НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахункова графічна робота

з дисципліни

«Бази даних і засоби управління»

на тему «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-31

Шило Максим Сергійович

Перевірив: Петрашенко А.В.

Київ – 2025

Предметна галузь: Система обліку та аналізу даних в галузі астрономії

GitHub Посилання на репозиторій із проектом
https://github.com/MaxShlLo/BD/tree/main

Створення ER-діаграми

Для баз даних «Система обліку та аналізу даних в галузі астрономії» я виділив наступні сутності: Дослідник, Об’єкт, та Лабораторія.

Визначено між сутностями такі зв’язки.

- Багато дослідників знаходяться тільки в одній лабораторії. (1:N)
- Одна лабораторія може вивчати кілька об’єктів. (1:N)

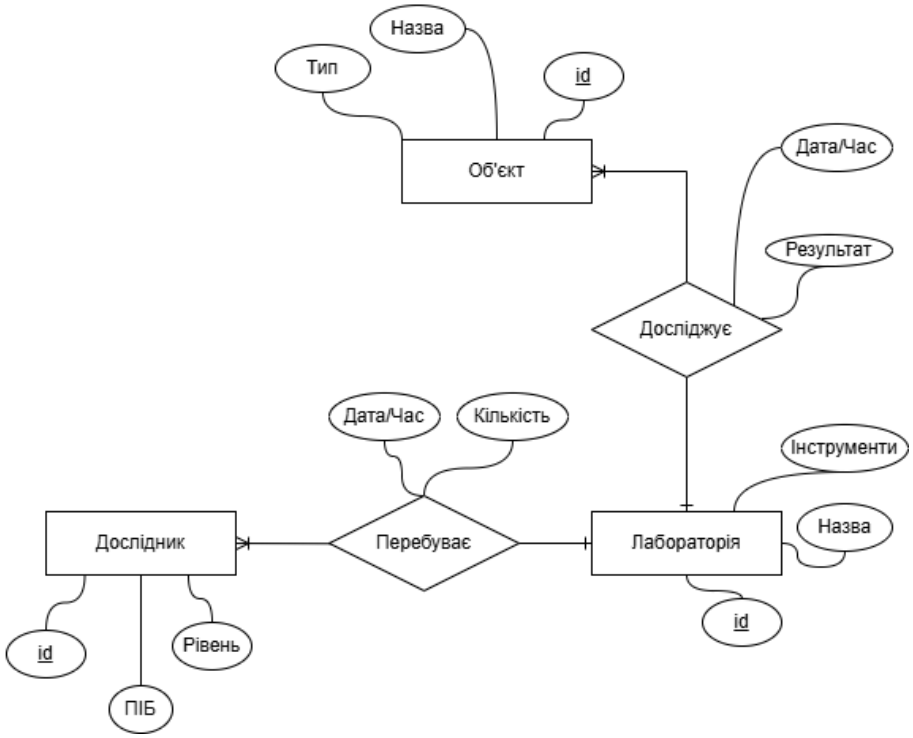
Таблиця сутностей з описом призначення:

Сутність	Атрибут	Тип
Дослідник: інформація про дослідника астрономічних об’єктів	ПІБ: прізвище, ім’я та по батькові дослідника Рівень: кваліфікація дослідника	Текст (50) Текст (50)
Об’єкт: інформація космічного об’єкта, який досліджують	Назва: відповідна назва об’єкта Тип: тип космічного об’єкта	Текст (50) Текст (50)
Лабораторія: інформація лабораторії де проводяться дослідження	Назва: назва місце проведення дослідження Інструменти: пристрій що використовуються для дослідження	Текст (100) Текст (50)

ER-діаграма:

