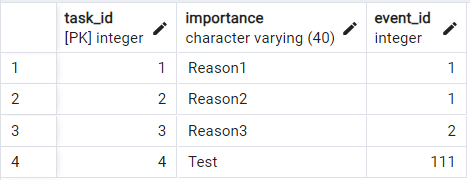
**Додавання рядка**

***Таблиця “tasks” до:***

******

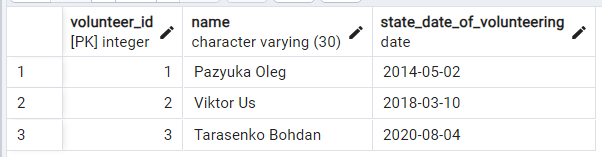
******

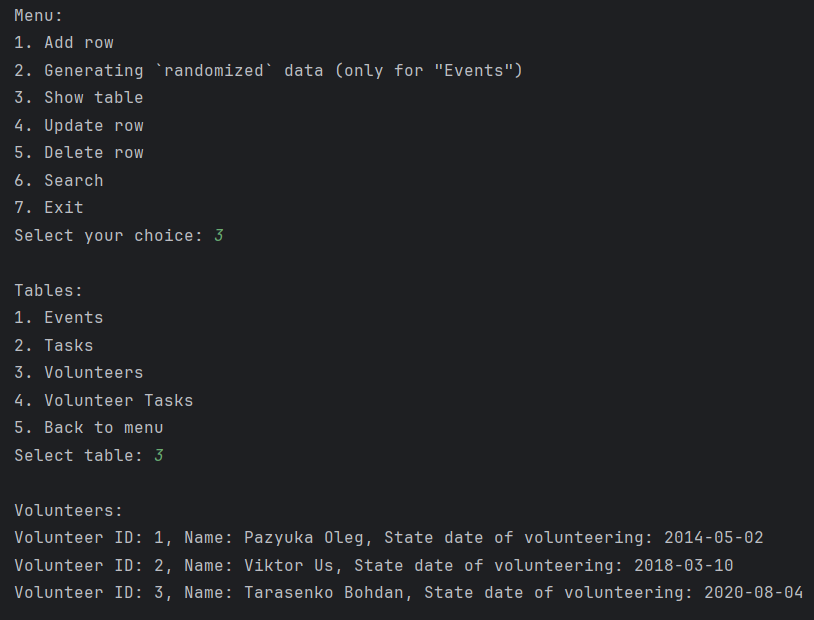
***Таблиця “tasks” після:***

******

**Перегляд таблиці**

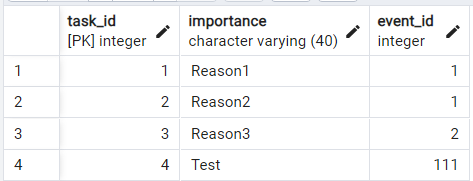
***Таблиця “volunteers”:***

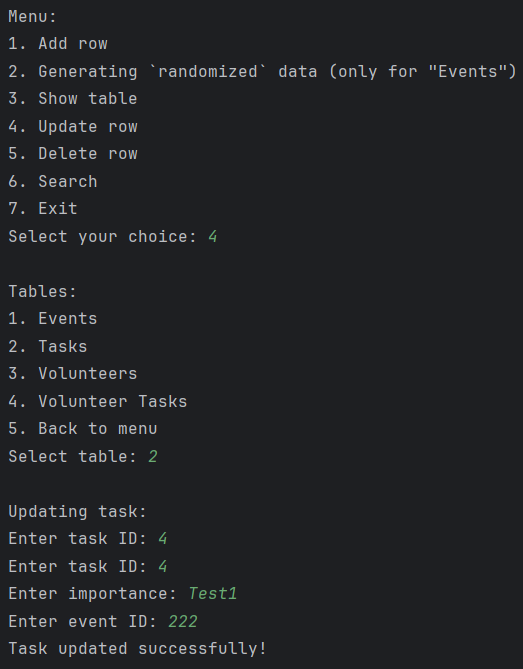
******

******

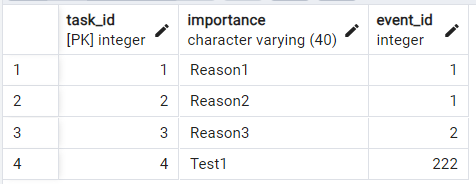
