



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
**Кафедра системного програмування та спеціалізованих комп'ютерних
систем**

Розрахункова робота

з дисципліни **Бази даних і засоби управління**

*на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД
PostgreSQL”*

Виконала:
студентка III курсу
групи КВ-34
Фіалковська Ольга
Перевірив:
Павловський В. І.

Київ – 2025

Мета: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Завдання:

- Реалізувати функції перегляду, внесення, редагування та видалення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
- Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
- 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.
- Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Опис предметної області

Система управління завданнями та проєктами для фрілансерів дозволяє взаємодіяти замовникам і виконавцям через різні онлайн-платформи. Кожен користувач реєструється на платформі, вказуючи свої дані. Замовник може створювати проєкти, а фрілансер — їх виконувати. Проєкт має назву, дати початку, дедлайну та завершення. Усі проєкти пов'язані з певною платформою, що забезпечує організовану роботу та контроль виконання завдань.

Опис сутностей

1. Платформа – це середовище, де працюють фрілансери та замовники. Вона зберігає основну інформацію: назву, дату створення та унікальний ID платформи.
2. Фрілансер – користувач, який виконує проєкти. Має такі дані: ім'я, прізвище, email, пароль і ID фрілансера.
3. Замовник – користувач, який створює проєкти для фрілансерів. Має ім'я, прізвище, email, пароль і ID замовника.
4. Проєкт – це завдання, яке створює замовник. Містить назву, дедлайн, дату початку, кінцеву дату та ID проєкта.

Зв'язки між сутностями:

- Фрілансер зареєстрований на платформі – показує, де він працює.
(N:M)
Фрілансер може мати багато платформ. Також, як і платформа може мати багато фрілансерів.
- Замовник зареєстрований на платформі – показує, де він створює проєкти.
(N:M)
Замовник може мати багато платформ. Також, як і платформа може мати багато замовників.
- Замовник створює проєкт – визначає, хто є автором проєкту.
(1:N)
Замовник може мати багато проєктів, але проєкт може мати всього одного замовника.
- Фрілансер виконує проєкт – показує, хто його виконавець.
(1:N)
Фрілансер може мати багато проєктів, але проєкт може мати всього одного фрілансера.

Опис зв'язків між сутностями

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

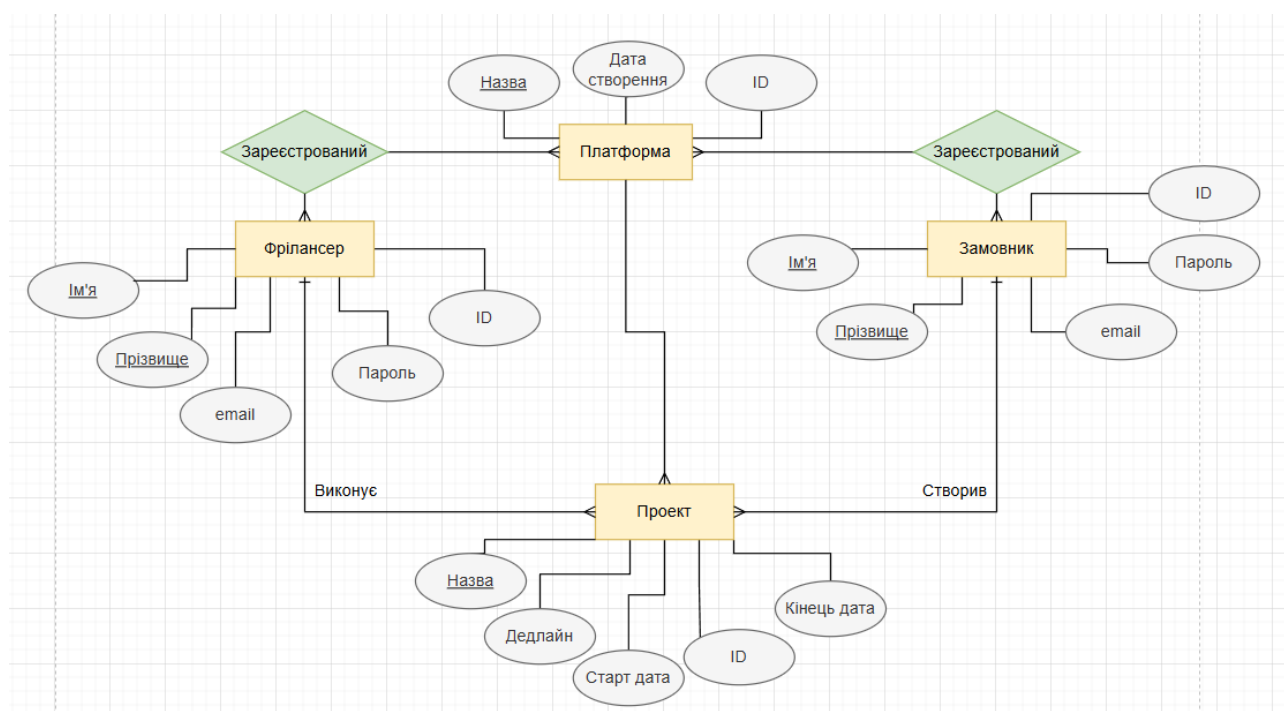
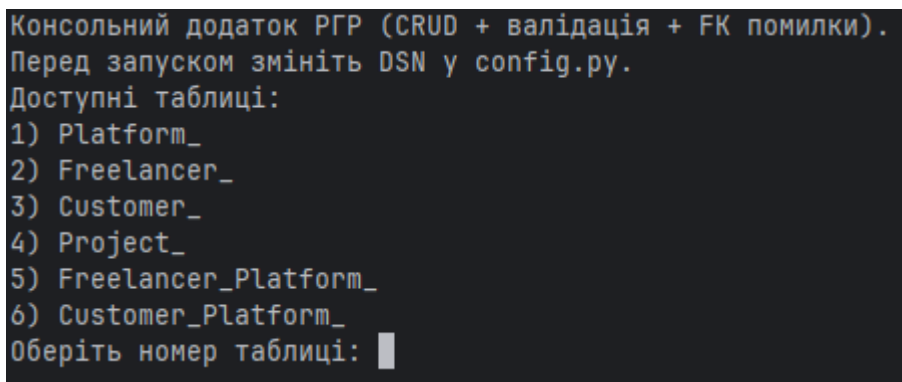


