



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 III курсу

групи КВ-12

Дмитрієвцев М.В.

Перевірив:

Павловский В. І.

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 у github: <https://github.com/Dmitriievtsev-Mykhail/BD.git>

Графічне подання концептуальної моделі зображено на рисунку 1

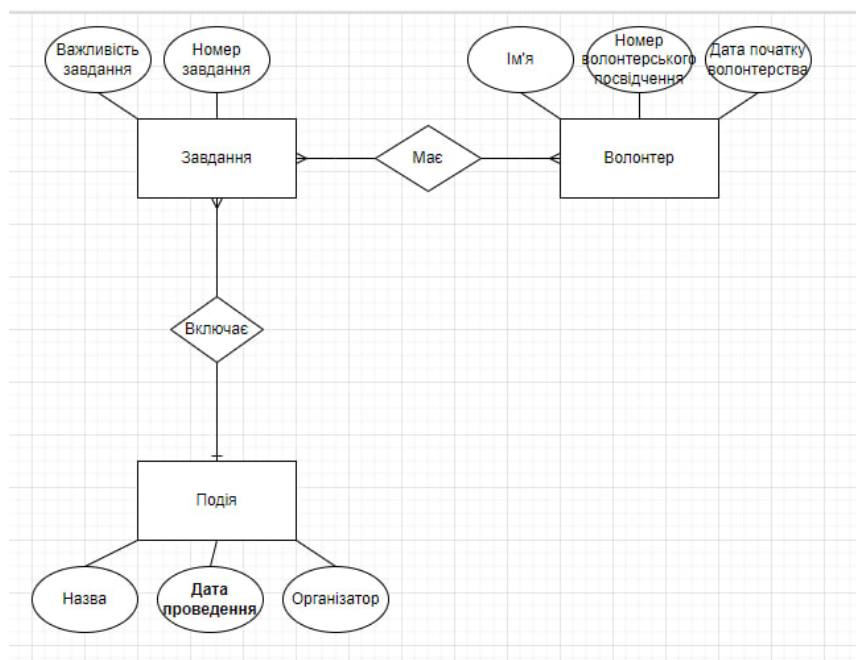


Рисунок 1 – ER-діаграма, побудована за нотацією «Crow`s foot»

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.

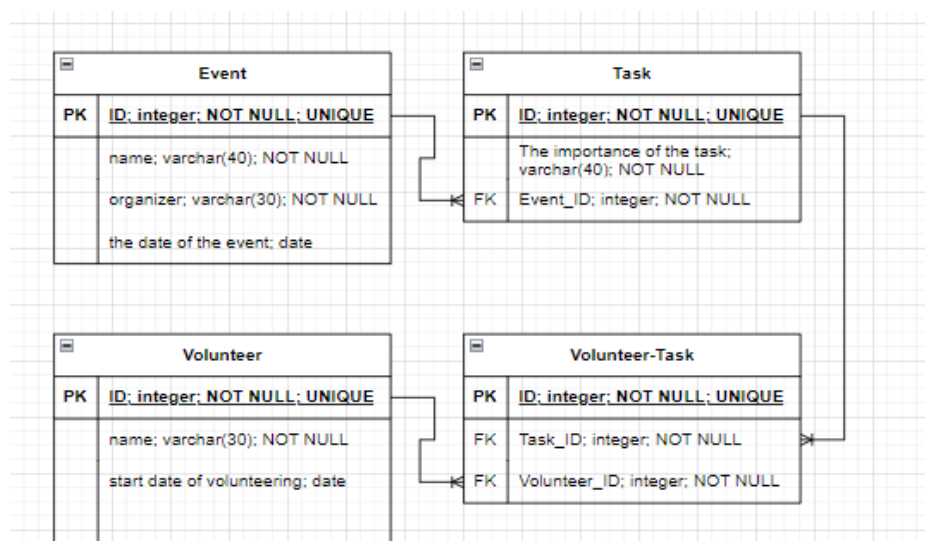


Рисунок 2 – Схема бази даних у графічному вигляді

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)

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice:
```

Рисунок 3 – Структура меню користувача