**Оновлення рядка**

***Таблиця “tasks” до:***

****

****

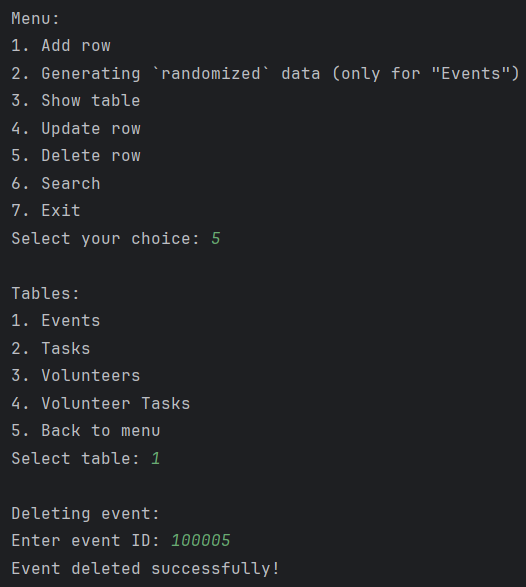
***Таблиця “tasks” після:***

****

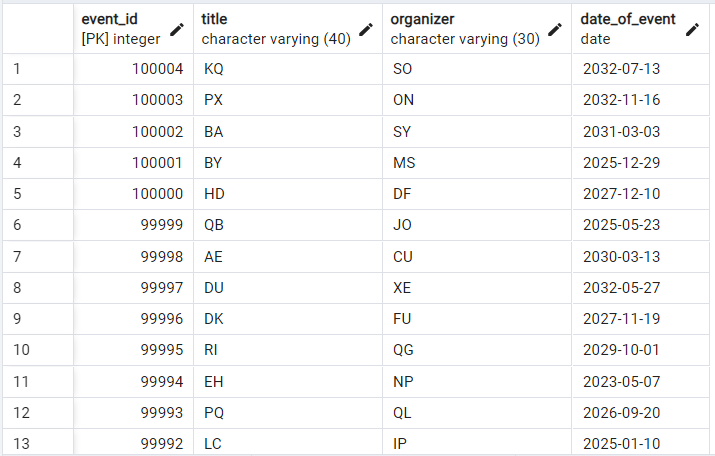
**Видалення рядка**

***Таблиця “events” до:***

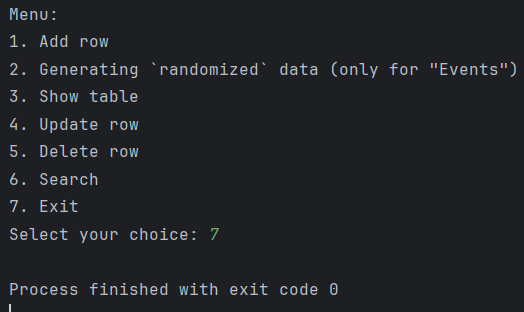
****

****

***Таблиця “events” після:***

****

**Вихід**

****

**Завдання 2**

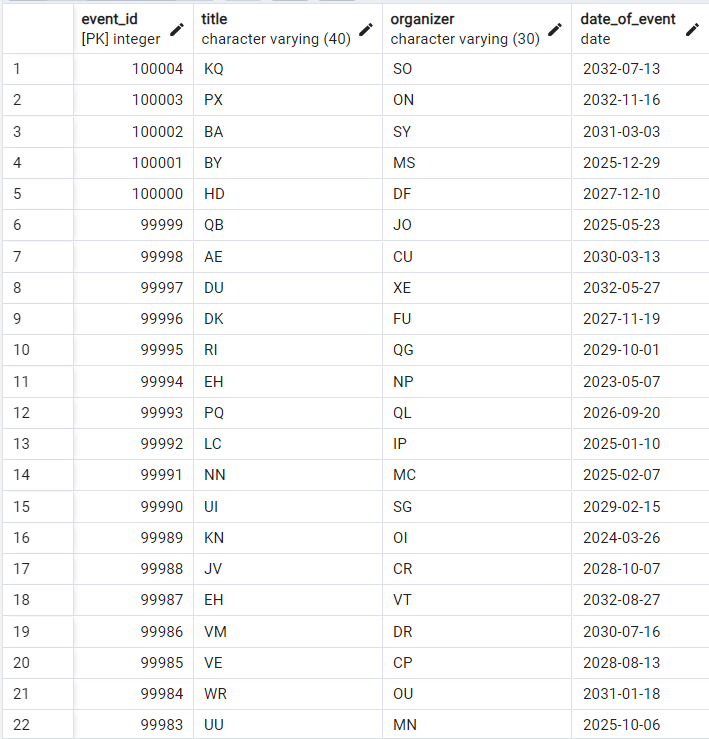
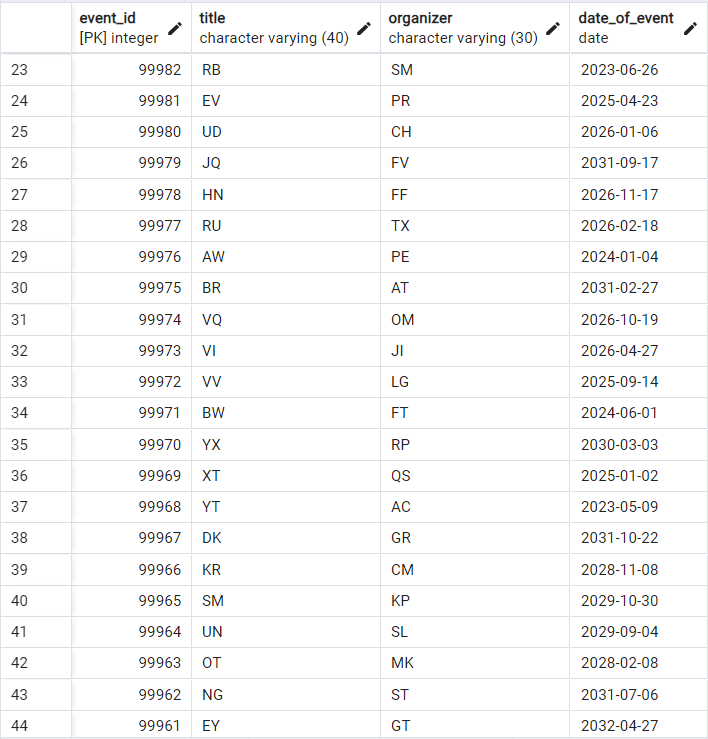
**Генерування «рандомізованих» даних**