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

Реалізування функцій засобами консольного інтерфейсу (Завдання 1)

Початкова сторінка

У початковому вікні програми відображається назва застосунку та підказка для користувача. Програма підключається до бази даних і показує перелік доступних таблиць. Користувач обирає потрібну таблицю за номером, щоб виконати подальші дії — перегляд, додавання, редагування або видалення записів.



```
Консольний додаток РГР (CRUD + валідація + FK помилки).  
Перед запуском змініть DSN у config.py.  
Доступні таблиці:  
1) Platform_  
2) Freelancer_  
3) Customer_  
4) Project_  
5) Freelancer_Platform_  
6) Customer_Platform_  
Оберіть номер таблиці: █
```

Пробувала додати новий проєкт із замовником, якого немає в таблиці. Програма показала повідомлення про помилку зовнішнього ключа. Це означає, що не можна додати проєкт без існуючого замовника.

```

Консольний додаток PGP (CRUD + валідація + FK помилки).
Перед запуском змініть DSN у config.py.
Доступні таблиці:
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
Оберіть номер таблиці: 4

=== Меню ===
1) Переглянути всі рядки таблиці
2) Додати рядок
3) Редагувати рядок
4) Видалити рядок
5) Вийти
Виберіть опцію: 2
Введення нового рядка:
id (int): 8
Name (str(20)): Do homework
Deadline (date): 2025-11-08
Start date (date): 2025-10-15
End date (date): 2025-11-08
Customer_id (int?): 99
Freelancer_id (int?): 6
!!!Порушення зовнішнього ключа (немає пов'язаного запису у батьківській таблиці).

```

Вставка у дочірню таблицю при відсутності батьківського запису

Під час спроби видалити запис із таблиці Customer_, який пов'язаний із таблицею Project_, програма вивела повідомлення про неможливість видалення. Це означає, що в базі встановлено обмеження ON DELETE RESTRICT, яке не дозволяє видалити батьківський запис, якщо для нього існують залежні записи в дочірній таблиці. Таким чином, зберігається цілісність даних.

```

Консольний додаток PGP (CRUD + валідація + FK помилки).
Перед запуском змініть DSN у config.py.

=== Список таблиць ===
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
7) Вийти
Оберіть таблицю (номер): 3

=== Меню дій ===
1) Переглянути всі рядки таблиці
2) Додати рядок
3) Редагувати рядок
4) Видалити рядок
5) Назад
Виберіть опцію: 4
Введіть значення id для видалення: 1
!!!Неможливо видалити: існують залежні записи в дочірній таблиці (RESTRICT).

```

Валідація типів

Під час додавання нового рядка у таблицю Project_ я спеціально ввела неправильний формат дати (39393 замість формату YYYY-MM-DD). Програма автоматично перевірила тип даних і вивела повідомлення:

«Помилка валідації: Очікувалась дата у форматі YYYY-MM-DD».

