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 вилучення 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.

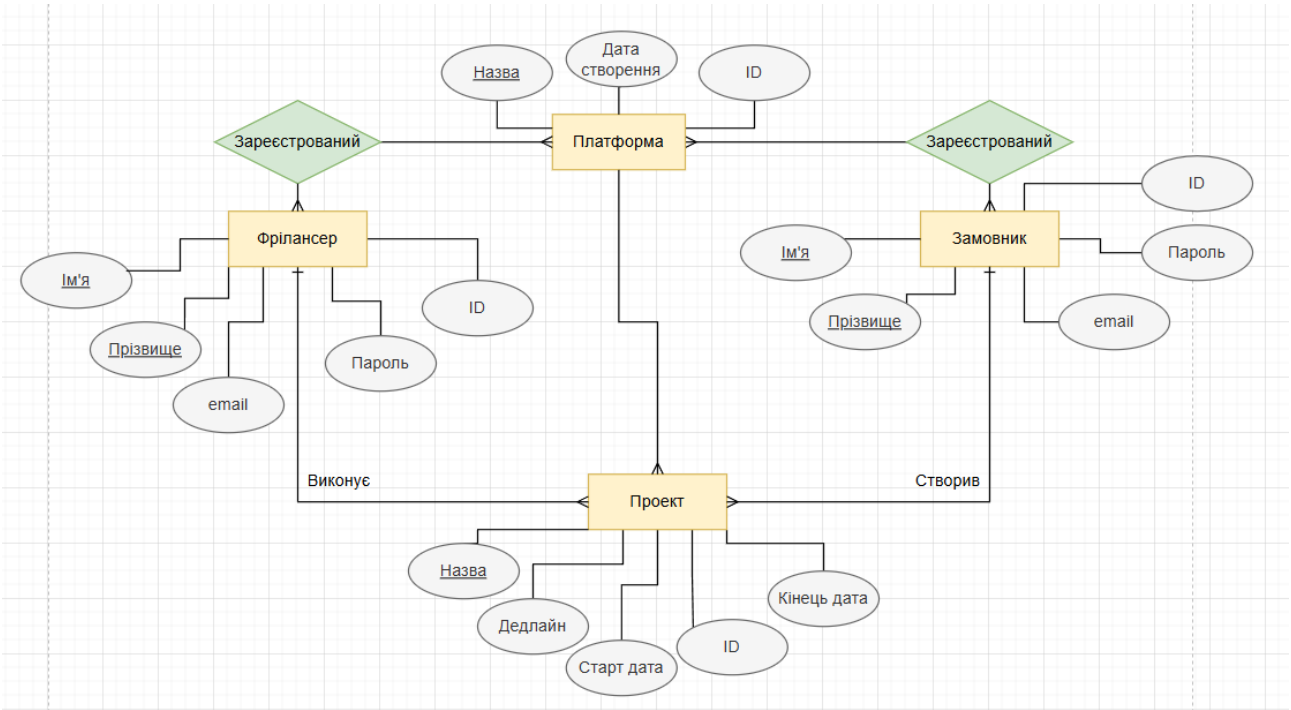
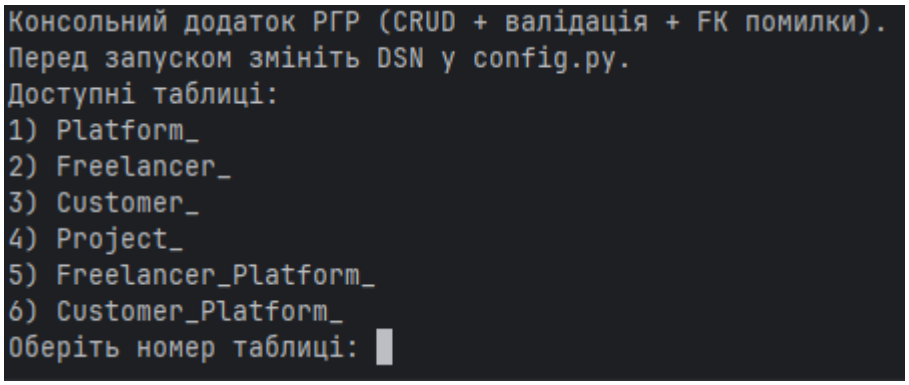