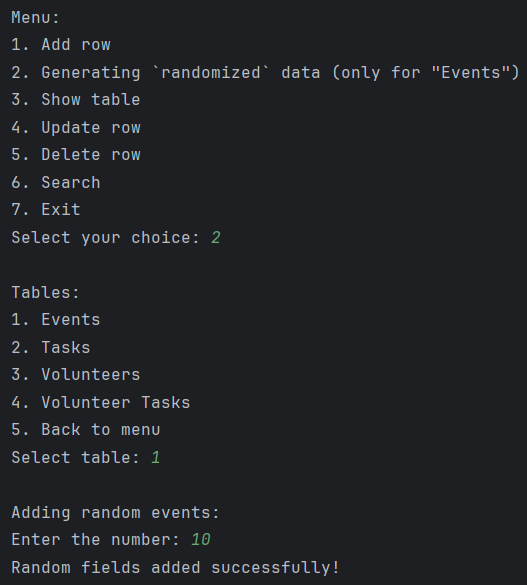
Запит, що був використаний для генерування «рандомізованих» даних:

INSERT INTO events (event\_id, title, organizer, date\_of\_event) SELECT row\_number() OVER () + (SELECT COALESCE(MAX(event\_id), 0) FROM events), chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), '2023-01-01'::date + (random() \* (365 \* 10))::integer FROM generate\_series(1, %s);

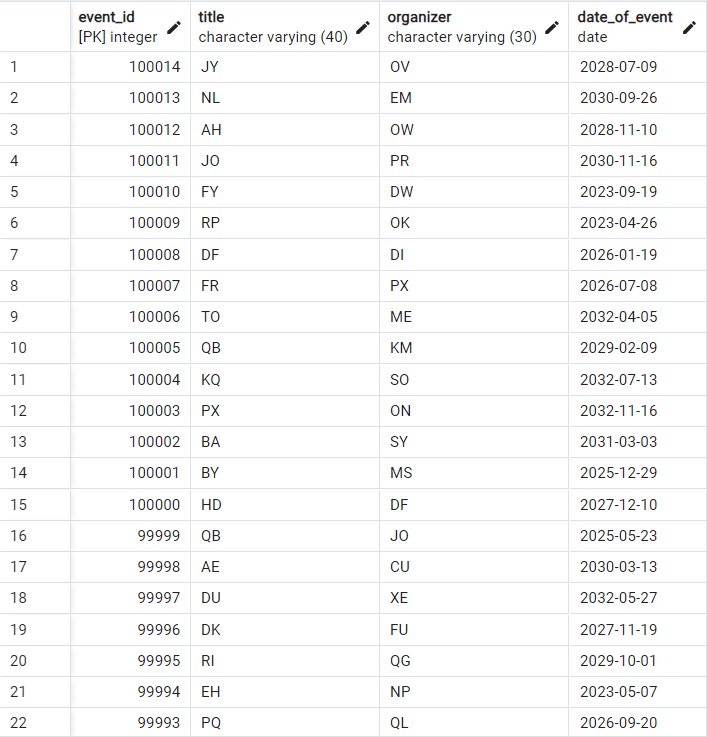
%s - підставляється число, яке вводиться користувачем, скільки рядків хоче додати користувач в таблицю.