Це означає, що працює контроль коректності введених даних (валідація типів) ще до надсилання запиту у базу. Завдяки цьому користувач бачить зрозуміле повідомлення, а програма не падає з системною помилкою.

```
Консольний додаток РГР (CRUD + валідація + FK помилки).
Перед запуском змініть DSN у config.py.

=== Список таблиць ===
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
7) Вийти
Оберіть таблицю (номер): 4

=== Меню дій ===
1) Переглянути всі рядки таблиці
2) Додати рядок
3) Редагувати рядок
4) Видалити рядок
5) Назад
Виберіть опцію: 2
Введення нового рядка:
id (int): 1
Name (str(20)): Polina
Deadline (date): 39393
!!!Помилка валідації: Очікувалась дата у форматі YYYY-MM-DD
```

Валідація типів

На цьому етапі я протестувала навігацію в консольному меню програми. Вибір таблиці та повернення назад працюють правильно — можна переходити між різними таблицями та відкривати підменю з діями (перегляд, додавання, редагування, видалення).

Консольний додаток РГР (CRUD + валідація + FK помилки).
Перед запуском змініть DSN у config.py.

```
=== Список таблиць ===
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
7) Вийти
Оберіть таблицю (номер): 1
```

```
=== Меню дій ===
1) Переглянути всі рядки таблиці
2) Додати рядок
3) Редагувати рядок
4) Видалити рядок
5) Назад
Виберіть опцію: 5
```

```
=== Список таблиць ===
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
7) Вийти
Оберіть таблицю (номер): 2
```

```
=== Меню дій ===
1) Переглянути всі рядки таблиці
2) Додати рядок
3) Редагувати рядок
4) Видалити рядок
5) Назад
Виберіть опцію: 5
```

```
=== Список таблиць ===
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
7) Вийти
Оберіть таблицю (номер): 5
```

```
=== Список таблиць ===
1) Platform_
2) Freelancer_
3) Customer_
4) Project_
5) Freelancer_Platform_
6) Customer_Platform_
7) Вийти
Оберіть таблицю (номер): 1
```

```
=== Меню дій ===
1) Переглянути всі рядки таблиці
2) Додати рядок
3) Редагувати рядок
4) Видалити рядок
5) Назад
Виберіть опцію: 1
```

```
Platform_ (1 рядків):
('freelancer', datetime.date(2000, 9, 1), 2)
```



```
(SELECT id FROM "Freelancer_" ORDER BY random() LIMIT 1)
FROM generate_series(1, 100000) AS gs;
```

Data Output Messages Notifications							
Showing rows: 1 to 1000 Page No: 1 of 100							
	Name character varying (20)	Deadline date	Start date date	End date date	id [PK] integer	Customer_id integer	Freelancer_id integer
1	Project_1	2024-01-...	2023-12-08	2024-01-...	1	870	1
2	Project_2	2024-08-...	2023-03-02	2024-04-...	2	870	1
3	Project_3	2024-06-...	2023-11-16	2024-05-...	3	870	1
4	Project_4	2024-06-...	2023-12-25	2024-05-...	4	870	1
5	Project_5	2024-01-...	2023-10-10	2024-04-...	5	870	1
6	Project_6	2024-01-...	2023-03-06	2024-11-...	6	870	1
7	Project_7	2024-07-...	2023-09-06	2024-04-...	7	870	1
8	Project_8	2024-05-...	2023-01-21	2024-09-...	8	870	1
9	Project_9	2024-12-...	2023-09-05	2024-08-...	9	870	1
10	Project_10	2024-03-...	2023-01-14	2024-11-...	10	870	1
11	Project_11	2024-12-...	2023-07-14	2024-02-...	11	870	1
12	Project_12	2024-07-...	2023-10-29	2024-11-...	12	870	1
13	Project_13	2024-08-...	2023-12-03	2024-09-...	13	870	1

Total rows: 100000 Query complete 00:00:00.251

Successfully run. Total query runtime: 251 msec. 100000 rows affected. X

PostgreSQL 18/postgres - Database connected X

CRLF Ln 1, Col 1

Таблиця Customer_:

SQL запит:

```
INSERT INTO "Customer_" (id, "Name", "Surname", "Email", "Password")
SELECT gs AS id,
initcap(chr(trunc(65 + random() * 25)::int)
|| chr(trunc(65 + random() * 25)::int)) AS "Name",
initcap(chr(trunc(65 + random() * 25)::int)
|| chr(trunc(65 + random() * 25)::int)) AS "Surname",
lower(chr(trunc(97 + random() * 25)::int)
|| chr(trunc(97 + random() * 25)::int)) AS "Email",
substring(md5(random()::text) for 30) AS "Password"
FROM generate_series(1, 1000) AS gs;
```