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

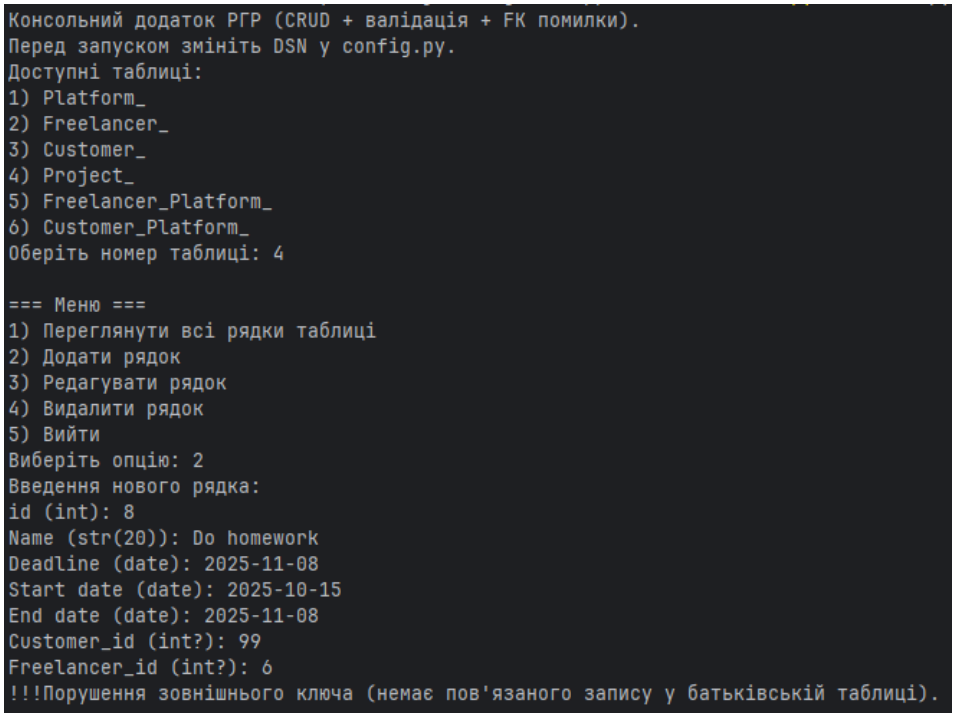
Реалізування функцій засобами консольного інтерфейсу (Завдання 1)

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.