***Таблиця “Events” до:***

*** ***

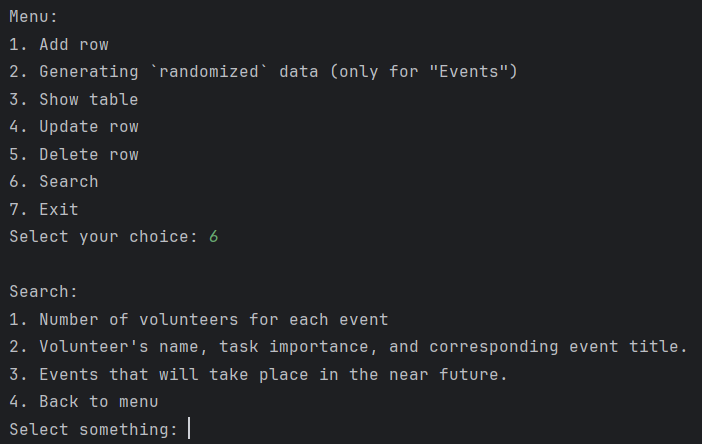
******

***Таблиця “Events” після:***

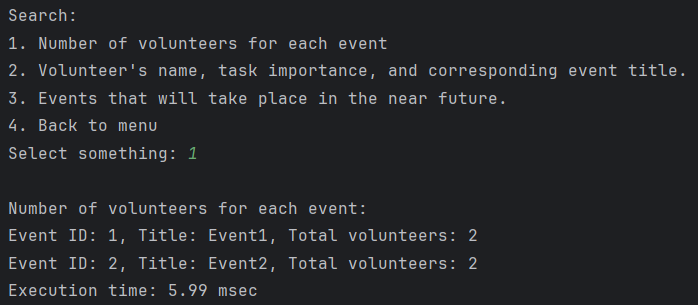
****

**Завдання 3**

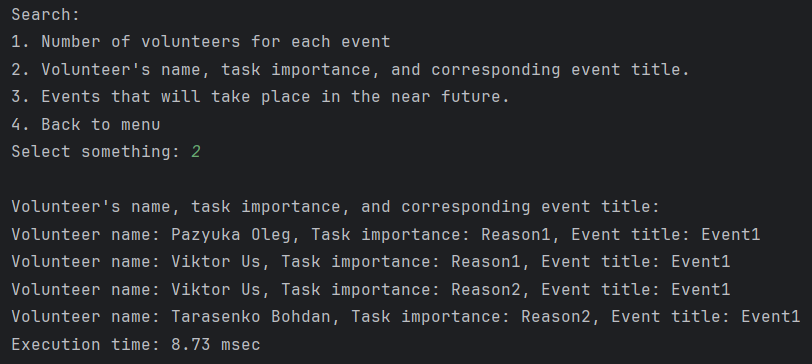
**Пошук даних**

****

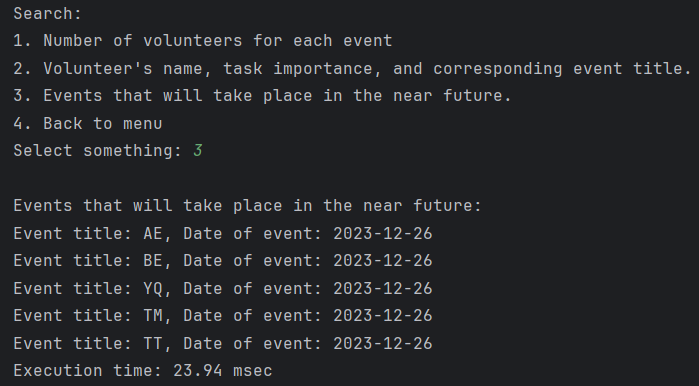
1. Що ми отримуємо, при виборі 1 пункту:



1. Що ми отримуємо, при виборі 2 пункту:



1. Що ми отримуємо, при виборі 3 пункту:



Також можна побачити час виконання запиту у мілісекундах, після виведення даних.

Наприклад, для 1 варіанту пошуку, час виконання запиту – 5.99 мс. Для 2 варіанту пошуку – 8.73 мс. Для 3 варіанту пошуку – 23.94 мс.

**Завдання 4**

**Шаблон MVC**

**MVC** визначає архітектурний шаблон програмування, який включає три основні компоненти: Модель (Model), Вид (View) та Контролер (Controller). Цей шаблон дозволяє розділити логічні частини програми, щоб полегшити розробку, управління та розуміння коду.

Основні компоненти шаблону MVC:

**Model** – представляє клас, що описує логіку використовуваних даних. Клас реалізований у файлі model.py, у ньому відбуваються найважчі процеси (вставка, видалення, оновлення, пошук, рандомізація даних, звернення до бази даних) і після виконаної події відправляє результат до View.

**View** – це консольний інтерфейс, з яким взаємодіє користувач. Відповідає за введення/виведення даних. У програмі це реалізовано за допомогою файлу view.py (клас View та клас Menu).

**Controller** – забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. У програмі це реалізовано у файлі controller.py

Користуючись шаблоном MVC, розробники можуть розділити програмний код на логічно зв'язані компоненти, що полегшує розуміння, тестування та зміну програми. Це особливо корисно для великих проектів, де структурованість і підтримка коду грають важливу роль.