Data Output Messages Notifications					
Showing rows: 1 to 1000 Page No: 1 of 1					
	id [PK] integer	Name character varying (30)	Surname character varying (30)	Email character varying (30)	Password character varying (30)
1	1	Re	Mc	osr@mail.com	2a843e7224859ea00a49c012752...
2	2	Vd	Pf	hdi@mail.com	f36962e7ae254ef0b16d7a130972...
3	3	Hi	Sd	toq@mail.com	edfa5246b602610c539a0816d3e...
4	4	Ob	Jy	qaw@mail.com	4e03b8716a14a79ddc86115174c...
5	5	Cd	Oc	yyk@mail.com	845e317e3a12fd9d0da52e4b751...
6	6	Ge	Vg	xkp@mail.com	eaab3b92339b8e48fbb60231429...
7	7	Cv	Sa	jjj@mail.com	06a54e72bb11f87329867b0c55a...
8	8	Ri	Fx	yhu@mail.com	79fe1bf5576affc7f400fe0a8103df
9	9	Rg	Ix	yla@mail.com	8687cda51605437a01058f8c323...
10	10	Qr	JK	trh@mail.com	37550d116d10d30692740c748e...
11	11	Be	Bp	oux@mail.com	58877f9244ebfbae2b92d1cfc303...
12	12	Vp	Ni	vyw@mail.com	f390ff1eac693e224bc7a1b12d21...
13	13	Hn	Jg	wfh@mail.com	0fc49df5a4743be27ad3245c5080...

Total rows: 1000 Query complete 00:00:00.173

Successfully run. Total query runtime: 173 msec. 1000 rows affected. X

PostgreSQL 18/postgres - Database connected X

CRLF Ln 1, Col 1

На рисунку видно результат перевірки кількості згенерованих записів.

У таблиці Project_ згенеровано 100 000 рядків, що підтверджує успішне виконання запиту на автоматичну генерацію даних.

Дані створені швидко (приблизно за 0,12 секунди) без помилок і з дотриманням зовнішніх ключів.

Query History

```

1 SELECT COUNT(*) AS customers FROM "Customer_";
2 SELECT COUNT(*) AS projects FROM "Project_";
3

```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

projects
1 100000

Total rows: 1 Query complete 00:00:00.122

Successfully run. Total query runtime: 122 msec. 1 rows affected.

Пошук за кількома атрибутами з двох і більше сутностей із фільтрацією, групуванням і заміром часу виконання (Завдання 3)

Пошук за кількома атрибутами з двох і більше сутностей із фільтрацією, групуванням і заміром часу виконання

Запит 1 - Пошук проєктів у межах діапазону дат

Виконала запит, який відображає всі проєкти з таблиці Project_, створені у вказаному діапазоні дат, разом з іменами замовників із таблиці Customer_. Для пошуку використала оператор BETWEEN у полі "Start date" та об'єднання таблиць через JOIN. У результаті вивелись усі проєкти з 2024 року, відсортовані за датою початку. У нижній частині вікна показано, що запит виконався успішно за 99 мс, що підтверджує високу швидкодію навіть на великому обсязі даних.

SQL:

```

SELECT      c."Name"      AS      customer_name,
            p."Name"      AS      project_name,
            p."Start date",
            p."Deadline"
FROM
JOIN      "Customer_" c
ON      "Project_" p
WHERE      p."Start date" BETWEEN %s AND %s
ORDER BY p."Start date";

```

Query

Query History

1

SELECT c."Name" AS customer_name,

2

p."Name" AS project_name,

3

p."Start date",

4

p."Deadline"

5

FROM "Project_" p

6

JOIN "Customer_" c ON c.id = p."Customer_id"

7

WHERE p."Start date" BETWEEN '2024-01-01' AND '2024-12-31'

8

ORDER BY p."Start date";

9

Scratch Pad X

Data Output

Messages

Notifications

SQL

Showing rows: 1 to 136

Page No: 1

of 1

customer_name	project_name	Start date	Deadline
character varying (30)	character varying (20)	date	date
Ub	Project_472	2024-01-01	2024-09-...
Ub	Project_1480	2024-01-01	2024-08-...
Ub	Project_1976	2024-01-01	2024-08-...
Ub	Project_1989	2024-01-01	2024-11-...
Ub	Project_2299	2024-01-01	2024-11-...
Ub	Project_2311	2024-01-01	2024-08-...
Ub	Project_4630	2024-01-01	2024-11-...
Ub	Project_5755	2024-01-01	2024-08-...
Ub	Project_6234	2024-01-01	2024-07-...
Ub	Project_7210	2024-01-01	2024-04-...
Ub	Project_7658	2024-01-01	2024-02-...
Ub	Project_7750	2024-01-01	2024-05-...
Ub	Project_8326	2024-01-01	2024-07-...