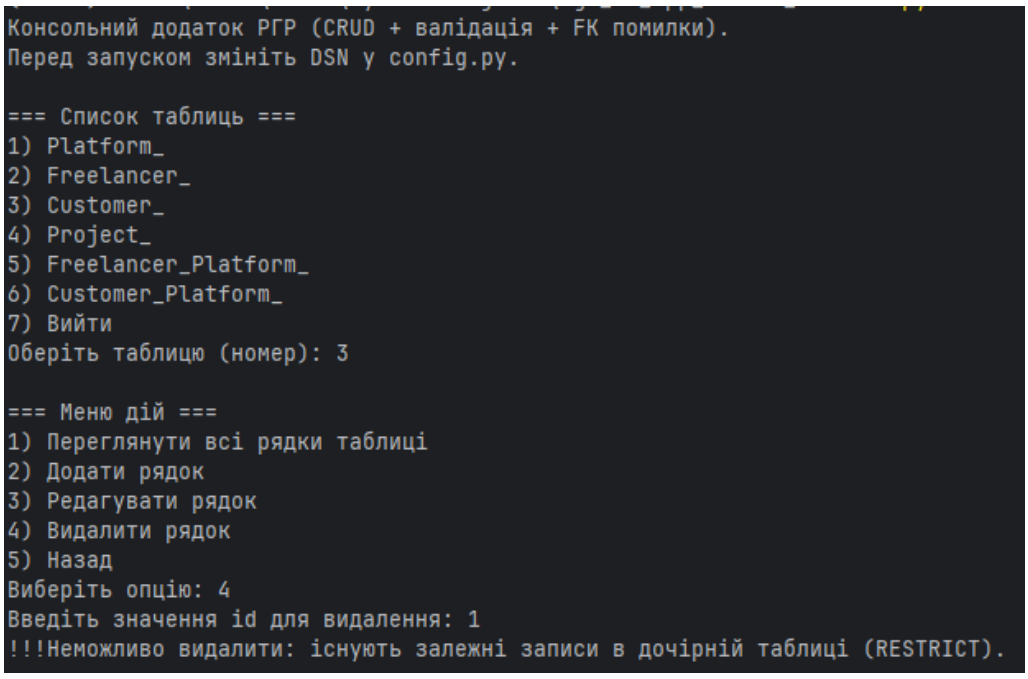


Пробувала додати новий проєкт із замовником, якого немає в таблиці. Програма показала повідомлення про помилку зовнішнього ключа. Це означає, що не можна додати проєкт без існуючого замовника.



Вставка у дочірню таблицю при відсутності батьківсього 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.



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)
```

```
=== 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)


=== 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): Ponchik
creation date (date): 2025-11-08
+Додано.
```

```
=== 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
+Видалено.

=== 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)
```

Автоматичне пакетне генерування “рандомізованих” даних у базі (Завдання 2)

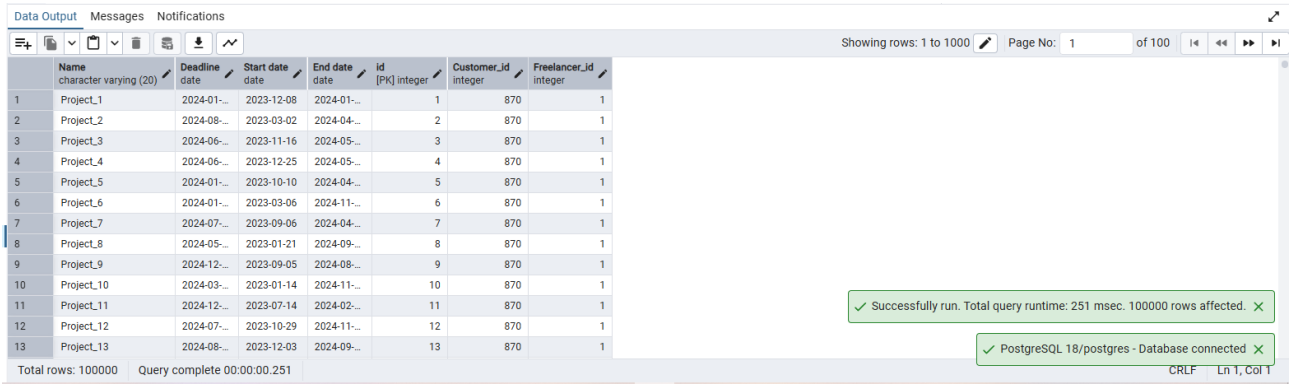
Створила SQL-запити з функціями random(), generate_series(), chr(), md5(), які автоматично заповнюють таблиці Customer_ і Project_. Дані генеруються в межах коректних значень, дати — у випадкових діапазонах, а поля Customer_id і Freelancer_id беруться зі вже існуючих таблиць, тому порушень зовнішніх ключів не виникає.

Таблиця Project_:

SQL запит:

```
INSERT INTO "Project_" ("Name", "Deadline", "Start date", "End date", "Customer_id", "Freelancer_id")
SELECT 'Project_' || gs,
    date '2024-01-01' + (random() * 365)::int,
    date '2023-01-01' + (random() * 365)::int,
    date '2024-01-01' + (random() * 365)::int,
    (SELECT id FROM "Customer_" ORDER BY random() LIMIT 1),
```

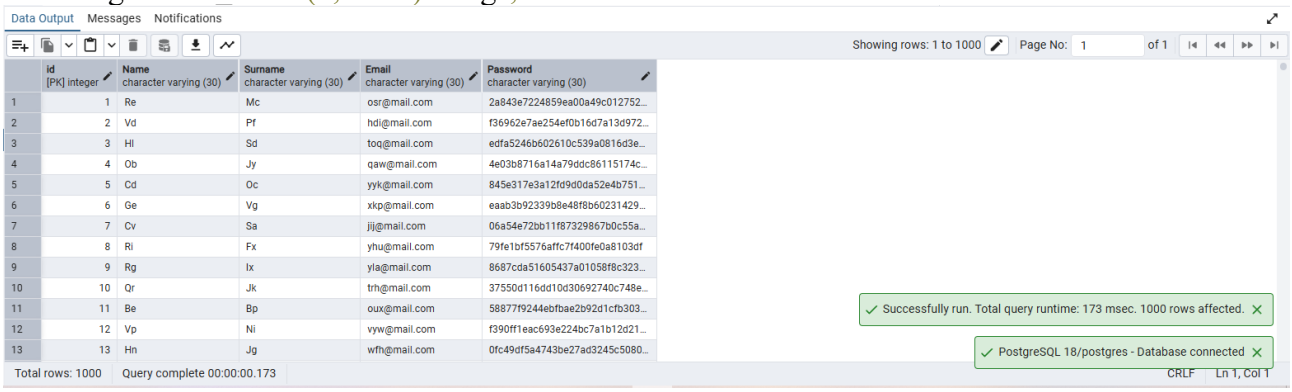
```
    (SELECT id FROM "Freelancer_" ORDER BY random() LIMIT 1)
FROM generate_series(1, 100000) AS gs;
```



Таблиця Customer_:

SQL запит:

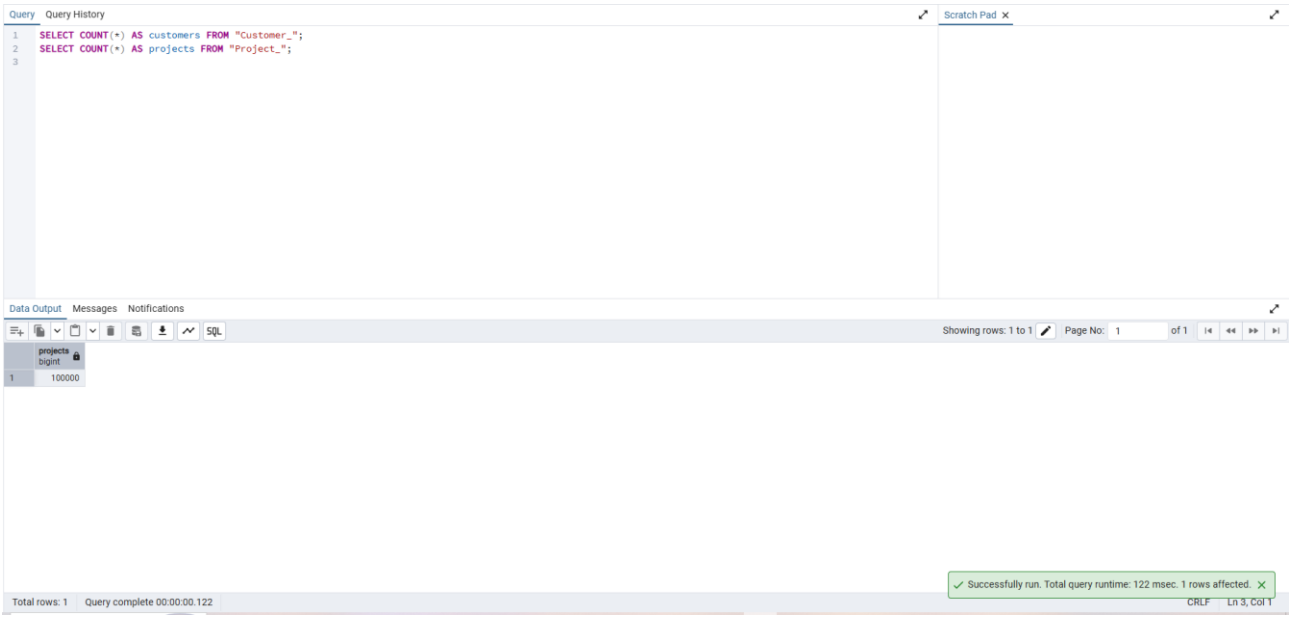
```
    INSERT INTO "Customer_" (id, "Name", "Surname", "Email", "Password")
SELECT gs AS id,
    initcap(chr(trunc(65 + random() * 25)::int)
        || chr(trunc(65 + random() * 25)::int)) AS "Name",
    initcap(chr(trunc(65 + random() * 25)::int)
        || chr(trunc(65 + random() * 25)::int)) AS "Surname",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