**Код програми**

**main.py**

**from** controller **import** Controller

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

controller = Controller()

controller.run()

**model.py**

**import** psycopg2

**class** Model:

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

self.conn = psycopg2.connect(

dbname='volunteer\_management\_system',

user='postgres',

password='1111',

host='localhost',

port=3000

)

**def** add\_event(self, event\_id, title, organizer, date\_of\_event):

c = self.conn.cursor()

c.execute('INSERT INTO events(event\_id, title, organizer, date\_of\_event) VALUES(%s, %s, %s, %s);',

(event\_id, title, organizer, date\_of\_event))

self.conn.commit()

**def** add\_task(self, task\_id, importance, event\_id):

c = self.conn.cursor()

c.execute('INSERT INTO tasks(task\_id, importance, event\_id) VALUES(%s, %s, %s);',

(task\_id, importance, event\_id))

self.conn.commit()

**def** add\_volunteer(self, volunteer\_id, name, state\_date\_of\_volunteering):

c = self.conn.cursor()

c.execute('INSERT INTO volunteers(volunteer\_id, name, state\_date\_of\_volunteering) VALUES(%s, %s, %s);',

(volunteer\_id, name, state\_date\_of\_volunteering))

self.conn.commit()

**def** add\_volunteer\_task(self, volunteer\_task\_id, volunteer\_id, task\_id):

c = self.conn.cursor()

c.execute('INSERT INTO volunteer\_tasks(volunteer\_task\_id, volunteer\_id, task\_id) VALUES(%s, %s, %s);',

(volunteer\_task\_id, volunteer\_id, task\_id))

self.conn.commit()

**def** get\_events(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM events;')

**return** c.fetchall()

**def** get\_tasks(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM tasks;')

**return** c.fetchall()

**def** get\_volunteers(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM volunteers;')

**return** c.fetchall()

**def** get\_volunteer\_tasks(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM volunteer\_tasks;')

**return** c.fetchall()

**def** update\_event(self, event\_id, title, organizer, date\_of\_event, id):

c = self.conn.cursor()

c.execute('UPDATE events SET event\_id=%s, title=%s, organizer=%s, date\_of\_event=%s WHERE event\_id=%s',

(event\_id, title, organizer, date\_of\_event, id))

self.conn.commit()

**def** update\_task(self, task\_id, importance, event\_id, id):

c = self.conn.cursor()

c.execute('UPDATE tasks SET task\_id=%s, importance=%s, event\_id=%s WHERE task\_id=%s',

(task\_id, importance, event\_id, id))

self.conn.commit()

**def** update\_volunteer(self, volunteer\_id, name, state\_date\_of\_volunteering, id):

c = self.conn.cursor()

c.execute('UPDATE volunteers SET volunteer\_id=%s, name=%s, state\_date\_of\_volunteering=%s WHERE volunteer\_id=%s',

(volunteer\_id, name, state\_date\_of\_volunteering, id))

self.conn.commit()

**def** update\_volunteer\_task(self, volunteer\_task\_id, volunteer\_id, task\_id, id):

c = self.conn.cursor()

c.execute('UPDATE volunteer\_tasks SET volunteer\_task\_id=%s, volunteer\_id=%s, task\_id=%s WHERE volunteer\_task\_id=%s',

(volunteer\_task\_id, volunteer\_id, task\_id, id))

self.conn.commit()

**def** delete\_event(self, event\_id):

c = self.conn.cursor()

c.execute('DELETE FROM events WHERE "event\_id"=%s', (event\_id,))

self.conn.commit()

**def** delete\_task(self, task\_id):

c = self.conn.cursor()

c.execute('DELETE FROM tasks WHERE "task\_id"=%s', (task\_id,))

self.conn.commit()

**def** delete\_volunteer(self, volunteer\_id):

c = self.conn.cursor()

c.execute('DELETE FROM volunteers WHERE "volunteer\_id"=%s', (volunteer\_id,))

self.conn.commit()

**def** delete\_volunteer\_task(self, volunteer\_task\_id):

c = self.conn.cursor()

c.execute('DELETE FROM volunteer\_tasks WHERE "volunteer\_task\_id"=%s', (volunteer\_task\_id,))

self.conn.commit()

**def** get\_number\_of\_volunteers\_for\_each\_event(self):

c = self.conn.cursor()

c.execute('SELECT e.event\_id, e.title, COUNT(vt.volunteer\_id) AS total\_volunteers FROM events e JOIN volunteer\_tasks vt ON e.event\_id = vt.task\_id GROUP BY e.event\_id, e.title;')

**return** c.fetchall()

**def** get\_volunteer\_task\_event(self):

c = self.conn.cursor()

c.execute('SELECT v.name AS volunteer\_name, t.importance AS task\_importance, e.title AS event\_title FROM volunteers v JOIN volunteer\_tasks vt ON v.volunteer\_id = vt.volunteer\_id JOIN tasks t ON vt.task\_id = t.task\_id JOIN events e ON t.event\_id = e.event\_id;')