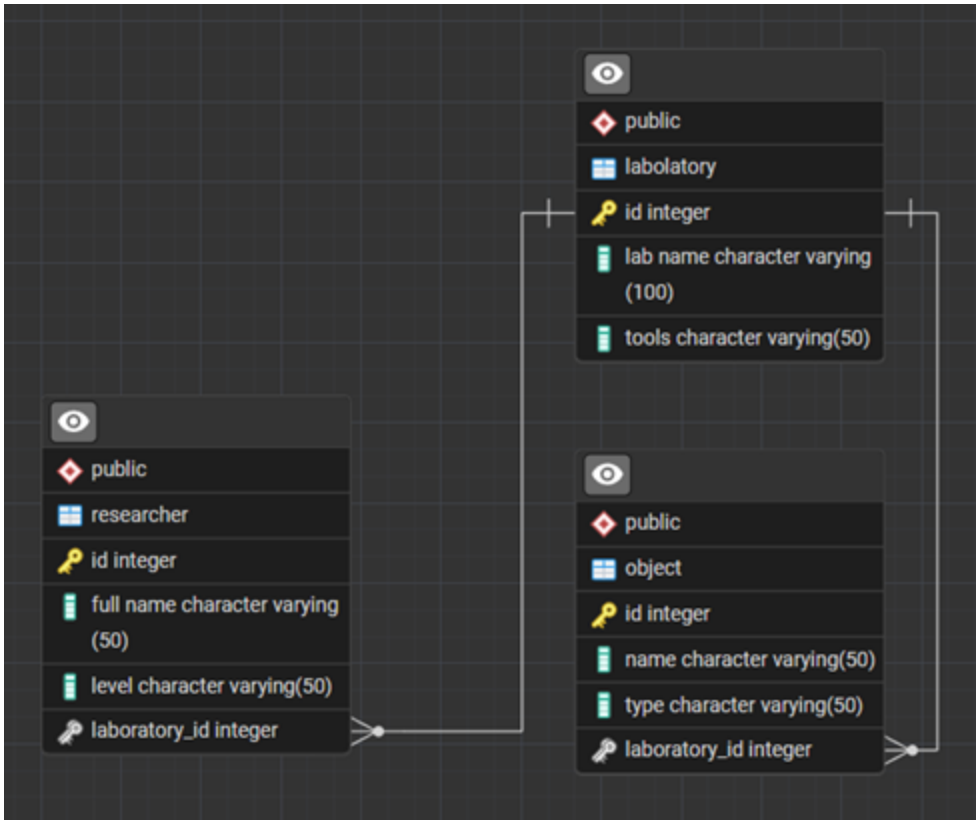
Система обліку та аналізу даних в галузі астрономії

Об'єкт
Дослідник
Лаборторія



Опис процесу перетворення в БД

Сутності «Об’єкт», «Лабораторія» та «Дослідник» були перетворені в таблиці БД з такими назвами «object», «laboratory» та «researcher» відповідно. Тепер вони мають Prime Key із назвами id. Мають так само такі ж атрибути тільки з другими назвами та Foreign Key їхні назви такі laboratory_id в таблиці researcher, та laboratory_id в object.



В результаті вийшла така база даних. Перевіримо її на три перших нормальні форми.

Функціональні залежності:

Таблиця researcher
id -> full name, level

Таблиця laboratory
id -> lab name, tools
id -> lab name
id-> tools - тут помилка першої нормальної форми, бо можу бути список.

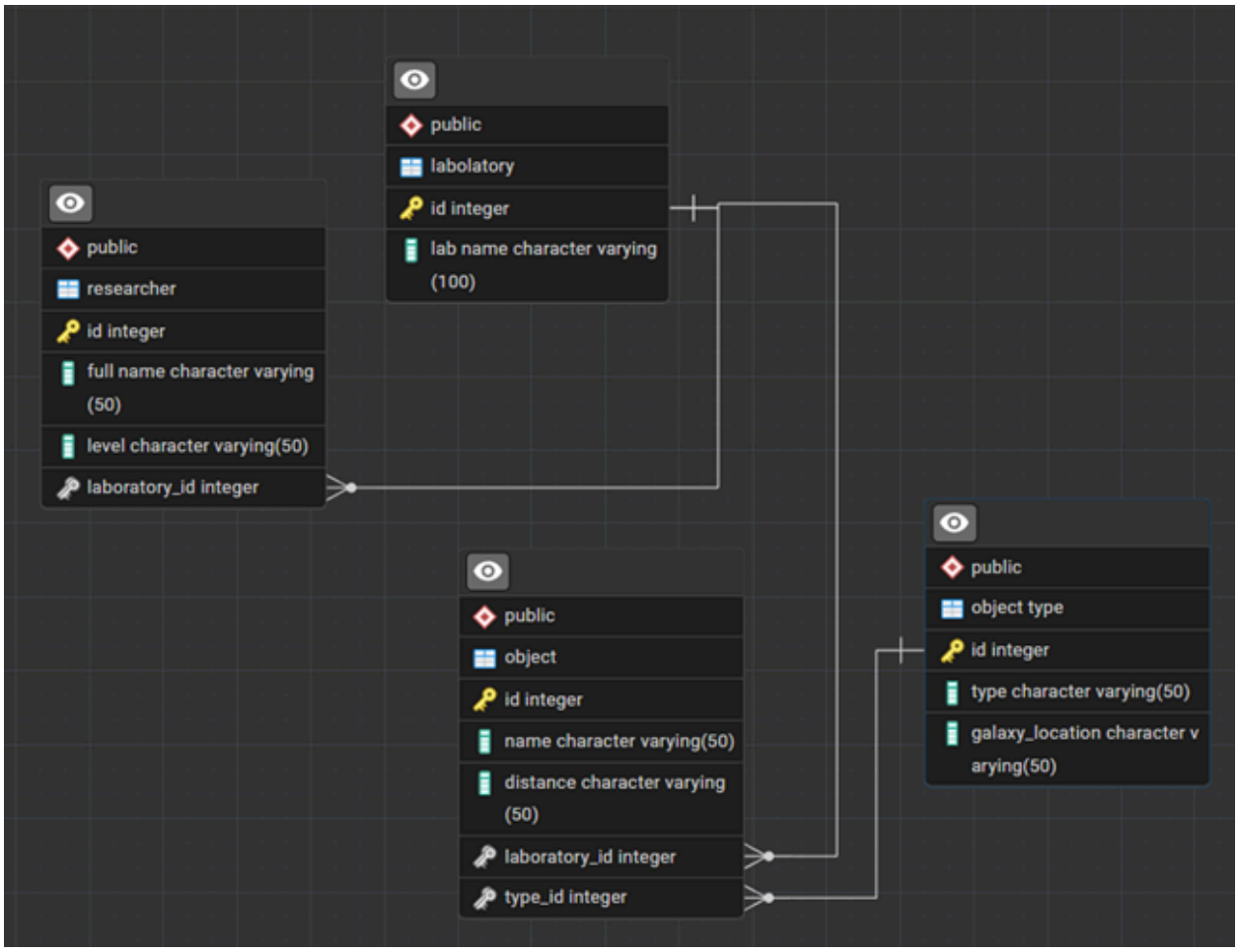
Таблиця object
id -> name, type
id -> name
id -> type