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.



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

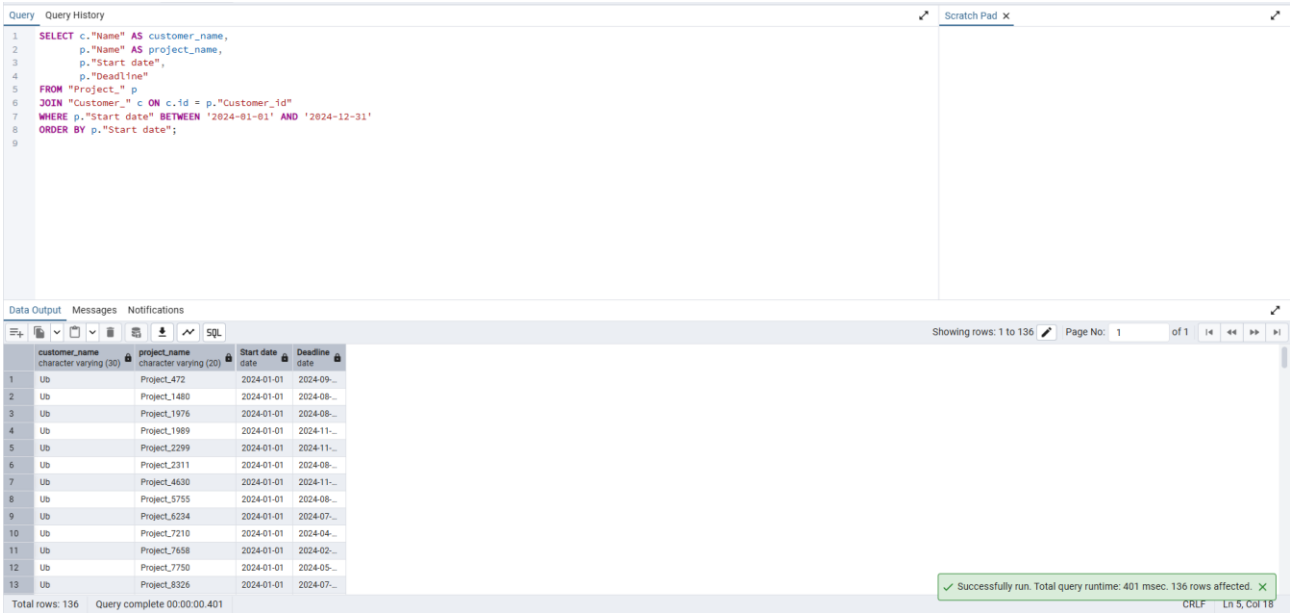
Пошук за кількома атрибутами з двох і більше сутностей із фільтрацією, групуванням і заміром часу виконання