```
Tables:
1. Events
2. Tasks
3. Volunteers
4. Volunteer Tasks
5. Back to menu
Select table: |
```

Рисунок 4 – Варіанти таблиць, 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.

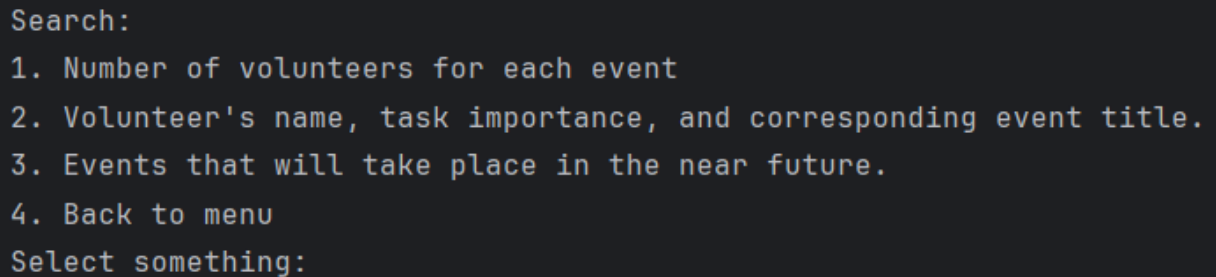
3) **Show table** служить для показу таблиці. Перед виведенням, користувач обирає, яку саме таблицю потрібно вивести. Після цього на екрані виводяться всі поля обраної таблиці БД.

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):



```
Search:
1. Number of volunteers for each event
2. Volunteer's name, task importance, and corresponding event title.
3. Events that will take place in the near future.
4. Back to menu
Select something:
```

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.

2. 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.

3. 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.

4. Back to menu. Повернення до основного меню.

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

7) **Exit** служить для виходу з програми.

Фрагмент коду (файл controller.py), в якому наведено головний цикл роботи програми

```
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.")
```

Фрагменти коду (файл model.py), в якому наведено функції внесення, редагування, видалення та генерації даних у базі даних

Функції внесення даних:

```
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()
```

Функції оновлення даних:

```
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()

```

Функції видалення значень:

```

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()

```

Функція генерування даних (тільки для подій):

```

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()

```

Завдання 1

Додавання рядка

Таблиця “tasks” до:

	task_id [PK] integer	importance character varying (40)	event_id integer
1	1	Reason1	1
2	2	Reason2	1
3	3	Reason3	2

```

Menu:
1. Add row
2. Generating `randomized` data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 1

Tables:
1. Events
2. Tasks
3. Volunteers
4. Volunteer Tasks
5. Back to menu
Select table: 4

Adding task:
Enter task ID: 4
Enter importance: Test
Enter event ID: 111
Task added successfully!

```

Таблиця “tasks” після:

	task_id [PK] integer	importance character varying (40)	event_id integer
1	1	Reason1	1
2	2	Reason2	1
3	3	Reason3	2
4	4	Test	111

Перегляд таблиці

Таблиця “volunteers”:

	volunteer_id [PK] integer	name character varying (30)	state_date_of_volunteering date
1	1	Pazyuka Oleg	2014-05-02
2	2	Viktor Us	2018-03-10
3	3	Tarasenko Bohdan	2020-08-04

Menu:

1. Add row
2. Generating `randomized` data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit

Select your choice: 3

Tables:

1. Events
2. Tasks
3. Volunteers
4. Volunteer Tasks
5. Back to menu

Select table: 3

Volunteers:

Volunteer ID: 1, Name: Pazyuka Oleg, State date of volunteering: 2014-05-02

Volunteer ID: 2, Name: Viktor Us, State date of volunteering: 2018-03-10

Volunteer ID: 3, Name: Tarasenko Bohdan, State date of volunteering: 2020-08-04

Оновлення рядка

Таблиця "tasks" до:

	task_id [PK] integer	importance character varying (40)	event_id integer
1	1	Reason1	1
2	2	Reason2	1
3	3	Reason3	2
4	4	Test	111

```

Menu:
1. Add row
2. Generating `randomized` data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 4

Tables:
1. Events
2. Tasks
3. Volunteers
4. Volunteer Tasks
5. Back to menu
Select table: 2