**return** c.fetchall()

**def** get\_upcoming\_events(self):

c = self.conn.cursor()

c.execute('SELECT title AS event\_title, date\_of\_event FROM events WHERE date\_of\_event >= CURRENT\_DATE ORDER BY date\_of\_event LIMIT 5;')

**return** c.fetchall()

**def** add\_random\_fields(self, number):

c = self.conn.cursor()

c.execute("INSERT INTO events (event\_id, title, organizer, date\_of\_event) SELECT row\_number() OVER () + (SELECT COALESCE(MAX(event\_id), 0) FROM events), chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), '2023-01-01'::date + (random() \* (365 \* 10))::integer FROM generate\_series(1, %s);",

(number,))

self.conn.commit()

**controller.py**

**import** time

**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.view.show\_menu()

**if** choice == '7':

**break**

**if** choice == '6':

self.process\_search\_option()

**elif** choice **in** ['1', '2', '3', '4', '5']:

self.process\_menu\_choice(choice)

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_menu\_choice(self, choice):

**while** True:

table = self.view.show\_tables()

**if** table == '6':

**break**

**if** choice == '1':

self.process\_add\_option(table)

**elif** choice == '2':

self.process\_add\_random\_option(table)

**elif** choice == '3':

self.process\_view\_option(table)

**elif** choice == '4':

self.process\_update\_option(table)

**elif** choice == '5':

self.process\_delete\_option(table)

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_add\_option(self, table):

**if** table == '1':

self.view.show\_message("\nAdding event:")

self.add\_event()

**elif** table == '2':

self.view.show\_message("\nAdding task:")

self.add\_task()

**elif** table == '3':

self.view.show\_message("\nAdding volunteer:")

self.add\_volunteer()

**elif** table == '4':

self.view.show\_message("\nAdding volunteer task:")

self.add\_volunteer\_task()

**elif** table == '5':

self.view.show\_menu()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_add\_random\_option(self, table):

**if** table == '1':

self.view.show\_message("\nAdding random events:")

self.add\_random\_fields()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_view\_option(self, table):

**if** table == '1':

self.show\_events()

**elif** table == '2':

self.show\_tasks()

**elif** table == '3':

self.show\_volunteers()

**elif** table == '4':

self.show\_volunteer\_tasks()

**elif** table == '5':

self.view.show\_menu()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_update\_option(self, table):

**if** table == '1':

self.view.show\_message("\nUpdating event:")

self.update\_event()

**elif** table == '2':

self.view.show\_message("\nUpdating task:")

self.update\_task()

**elif** table == '3':

self.view.show\_message("\nUpdating volunteer:")

self.update\_volunteer()

**elif** table == '4':

self.view.show\_message("\nUpdating volunteer task:")

self.update\_volunteer\_task()

**elif** table == '5':

self.view.show\_menu()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_delete\_option(self, table):

**if** table == '1':

self.view.show\_message("\nDeleting event:")

self.delete\_event()

**elif** table == '2':

self.view.show\_message("\nDeleting task:")

self.delete\_task()

**elif** table == '3':

self.view.show\_message("\nDeleting volunteer:")

self.delete\_volunteer()

**elif** table == '4':

self.view.show\_message("\nDeleting volunteer task:")

self.delete\_volunteer\_task()

**else**:

self.view.show\_message("Wrong choice. Try again.")

**def** process\_search\_option(self):

option = self.view.show\_search()

**if** option == '1':

start\_time = time.time()

self.show\_number\_of\_volunteers\_for\_each\_event()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Execution time: {elapsed\_time:.2f} msec")

**elif** option == '2':

start\_time = time.time()

self.show\_volunteer\_and\_task\_and\_event()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Execution time: {elapsed\_time:.2f} msec")

**elif** option == '3':

start\_time = time.time()

self.show\_upcoming\_events()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

**print**(f"Execution time: {elapsed\_time:.2f} msec")

**else**:

self.view.show\_menu()

**def** add\_event(self):

**try**:

event\_id, title, organizer, date\_of\_event = self.view.get\_event\_input()

self.model.add\_event(event\_id, title, organizer, date\_of\_event)

self.view.show\_message("Event added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_task(self):

**try**:

task\_id, importance, event\_id = self.view.get\_task\_input()

self.model.add\_task(task\_id, importance, event\_id)

self.view.show\_message("Task added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_volunteer(self):

**try**:

volunteer\_id, name, state\_date\_of\_volunteering = self.view.get\_volunteer\_input()

self.model.add\_volunteer(volunteer\_id, name, state\_date\_of\_volunteering)

self.view.show\_message("Volunteer added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_volunteer\_task(self):

**try**:

volunteer\_task\_id, volunteer\_id, task\_id = self.view.get\_volunteer\_task\_input()

self.model.add\_volunteer\_task(volunteer\_task\_id, volunteer\_id, task\_id)

self.view.show\_message("Volunteer Task added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_events(self):

**try**:

events = self.model.get\_events()

self.view.show\_events(events)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_tasks(self):