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 "Project_" p
JOIN "Customer_" c ON c.id = p."Customer_id"
WHERE p."Start date" BETWEEN %s AND %s
ORDER BY p."Start date";
```

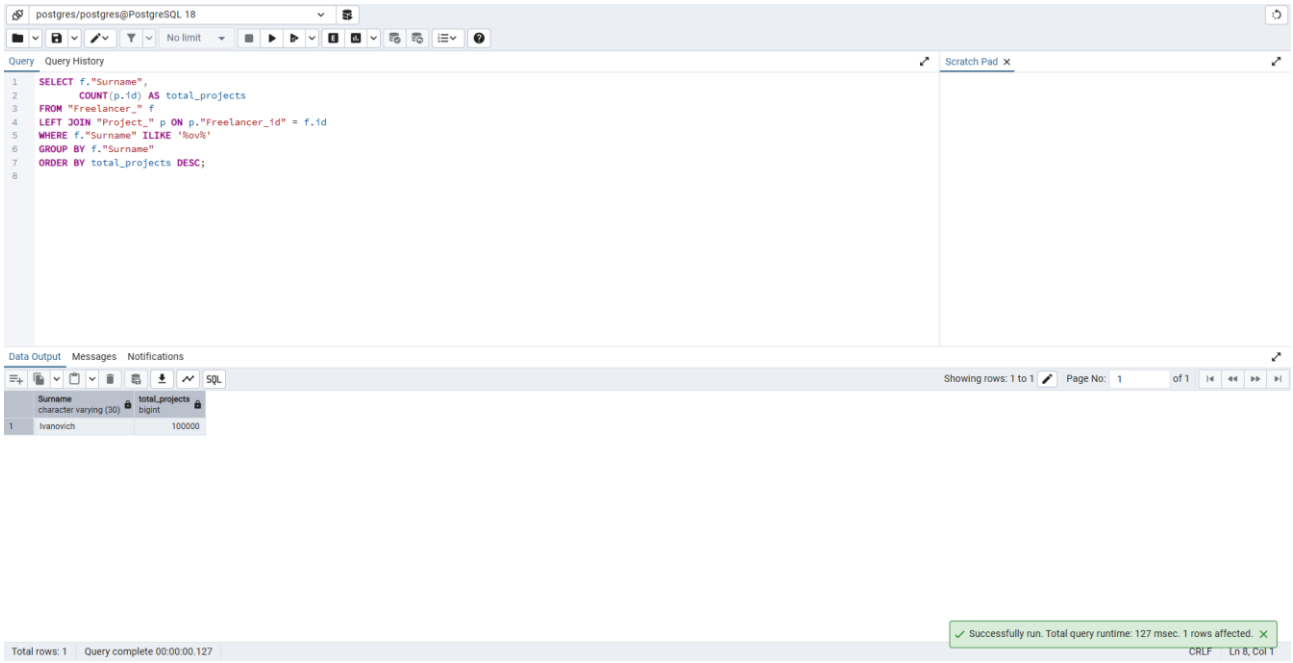


Запит 2 - Пошук фрілансерів за частиною прізвища + кількість їх проєктів

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 f."Surname", COUNT(p.id) AS total_projects