name -> type - може бути помилка третьої нормальної форми, через транзитивну залежність.

1) Перша нормальна форма - всі атрибути повинні бути **атомарними**. Це виконується у всіх таблицях, але є деякі проблеми в таблиці labolatory із атрибутом tools. Суть в тому, що може бути багато інструментів, тому це вже порушення ПНФ. Щоб це виправити є два виходи, перший складніший це зробити окрему таблицю для інструментів із зв'язком M:M, або простіший спосіб прибрати цей атрибут, багато інформації вона не дає, а проблем надагато більше. Тому з огляду простоти, було вибрано варіант прибрати цей атрибут.

2) Друга нормальна форма – повинно виконуватися перша форма і **кожен неключовий атрибут має залежати від усього первинного ключа**, а не від його частини. Це також виконується бо у всіх таблицях первинний ключ складається з одного атрибуту «id», а його розділити на частини не можливо.

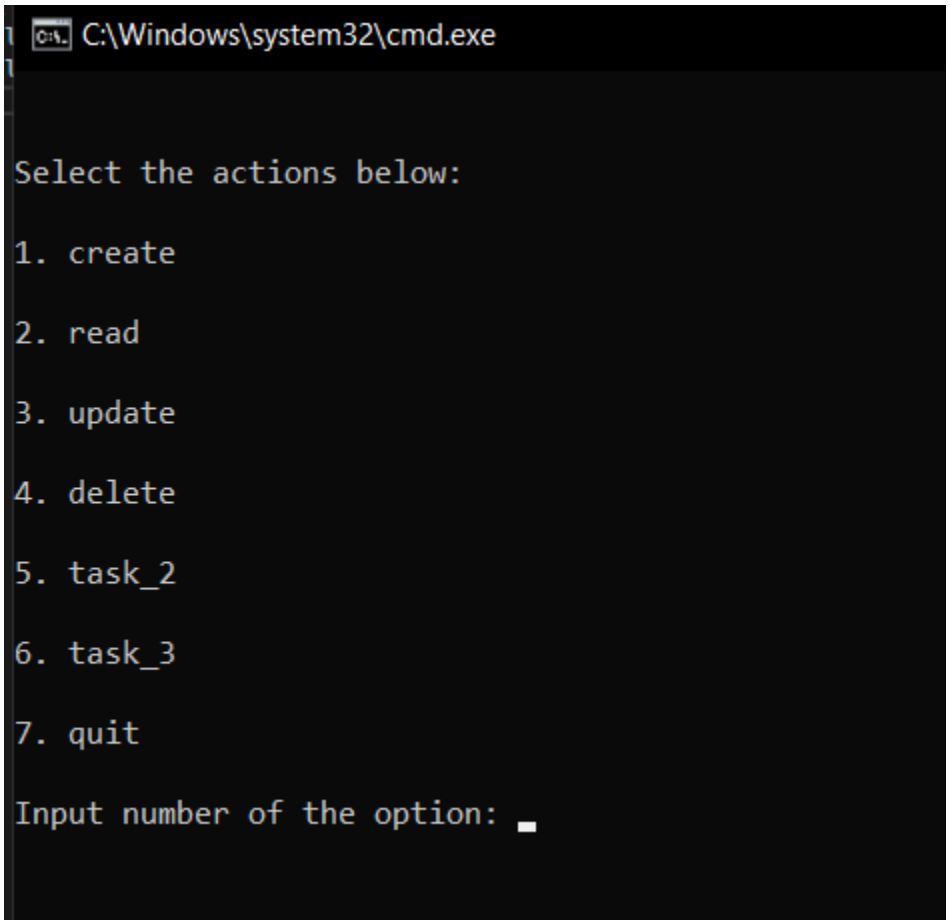