Updating task:
Enter task ID: 4
Enter task ID: 4
Enter importance: Test1
Enter event ID: 222
Task updated successfully!

```

Таблиця “tasks” після:

	task_id [PK] integer	importance character varying (40)	event_id integer
1	1	Reason1	1
2	2	Reason2	1
3	3	Reason3	2
4	4	Test1	222

Видалення рядка

Таблиця “events” до:

	event_id [PK] integer	title character varying (40)	organizer character varying (30)	date_of_event date
1	100005	PK	IA	2024-10-22
2	100004	KQ	SO	2032-07-13
3	100003	PX	ON	2032-11-16
4	100002	BA	SY	2031-03-03
5	100001	BY	MS	2025-12-29
6	100000	HD	DF	2027-12-10
7	99999	QB	JO	2025-05-23
8	99998	AE	CU	2030-03-13
9	99997	DU	XE	2032-05-27
10	99996	DK	FU	2027-11-19
11	99995	RI	QG	2029-10-01
12	99994	EH	NP	2023-05-07
13	99993	PQ	QL	2026-09-20

```



Menu:
1. Add row
2. Generating `randomized` data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 5

Tables:
1. Events
2. Tasks
3. Volunteers
4. Volunteer Tasks
5. Back to menu
Select table: 1

Deleting event:
Enter event ID: 100005
Event deleted successfully!

```

Таблиця “events” після:

	eventLid [PK] integer 	title character varying (40) 	organizer character varying (30) 	date_of_event date 
1	100004	KQ	SO	2032-07-13
2	100003	PX	ON	2032-11-16
3	100002	BA	SY	2031-03-03
4	100001	BY	MS	2025-12-29
5	100000	HD	DF	2027-12-10
6	99999	QB	JO	2025-05-23
7	99998	AE	CU	2030-03-13
8	99997	DU	XE	2032-05-27
9	99996	DK	FU	2027-11-19
10	99995	RI	QG	2029-10-01
11	99994	EH	NP	2023-05-07
12	99993	PQ	QL	2026-09-20
13	99992	LC	IP	2025-01-10

Вихід

```

Menu:
1. Add row
2. Generating `randomized` data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 7

Process finished with exit code 0

```

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:

	event_id [PK] integer	title character varying (40)	organizer character varying (30)	date_of_event date
1	100004	KQ	SO	2032-07-13
2	100003	PX	ON	2032-11-16
3	100002	BA	SY	2031-03-03
4	100001	BY	MS	2025-12-29
5	100000	HD	DF	2027-12-10
6	99999	QB	JO	2025-05-23
7	99998	AE	CU	2030-03-13
8	99997	DU	XE	2032-05-27
9	99996	DK	FU	2027-11-19
10	99995	RI	QG	2029-10-01
11	99994	EH	NP	2023-05-07
12	99993	PQ	QL	2026-09-20
13	99992	LC	IP	2025-01-10
14	99991	NN	MC	2025-02-07
15	99990	UI	SG	2029-02-15
16	99989	KN	OI	2024-03-26
17	99988	JV	CR	2028-10-07
18	99987	EH	VT	2032-08-27
19	99986	VM	DR	2030-07-16
20	99985	VE	CP	2028-08-13
21	99984	WR	OU	2031-01-18
22	99983	UU	MN	2025-10-06

	event_id [PK] integer	title character varying (40)	organizer character varying (30)	date_of_event date
23	99982	RB	SM	2023-06-26
24	99981	EV	PR	2025-04-23
25	99980	UD	CH	2026-01-06
26	99979	JQ	FV	2031-09-17
27	99978	HN	FF	2026-11-17
28	99977	RU	TX	2026-02-18
29	99976	AW	PE	2024-01-04
30	99975	BR	AT	2031-02-27
31	99974	VQ	OM	2026-10-19
32	99973	VI	JI	2026-04-27
33	99972	VV	LG	2025-09-14
34	99971	BW	FT	2024-06-01
35	99970	YX	RP	2030-03-03
36	99969	XT	QS	2025-01-02
37	99968	YT	AC	2023-05-09
38	99967	DK	GR	2031-10-22
39	99966	KR	CM	2028-11-08
40	99965	SM	KP	2029-10-30
41	99964	UN	SL	2029-09-04
42	99963	OT	MK	2028-02-08
43	99962	NG	ST	2031-07-06
44	99961	EY	GT	2032-04-27