FROM "Freelancer_" f
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).

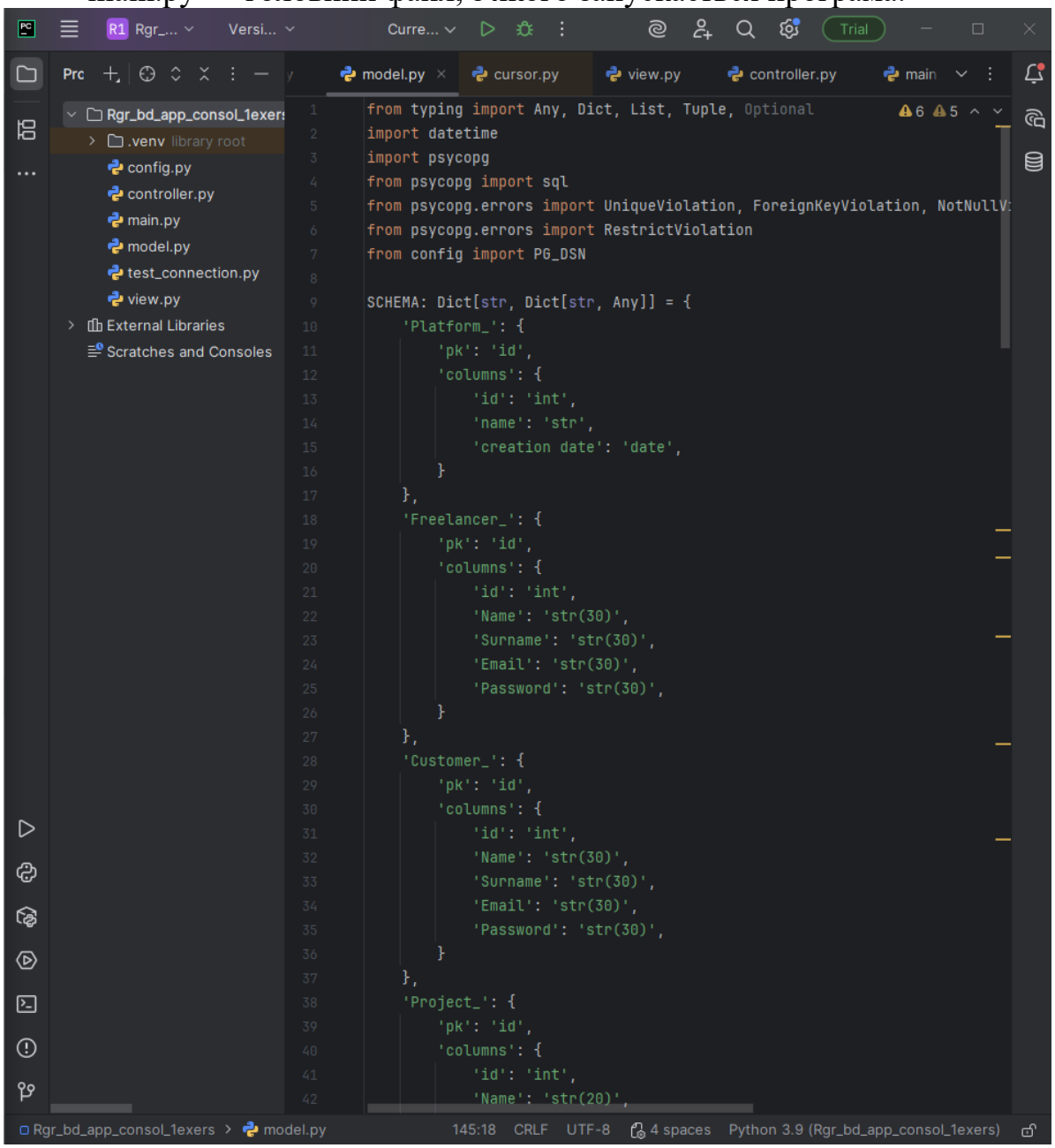
Кожен файл відповідає за свою частину роботи програми:

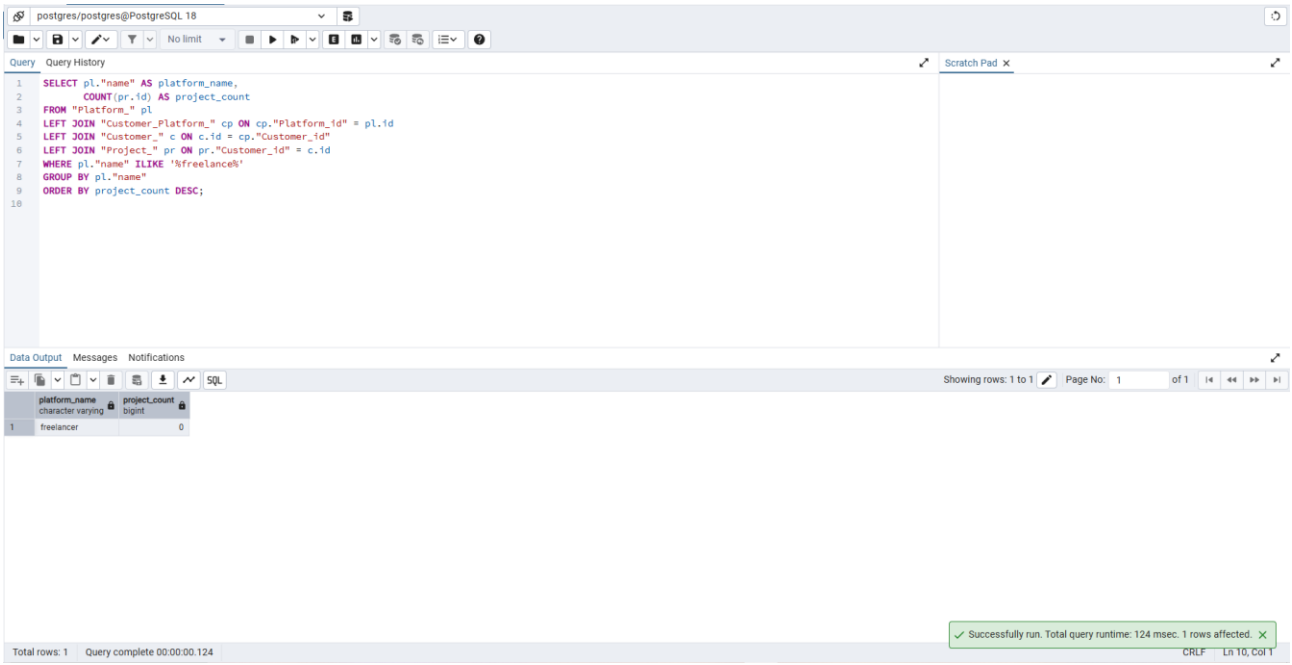
model.py — зберігає SQL-запити й функції для взаємодії з базою даних;

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.





Фрагменти коду модуля model.py (реалізація MVC)