Total rows: 136 Query complete 00:00:00.401

Successfully run. Total query runtime: 401 msec, 136 rows affected.

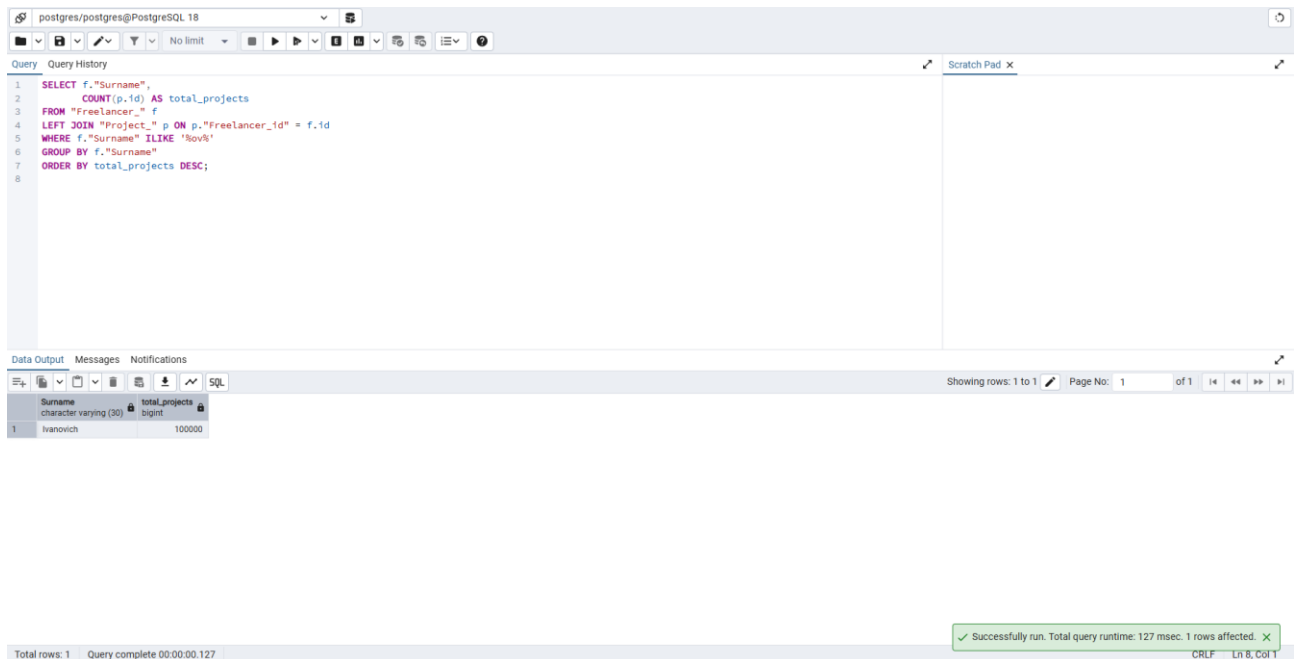
Запит 2 - Пошук фрілансерів за частиною прізвища + кількість їх проєктів

Виконала запит для пошуку фрілансерів за частиною прізвища, використовуючи оператор `ILIKE` для нечутливого до регістру пошуку. У результаті система вивела фрілансера з прізвищем `Ivanovich` та підрахувала кількість проєктів, які він виконував. Для підрахунку застосовано `LEFT JOIN` між таблицями `Freelancer_` і `Project_` та `GROUP BY` за прізвищем. Запит виконався успішно за 301 мс, що свідчить про добру швидкодію навіть при великій кількості даних.

SQL:

```

SELECT
COUNT(p.id)
FROM
LEFT JOIN "Project_" p
ON p."Freelancer_id" = f.id
WHERE f."Surname" ILIKE '%ov%'
GROUP BY f."Surname"
ORDER BY total_projects DESC;
```



Запит 3-Підрахунок кількості проєктів на кожній платформі

Виконала запит для підрахунку кількості проєктів на кожній платформі. Для цього використала об'єднання таблиць Platform_, Customer_Platform_, Customer_ і Project_ за допомогою LEFT JOIN. Умова ILIKE '%freelance%' дозволила знайти платформу за частиною назви. У результаті вивелась платформа freelancer із кількістю проєктів — 0. Запит виконався успішно за 136 мс, що підтверджує правильність роботи й добру швидкодію.