```
Menu:  
1. Add row  
2. Generating 'randomized' data (only for "Events")  
3. Show table  
4. Update row  
5. Delete row  
6. Search  
7. Exit  
Select your choice: 2  
  
Tables:  
1. Events  
2. Tasks  
3. Volunteers  
4. Volunteer Tasks  
5. Back to menu  
Select table: 1  
  
Adding random events:  
Enter the number: 10  
Random fields added successfully!
```

Таблиця “Events” після:

	event_id [PK] integer	title character varying (40)	organizer character varying (30)	date_of_event date
1	100014	JY	OV	2028-07-09
2	100013	NL	EM	2030-09-26
3	100012	AH	OW	2028-11-10
4	100011	JO	PR	2030-11-16
5	100010	FY	DW	2023-09-19
6	100009	RP	OK	2023-04-26
7	100008	DF	DI	2026-01-19
8	100007	FR	PX	2026-07-08
9	100006	TO	ME	2032-04-05
10	100005	QB	KM	2029-02-09
11	100004	KQ	SO	2032-07-13
12	100003	PX	ON	2032-11-16
13	100002	BA	SY	2031-03-03
14	100001	BY	MS	2025-12-29
15	100000	HD	DF	2027-12-10
16	99999	QB	JO	2025-05-23
17	99998	AE	CU	2030-03-13
18	99997	DU	XE	2032-05-27
19	99996	DK	FU	2027-11-19
20	99995	RI	QG	2029-10-01
21	99994	EH	NP	2023-05-07
22	99993	PQ	QL	2026-09-20

Завдання 3

Пошук даних

```
Menu:
1. Add row
2. Generating 'randomized' data (only for "Events")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 6

Search:
1. Number of volunteers for each event
2. Volunteer's name, task importance, and corresponding event title.
3. Events that will take place in the near future.
4. Back to menu
Select something: |
```

1. Що ми отримуємо, при виборі 1 пункту:

```
Search:
1. Number of volunteers for each event
2. Volunteer's name, task importance, and corresponding event title.
3. Events that will take place in the near future.
4. Back to menu
Select something: 1

Number of volunteers for each event:
Event ID: 1, Title: Event1, Total volunteers: 2
Event ID: 2, Title: Event2, Total volunteers: 2
Execution time: 5.99 msec
```

2. Що ми отримуємо, при виборі 2 пункту:

```
Search:
1. Number of volunteers for each event
2. Volunteer's name, task importance, and corresponding event title.
3. Events that will take place in the near future.
4. Back to menu
Select something: 2

Volunteer's name, task importance, and corresponding event title:
Volunteer name: Pazyuka Oleg, Task importance: Reason1, Event title: Event1
Volunteer name: Viktor Us, Task importance: Reason1, Event title: Event1
Volunteer name: Viktor Us, Task importance: Reason2, Event title: Event1
Volunteer name: Tarasenko Bohdan, Task importance: Reason2, Event title: Event1
Execution time: 8.73 msec
```

3. Що ми отримуємо, при виборі 3 пункту:

```
Search:
1. Number of volunteers for each event
2. Volunteer's name, task importance, and corresponding event title.
3. Events that will take place in the near future.
4. Back to menu
Select something: 3

Events that will take place in the near future:
Event title: AE, Date of event: 2023-12-26
Event title: BE, Date of event: 2023-12-26
Event title: YQ, Date of event: 2023-12-26
Event title: TM, Date of event: 2023-12-26
Event title: IT, Date of event: 2023-12-26
Execution time: 23.94 msec
```

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 найважливіші 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
            except ValueError:
                print("Event ID cannot be empty.")
                print("It must be a number.")
        while True:
            try:
                title = input("Enter title: ")
                if title.strip():
                    break
            except ValueError:
                print("Title cannot be empty.")
                print("It must be a string.")
        while True:
            try:
                organizer = input("Enter organizer: ")
                if organizer.strip():
                    break
            except ValueError:
                print("Organizer cannot be empty.")
                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

```