```python
def get_conn():  4 usages
    return psycopg.connect(PG_DSN)
```

Функція створює підключення до PostgreSQL через бібліотеку Psycopg 3.

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

Ця функція повертає всі записи обраної таблиці.
Використовує безпечне форматування запитів через psycopg.sql.

```python
def insert_row(table: str, data: Dict[str, Any]) -> None:  1 usage
    cols = list(data.keys())
    values = [data[c] for c in cols]
    with get_conn() as conn, conn.cursor() as cur:
        query = sql.SQL("INSERT INTO {}.{} ({}) VALUES ({})").format(
            *args: ident('public'),
            ident(table),
            sql.SQL(', ').join(sql.Identifier(c) for c in cols),
            sql.SQL(', ').join(sql.Placeholder() for _ in cols),
        )
        try:
            cur.execute(query, values)
        except (UniqueViolation, ForeignKeyViolation, NotNullViolation, CheckViolation) as e:
            conn.rollback()
            raise
        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).

```python
def search_projects_by_date_range(start_date, end_date):
    with get_conn() as conn, conn.cursor() as cur:
        t0 = time.time()
        cur.execute("""
            SELECT c."Name" AS customer_name,
                   p."Name" AS project_name,
                   p."Start date", p."Deadline"
            FROM "Project_" p
            JOIN "Customer_" c ON c.id = p."Customer_id"
            WHERE p."Start date" BETWEEN %s AND %s
            ORDER BY p."Start date";
        """, (start_date, end_date))
        rows = cur.fetchall()
        ms = (time.time() - t0) * 1000
        return rows, ms
```

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

```python
def search_freelancers_by_surname_like(pattern):  1 usage
    with get_conn() as conn, conn.cursor() as cur:
        t0 = time.time()
        cur.execute("""
            SELECT f."Surname", COUNT(p.id) AS total_projects
            FROM "Freelancer_" f
            LEFT JOIN "Project_" p ON p."Freelancer_id" = f.id
            WHERE f."Surname" ILIKE %s
            GROUP BY f."Surname"
            ORDER BY total_projects DESC;
        """, (pattern,))
        rows = cur.fetchall()
        ms = (time.time() - t0) * 1000
        return rows, ms
```

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

```python
def count_projects_by_platform(name_like):  1 usage
    with get_conn() as conn, conn.cursor() as cur:
        t0 = time.time()
        cur.execute("""
            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;
        """, (name_like,))
        rows = cur.fetchall()
        ms = (time.time() - t0) * 1000
        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.