3) Третє нормальна форма – повинно виконуватися друга нормальна форма та жоден неключовий атрибут не повинен залежати від іншого неключового атрибуту. Тут у таблицях «researcher» та «laboratory» все добре, рівень не може визначити ПІБ дослідника і інструмент не зможе показати яка це лабораторія. А в таблиці «object» є невелика проблема. Зробимо уявний рядок у цій таблиці, де буде id, Сонце, Зоря і посилання на лабораторію. Тут зразу буду ясно що якщо це зоря то можна вже подумати що це Сонце. І так само навпаки, якщо вже бачу що це Сонце то ясно що це Зоря і це явне порушення 3НФ. Тому зробимо такі зміни в таблиці винесемо цю змінну в нову таблицю «object_type» де буде id, type та description, де буде невелике пояснення про об'єкт. В результаті вийде наступна база даних.



Тут змінено таблицю «object» де додано ще рядок про відстань до об’єкта від Землі і додано зовнішній ключ на тип об’єкта. Також прибрано атрибут tools із таблиці laboratory. Таким чином порушень 1НФ та 3НФ не буде.

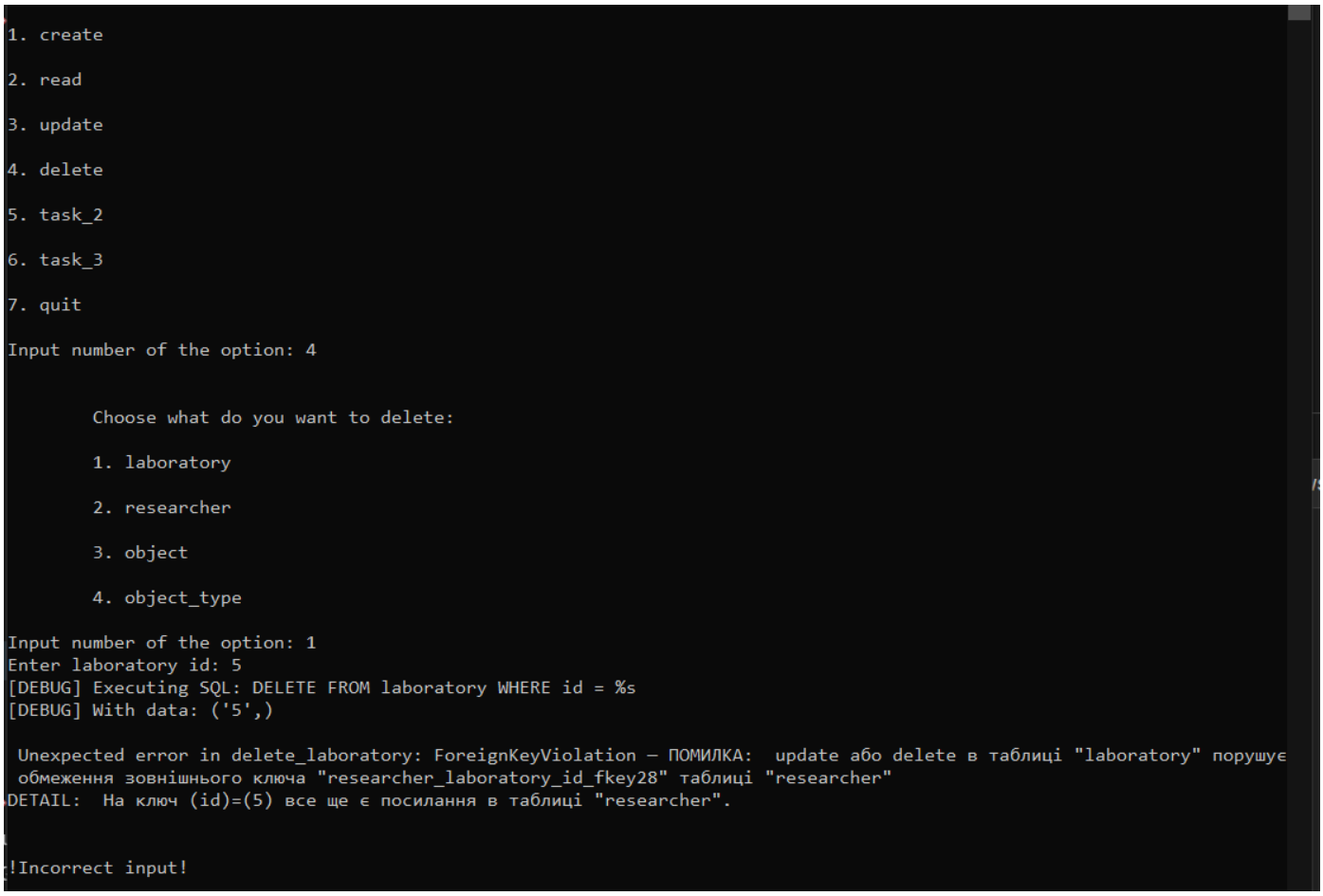
Тепер всі три форми є вірними.