**try**:

tasks = self.model.get\_tasks()

self.view.show\_tasks(tasks)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_volunteers(self):

**try**:

volunteers = self.model.get\_volunteers()

self.view.show\_volunteers(volunteers)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_volunteer\_tasks(self):

**try**:

volunteer\_tasks = self.model.get\_volunteer\_tasks()

self.view.show\_volunteer\_tasks(volunteer\_tasks)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_event(self):

**try**:

id = self.view.get\_event\_id()

event\_id, title, organizer, date\_of\_event = self.view.get\_event\_input()

self.model.update\_event(event\_id, title, organizer, date\_of\_event, id)

self.view.show\_message("Event updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_task(self):

**try**:

id = self.view.get\_task\_id()

task\_id, importance, event\_id = self.view.get\_task\_input()

self.model.update\_task(task\_id, importance, event\_id, id)

self.view.show\_message("Task updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_volunteer(self):

**try**:

id = self.view.get\_volunteer\_id()

volunteer\_id, name, state\_date\_of\_volunteering = self.view.get\_volunteer\_input()

self.model.update\_volunteer(volunteer\_id, name, state\_date\_of\_volunteering, id)

self.view.show\_message("Volunteer updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** update\_volunteer\_task(self):

**try**:

id = self.view.get\_volunteer\_task\_id()

volunteer\_task\_id, volunteer\_id, task\_id = self.view.get\_volunteer\_task\_input()

self.model.update\_volunteer\_task(volunteer\_task\_id, volunteer\_id, task\_id, id)

self.view.show\_message("Volunteer Task updated successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_event(self):

**try**:

event\_id = self.view.get\_event\_id()

self.model.delete\_event(event\_id)

self.view.show\_message("Event deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_task(self):

**try**:

task\_id = self.view.get\_task\_id()

self.model.delete\_task(task\_id)

self.view.show\_message("Task deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_volunteer(self):

**try**:

volunteer\_id = self.view.get\_volunteer\_id()

self.model.delete\_volunteer(volunteer\_id)

self.view.show\_message("Volunteer deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** delete\_volunteer\_task(self):

**try**:

volunteer\_task\_id = self.view.get\_volunteer\_task\_id()

self.model.delete\_volunteer\_task(volunteer\_task\_id)

self.view.show\_message("Volunteer Task deleted successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_number\_of\_volunteers\_for\_each\_event(self):

**try**:

number\_of\_volunteers = self.model.get\_number\_of\_volunteers\_for\_each\_event()

self.view.show\_number\_of\_volunteers\_for\_each\_event(number\_of\_volunteers)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_volunteer\_and\_task\_and\_event(self):

**try**:

volunteer\_task\_event = self.model.get\_volunteer\_task\_event()

self.view.show\_volunteer\_task\_event(volunteer\_task\_event)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** show\_upcoming\_events(self):

**try**:

rows = self.model.get\_upcoming\_events()

self.view.show\_upcoming\_events(rows)

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**def** add\_random\_fields(self):

**try**:

number = self.view.get\_number()

self.model.add\_random\_fields(number)

self.view.show\_message("Random fields added successfully!")

**except** Exception **as** e:

self.view.show\_message(f"Something went wrong: {e}")

**view.py**

**from** datetime **import** datetime

**class** View:

**def** show\_menu(self):

self.show\_message("\nMenu:")

self.show\_message("1. Add row")

self.show\_message('2. Generating `randomized` data (only for "Events")')

self.show\_message("3. Show table")

self.show\_message("4. Update row")

self.show\_message("5. Delete row")

self.show\_message("6. Search")

self.show\_message("7. Exit")

choice = input("Select your choice: ")

**return** choice

**def** show\_tables(self):

self.show\_message("\nTables:")

self.show\_message("1. Events")

self.show\_message("2. Tasks")

self.show\_message("3. Volunteers")

self.show\_message("4. Volunteer Tasks")

self.show\_message("5. Back to menu")

table = input("Select table: ")

**return** table

**def** show\_search(self):

self.show\_message("\nSearch:")

self.show\_message("1. Number of volunteers for each event")

self.show\_message("2. Volunteer's name, task importance, and corresponding event title.")

self.show\_message("3. Events that will take place in the near future.")

self.show\_message("4. Back to menu")

choice = input("Select something: ")

**return** choice

**def** show\_events(self, events):

**print**("\nEvents:")

**for** event **in** events:

**print**(f"Event ID: {event[0]}, Title: {event[1]}, Organizer: {event[2]}, Date of event: {event[3]}")

**def** show\_tasks(self, tasks):

**print**("\nTasks:")

**for** task **in** tasks:

**print**(f"Task ID: {task[0]}, Importance: {task[1]}, Event ID: {task[2]}")

**def** show\_volunteers(self, volunteers):

**print**("\nVolunteers:")

**for** volunteer **in** volunteers:

**print**(f"Volunteer ID: {volunteer[0]}, Name: {volunteer[1]}, State date of volunteering: {volunteer[2]}")

**def** show\_volunteer\_tasks(self, volunteer\_tasks):

**print**("\nVolunteer Tasks:")

**for** volunteer\_task **in** volunteer\_tasks:

**print**(f"Volunteer task ID: {volunteer\_task[0]}, Volunteer ID: {volunteer\_task[1]}, Task ID: {volunteer\_task[2]}")

**def** get\_event\_input(self):

**while** True:

**try**:

event\_id = input("Enter event ID: ")