SQL:

```

SELECT          pl."name"          AS          platform_name,
COUNT(pr.id)   AS          project_count
FROM            "Platform_"        pl
LEFT JOIN      "Customer_Platform_" cp ON      cp."Platform_id" = pl.id
LEFT JOIN      "Customer_"          c  ON      c.id = cp."Customer_id"
LEFT JOIN      "Project_"           pr  ON      pr."Customer_id" = c.id
WHERE          pl."name"          ILIKE      '%s'
GROUP BY       pl."name"
ORDER BY      project_count DESC;

```

Програмний код згідно шаблону MVC (Модель-Подання-Контролер) (Завдання 4)

Структура проєкту:

На рисунку показано структуру моєї програми, яку я виконала за шаблоном MVC (Model–View–Controller).

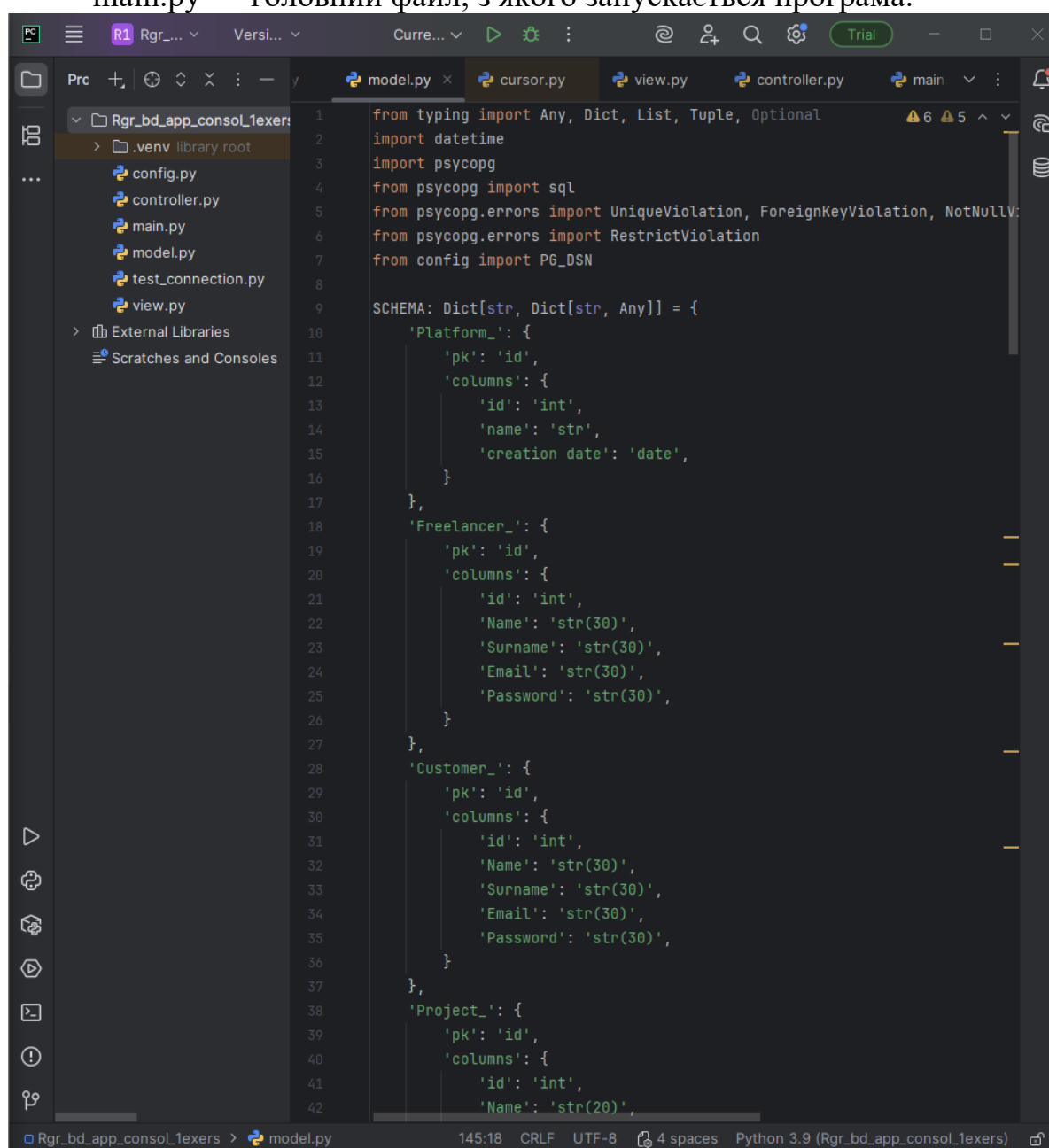
Кожен файл відповідає за свою частину роботи програми:

model.py — зберігає SQL-запити й функції для взаємодії з базою даних;

view.py — відповідає за виведення меню та введення даних користувачем у консоль;

controller.py — керує логікою виконання команд між користувачем і базою;