Демонстрація роботи коду:
Показ меню дій які можна виконати.

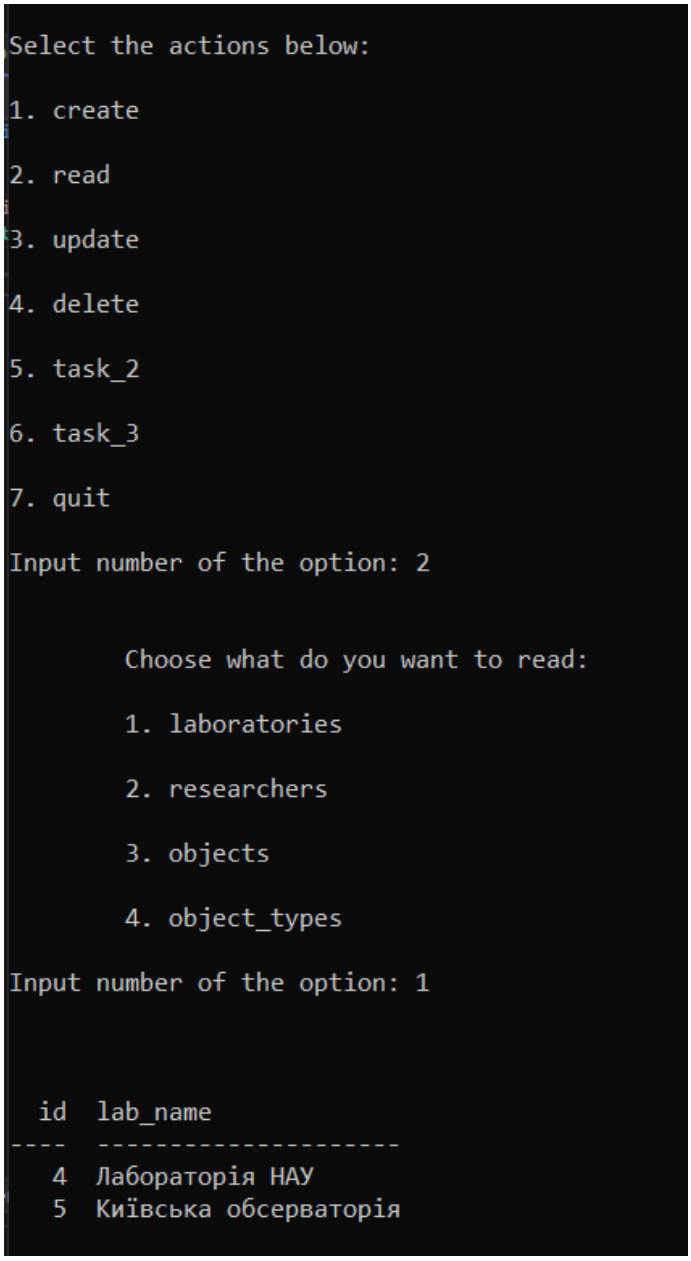


Для демонстрації наступних дій було створено даякі дані завдяки пункту create. Тепер спробуємо видалити щось що має зв’язок між таблицями. Наприклад є рядок у таблиці laboratory яка має зв’язок 1:N з researcher. При

спробі видалити видає помилку.