**if** event\_id.strip():

event\_id = int(event\_id)

**break**

**else**:

**print**("Event ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

title = input("Enter title: ")

**if** title.strip():

**break**

**else**:

**print**("Title cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

organizer = input("Enter organizer: ")

**if** organizer.strip():

**break**

**else**:

**print**("Organizer cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

date\_of\_event = input("Enter date of event (YYYY-MM-DD): ")

**if** date\_of\_event.strip():

datetime.strptime(date\_of\_event, "%Y-%m-%d")

**break**

**else**:

**print**("Date of event cannot be empty.")

**except** ValueError:

**print**("Invalid date format. Please use YYYY-MM-DD.")

**return** event\_id, title, organizer, date\_of\_event

**def** get\_task\_input(self):

**while** True:

**try**:

task\_id = input("Enter task ID: ")

**if** task\_id.strip():

task\_id = int(task\_id)

**break**

**else**:

**print**("Task ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

importance = input("Enter importance: ")

**if** importance.strip():

**break**

**else**:

**print**("Importance cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

event\_id = input("Enter event ID: ")

**if** event\_id.strip():

event\_id = int(event\_id)

**break**

**else**:

**print**("Event ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**return** task\_id, importance, event\_id

**def** get\_volunteer\_input(self):

**while** True:

**try**:

volunteer\_id = input("Enter volunteer ID: ")

**if** volunteer\_id.strip():

volunteer\_id = int(volunteer\_id)

**break**

**else**:

**print**("Volunteer ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

name = input("Enter name: ")

**if** name.strip():

**break**

**else**:

**print**("Name cannot be empty.")

**except** ValueError:

**print**("It must be a string.")

**while** True:

**try**:

state\_date\_of\_volunteering = input("Enter state date of volunteering (YYYY-MM-DD): ")

**if** state\_date\_of\_volunteering.strip():

datetime.strptime(state\_date\_of\_volunteering, "%Y-%m-%d")

**break**

**else**:

**print**("State date of volunteering cannot be empty.")

**except** ValueError:

**print**("Invalid date format. Please use YYYY-MM-DD.")

**return** volunteer\_id, name, state\_date\_of\_volunteering

**def** get\_volunteer\_task\_input(self):

**while** True:

**try**:

volunteer\_task\_id = input("Enter volunteer task ID: ")

**if** volunteer\_task\_id.strip():

volunteer\_task\_id = int(volunteer\_task\_id)

**break**

**else**:

**print**("Volunteer task ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

volunteer\_id = input("Enter volunteer ID: ")

**if** volunteer\_id.strip():

volunteer\_id = int(volunteer\_id)

**break**

**else**:

**print**("Volunteer ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**while** True:

**try**:

task\_id = input("Enter task ID: ")

**if** task\_id.strip():

task\_id = int(task\_id)

**break**

**else**:

**print**("Task ID cannot be empty.")

**except** ValueError:

**print**("It must be a number.")

**return** volunteer\_task\_id, volunteer\_id, task\_id

**def** get\_event\_id(self):

**while** True:

**try**:

id = int(input("Enter event ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** get\_task\_id(self):

**while** True:

**try**:

id = int(input("Enter task ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** get\_volunteer\_id(self):

**while** True:

**try**:

id = int(input("Enter volunteer ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** get\_volunteer\_task\_id(self):

**while** True:

**try**:

id = int(input("Enter volunteer task ID: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** id

**def** show\_number\_of\_volunteers\_for\_each\_event(self, rows):

**print**("\nNumber of volunteers for each event:")

**for** row **in** rows:

**print**(f"Event ID: {row[0]}, Title: {row[1]}, Total volunteers: {row[2]}")

**def** show\_volunteer\_task\_event(self, rows):

**print**("\nVolunteer's name, task importance, and corresponding event title:")

**for** row **in** rows:

**print**(f"Volunteer name: {row[0]}, Task importance: {row[1]}, Event title: {row[2]}")

**def** show\_upcoming\_events(self, rows):

**print**("\nEvents that will take place in the near future:")

**for** row **in** rows:

**print**(f"Event title: {row[0]}, Date of event: {row[1]}")

**def** show\_message(self, message):

**print**(message)

**def** get\_number(self):

**while** True:

**try**:

number = int(input("Enter the number: "))

**break**

**except** ValueError:

**print**("It must be a number.")

**return** number