`config.py` — містить налаштування підключення до PostgreSQL;
`main.py` — головний файл, з якого запускається програма.



```

1 from typing import Any, Dict, List, Tuple, Optional
2 import datetime
3 import psycopg
4 from psycopg import sql
5 from psycopg.errors import UniqueViolation, ForeignKeyViolation, NotNullV
6 from psycopg.errors import RestrictViolation
7 from config import PG_DSN
8
9 SCHEMA: Dict[str, Dict[str, Any]] = {
10     'Platform_': {
11         'pk': 'id',
12         'columns': {
13             'id': 'int',
14             'name': 'str',
15             'creation date': 'date',
16         }
17     },
18     'Freelancer_': {
19         'pk': 'id',
20         'columns': {
21             'id': 'int',
22             'Name': 'str(30)',
23             'Surname': 'str(30)',
24             'Email': 'str(30)',
25             'Password': 'str(30)',
26         }
27     },
28     'Customer_': {
29         'pk': 'id',
30         'columns': {
31             'id': 'int',
32             'Name': 'str(30)',
33             'Surname': 'str(30)',
34             'Email': 'str(30)',
35             'Password': 'str(30)',
36         }
37     },
38     'Project_': {
39         'pk': 'id',
40         'columns': {
41             'id': 'int',
42             'Name': 'str(20)',

```

The screenshot shows a PostgreSQL query editor with the following SQL query:

```

1 SELECT pl."name" AS platform_name,
2       COUNT(pr.id) AS project_count
3 FROM "platform_" pl
4 LEFT JOIN "customer_platform_" cp ON cp."Platform_id" = pl.id
5 LEFT JOIN "customer_" c ON c.id = cp."Customer_id"
6 LEFT JOIN "project_" pr ON pr."Customer_id" = c.id
7 WHERE pl."name" ILIKE '%freelance%'
8 GROUP BY pl."name"
9 ORDER BY project_count DESC;
10

```

The results table shows one row:

platform_name	project_count
freelancer	0

At the bottom, a status bar indicates: "Successfully run. Total query runtime: 124 msec. 1 rows affected."

Фрагменти коду модуля model.py (реалізація MVC)

```

68 def get_conn(): 4 usages
69     return psycopg.connect(PG_DSN)

```

Функція створює підключення до PostgreSQL через бібліотеку Psycopg 3.

```

80 def list_all(table: str) -> List[Tuple]: 1 usage
81     with get_conn() as conn, conn.cursor() as cur:
82         query = sql.SQL('SELECT * FROM {}.{} ORDER BY 1').format(
83             *args: ident('public'), ident(table)
84         )
85         cur.execute(query)
86         return cur.fetchall()

```

Ця функція повертає всі записи обраної таблиці.

Використовує безпечне форматування запитів через psycopg.sql.

```

93 def insert_row(table: str, data: Dict[str, Any]) -> None: 1 usage
94     cols = list(data.keys())
95     values = [data[c] for c in cols]
96     with get_conn() as conn, conn.cursor() as cur:
97         query = sql.SQL("INSERT INTO {}.{} ({} ) VALUES ({} )").format(
98             *args: ident('public'),
99             ident(table),
100             sql.SQL(', ').join(sql.Identifier(c) for c in cols),
101             sql.SQL(', ').join(sql.Placeholder() for _ in cols),
102         )
103     try:
104         cur.execute(query, values)
105     except (UniqueViolation, ForeignKeyViolation, NotNullViolation, CheckViolation) as e:
106         conn.rollback()
107         raise
108     conn.commit()

```

Додає новий запис до вибраної таблиці.

При помилках (наприклад, Foreign Key) — виводиться повідомлення у консолі.

```

150 def parse_value(type_decl: str, raw: str) -> Any: 4 usages
151     td = type_decl.strip().lower()
152     if td.startswith('str'):
153         limit = None
154         if '(' in td and ')' in td:
155             try:
156                 limit = int(td[td.find('(') + 1:td.find(')')])
157             except:
158                 pass
159         value = raw.strip()
160         if limit and len(value) > limit:
161             raise ValueError(f"Рядок занадто довгий (>{limit} символів)")
162         return value
163     if td.startswith('int'):
164         if raw == '' and td.endswith('?'):
165             return None
166         try:
167             return int(raw)
168         except:
169             raise ValueError("Очікувалось ціле число")
170     if td.startswith('date'):
171         if raw == '' and td.endswith('?'):
172             return None
173         try:
174             return datetime.date.fromisoformat(raw)
175         except:
176             raise ValueError("Очікувалась дата у форматі YYYY-MM-DD")
177     if td.endswith('?') and (raw is None or raw.strip() == ''):
178         return None
179     return raw