Це не можу бути виконано, через те що рядок що ми хочемо прибрати досі є в других таблицях і якщо ми його прибираємо то посилання других таблиць ні на що не вказує. На самому скрині видно, що виводиться інформація що помилка зв’язана саме з таблицею researcher, бо там є посилання на лабораторію із id = 5. Спробуємо оновити таблицю laboratory щоб замінити посилання з 5 на 4, щоб можна було видалити саме 5 лабораторію. Глянемо що зараз є в laboratory.



Тепер змінимо функцією update. Случайно обрав не те оновлення, але тут якраз показано що буде якщо вибрати оновлювати не існуюче поле. Помилки видавати не буде, але покаже що такого id не має в таблиці, тому нічого не

оновлено.

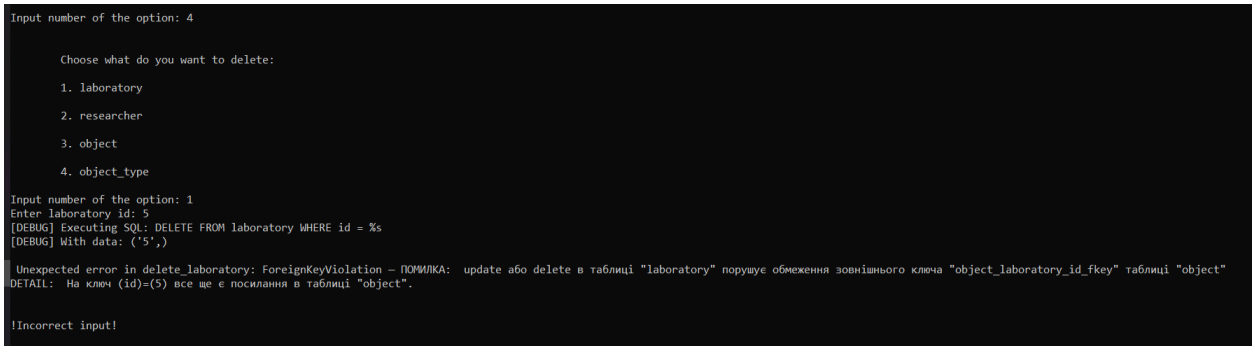
```
Select the actions below:

1. create

2. read

3. update

4. delete

5. task_2

6. task_3

7. quit

Input number of the option: 3


        Choose what do you want to update:

        1. laboratory

        2. researcher

        3. object

        4. object_type

Input number of the option: 1
Enter current id: 6
Enter new name: fg
[DEBUG] Executing SQL: UPDATE laboratory SET lab_name = %s WHERE id = %s
[DEBUG] With data: ('fg', '6')
[!INFO!] NO LABORATORY WITH ID=6 — nothing was updated.
```

Тепер оновимо що потрібно.