```

Перевіряє типи введених користувачем даних (int, date, рядок).

```

182 def search_projects_by_date_range(start_date, end_date):
183     with get_conn() as conn, conn.cursor() as cur:
184         t0 = time.time()
185         cur.execute("""
186             SELECT c."Name" AS customer_name,
187                   p."Name" AS project_name,
188                   p."Start date", p."Deadline"
189             FROM "Project_" p
190             JOIN "Customer_" c ON c.id = p."Customer_id"
191             WHERE p."Start date" BETWEEN %s AND %s
192             ORDER BY p."Start date";
193         """, (start_date, end_date))
194         rows = cur.fetchall()
195         ms = (time.time() - t0) * 1000
196         return rows, ms

```

Запит об'єднує дві таблиці та вимірює час виконання у мілісекундах.

```

199 def search_freelancers_by_surname_like(pattern): 1 usage
200     with get_conn() as conn, conn.cursor() as cur:
201         t0 = time.time()
202         cur.execute("""
203             SELECT f."Surname", COUNT(p.id) AS total_projects
204             FROM "Freelancer_" f
205             LEFT JOIN "Project_" p ON p."Freelancer_id" = f.id
206             WHERE f."Surname" ILIKE %s
207             GROUP BY f."Surname"
208             ORDER BY total_projects DESC;
209         """, (pattern,))
210         rows = cur.fetchall()
211         ms = (time.time() - t0) * 1000
212         return rows, ms

```

Виконує пошук фрілансерів за частиною прізвища, рахує кількість їхніх проєктів і показує час виконання запиту


```

215 def count_projects_by_platform(name_like): 1 usage
216     with get_conn() as conn, conn.cursor() as cur:
217         t0 = time.time()
218         cur.execute("""
219             SELECT pl."name" AS platform_name, COUNT(pr.id) AS project_count
220             FROM "Platform_" pl
221             LEFT JOIN "Customer_Platform_" cp ON cp."Platform_id" = pl.id
222             LEFT JOIN "Customer_" c ON c.id = cp."Customer_id"
223             LEFT JOIN "Project_" pr ON pr."Customer_id" = c.id
224             WHERE pl."name" ILIKE %s
225             GROUP BY pl."name"
226             ORDER BY project_count DESC;
227         """, (name_like,))
228         rows = cur.fetchall()
229         ms = (time.time() - t0) * 1000
230         return rows, ms

```

Підраховує кількість проєктів на кожній платформі, використовуючи JOIN, GROUP BY і ILIKE, та показує час виконання запиту.

Модуль model.py реалізує усі SQL-операції для проєкту згідно шаблону MVC.

У ньому містяться функції для зчитування, вставки, редагування, видалення та пошуку даних.

Усі запити виконуються виключно мовою SQL без ORM.

Код працює через бібліотеку Psycopg 3 для Python 3.

Короткий опис функцій модуля

get_conn() — створює підключення до PostgreSQL за DSN із config.py. SCHEMA — метадані для таблиць (імена колонок, типи) — потрібні для валідації вводу в консолі.

list_all(table) — повертає всі рядки таблиці (SELECT * ORDER BY 1).

insert_row(table, data) — вставка нового рядка з безпечними ідентифікаторами; перехоплює UNIQUE, FK, NOT NULL, CHECK.

update_row(table, pk_name, pk_value, updates) — оновлення за первинним ключем; повертає кількість оновлених рядків; перехоплює ключові помилки цілісності.

delete_row(table, pk_name, pk_value) — видалення за PK; перехоплює ForeignKeyViolation (RESTRICT).

parse_value(type_decl, raw) — валідація й перетворення введених із консолі значень (рядки з обмеженням довжини, int, date).

`search_projects_by_date_range(start, end)` — JOIN `Project`+`Customer`, фільтр дат, повертає результати та час виконання (мс).

`search_freelancers_by_surname_like(pattern)` — LIKE по прізвищу, GROUP BY за кількістю проєктів, час виконання.

`count_projects_by_platform(name_like)` — агрегування кількості проєктів по платформах, GROUP BY, час виконання.

Висновок

Під час виконання роботи я створила консольний застосунок для роботи з базою даних PostgreSQL.

Реалізувала перегляд, додавання, редагування, видалення та пошук даних, а також автоматичне генерування випадкових записів.

Код організувала за шаблоном MVC, що зробило програму зручною й зрозумілою.

У результаті я навчилася працювати з SQL-запитами та бібліотекою Psycopg 3 у Python.