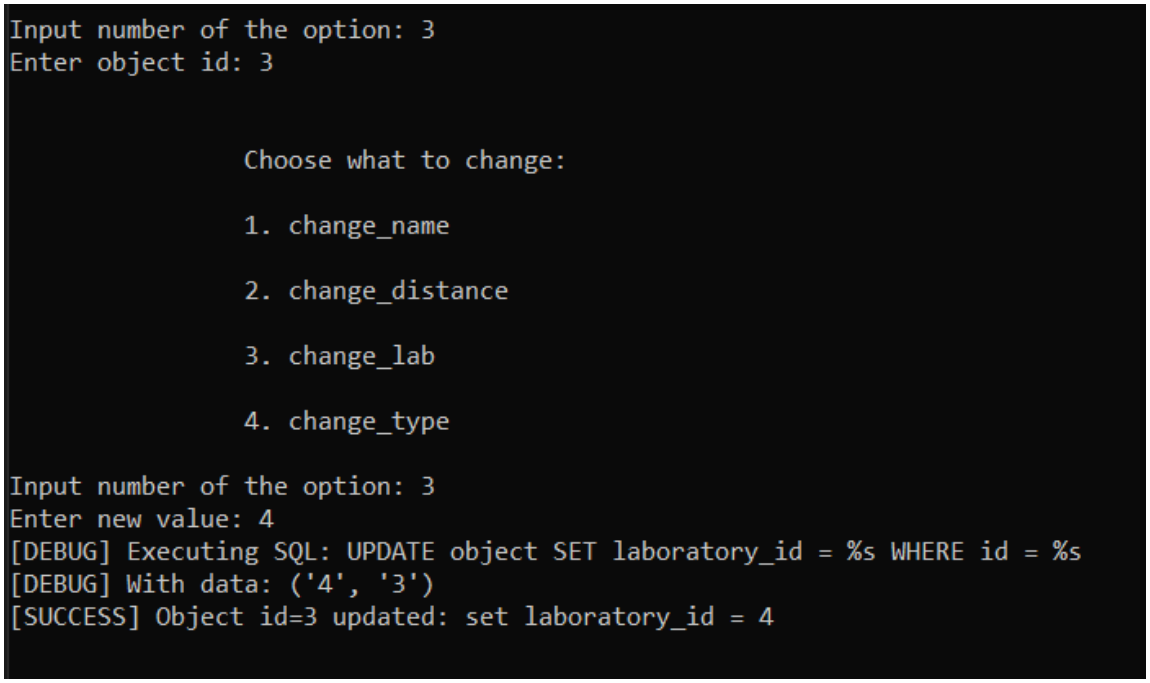
```
Input number of the option: 3


       Choose what do you want to update:

       1. laboratory

       2. researcher

       3. object

       4. object_type

Input number of the option: 2
Enter researcher id: 32


              Choose what to change:

              1. change_full_name

              2. change_level

              3. change_laboratory

Input number of the option: 3
Enter new value: 4
[DEBUG] Executing SQL: UPDATE researcher SET laboratory_id = %s WHERE id = %s
[DEBUG] With data: (4, '32')
[SUCCESS] Researcher id=32 updated: set laboratory_id = 4
```

```
Input number of the option: 3


       Choose what do you want to update:

       1. laboratory

       2. researcher

       3. object

       4. object_type

Input number of the option: 2
Enter researcher id: 31


              Choose what to change:

              1. change_full_name

              2. change_level

              3. change_laboratory

Input number of the option: 3
Enter new value: 4
[DEBUG] Executing SQL: UPDATE researcher SET laboratory_id = %s WHERE id = %s
[DEBUG] With data: (4, '31')
[SUCCESS] Researcher id=31 updated: set laboratory_id = 4
```

І спробуємо знову видалити цю лабораторію із id = 5.

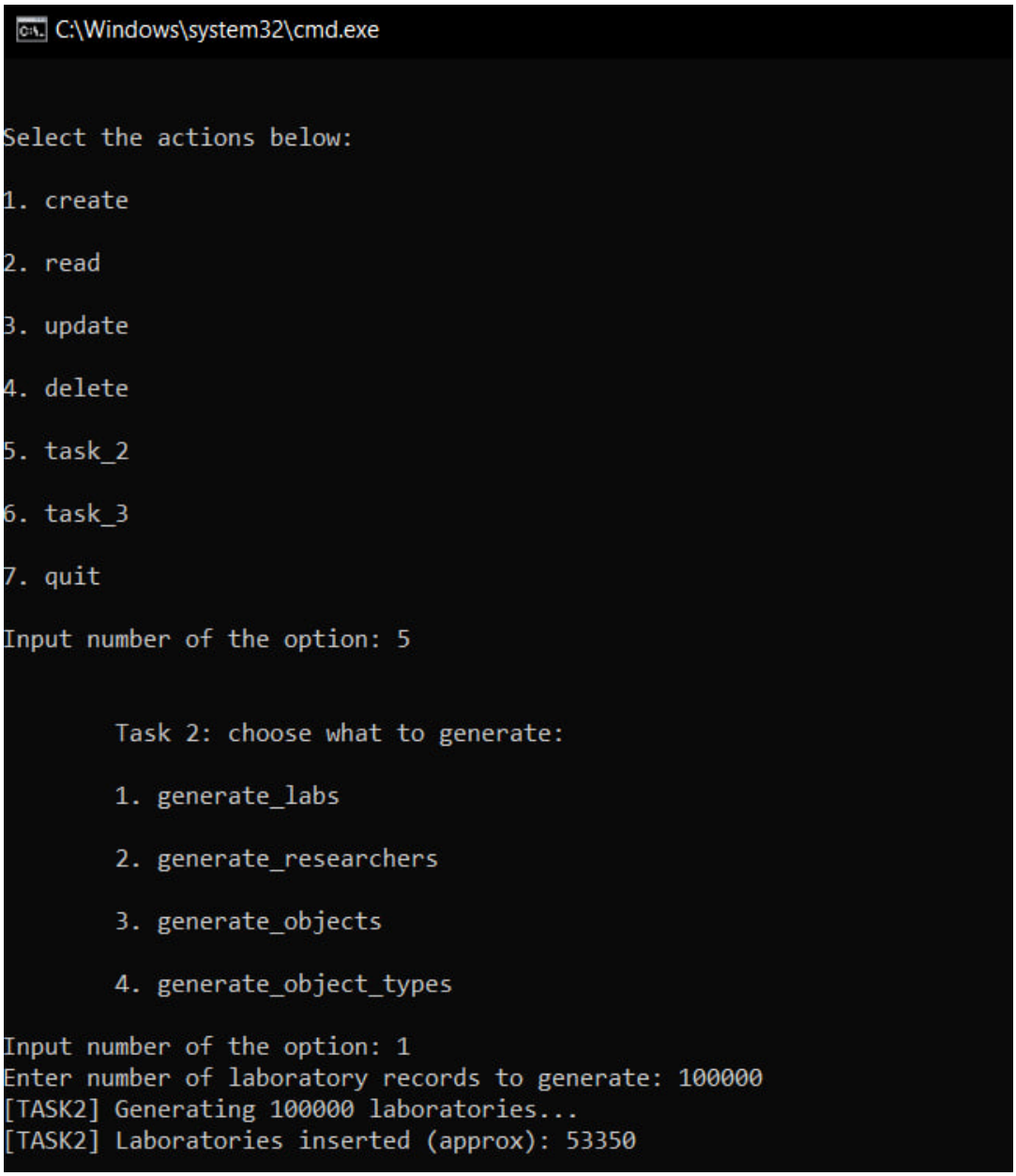


Знову видає помилку, тут причина в тому самому але зв’язана з другою таблицею. Тепер воно каже, що є ще посилання на цю лабораторію у таблиці object. Уже коли ми приберемо і там посилання на ту лабораторію то тоді воно зможе видалитися бо не буде посилань на нього.

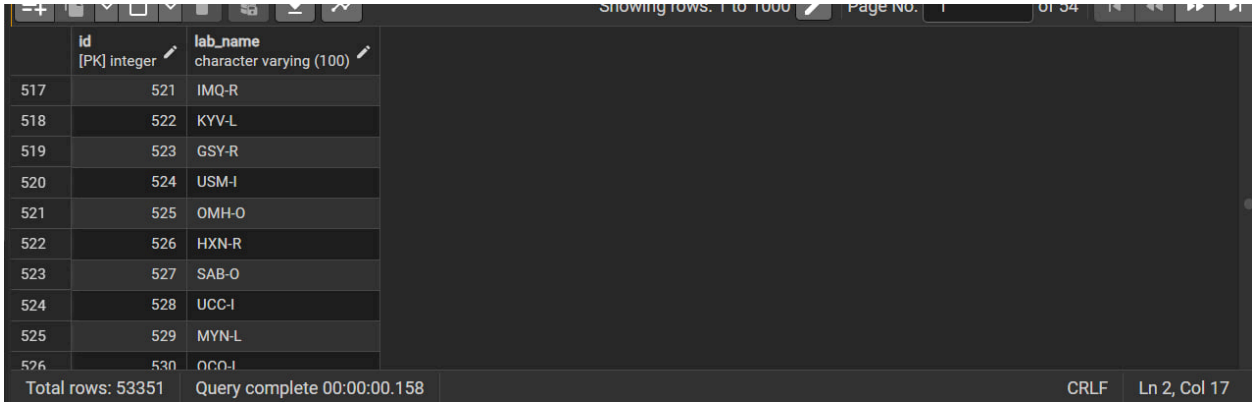


```
Input number of the option: 4


        Choose what do you want to delete:

        1. laboratory

        2. researcher

        3. object

        4. object_type

Input number of the option: 1
Enter laboratory id: 5
[DEBUG] Executing SQL: DELETE FROM laboratory WHERE id = %s
[DEBUG] With data: ('5',)
[SUCCESS] Laboratory id=5 deleted successfully.


Select the actions below:

1. create

2. read

3. update

4. delete

5. task_2

6. task_3

7. quit

Input number of the option: 2


        Choose what do you want to read:

        1. laboratories

        2. researchers

        3. objects

        4. object_types

Input number of the option: 1



  id  lab_name
----  ---------------
   4  Лабораторія НАУ
```

Тепер протестуємо функції для створення великих масивів даних. Першим ділом створимо дані в таблиці laboratory.



Для перевірки глянемо в саму программу і пересвідчився чи справді там з'явилися дані.



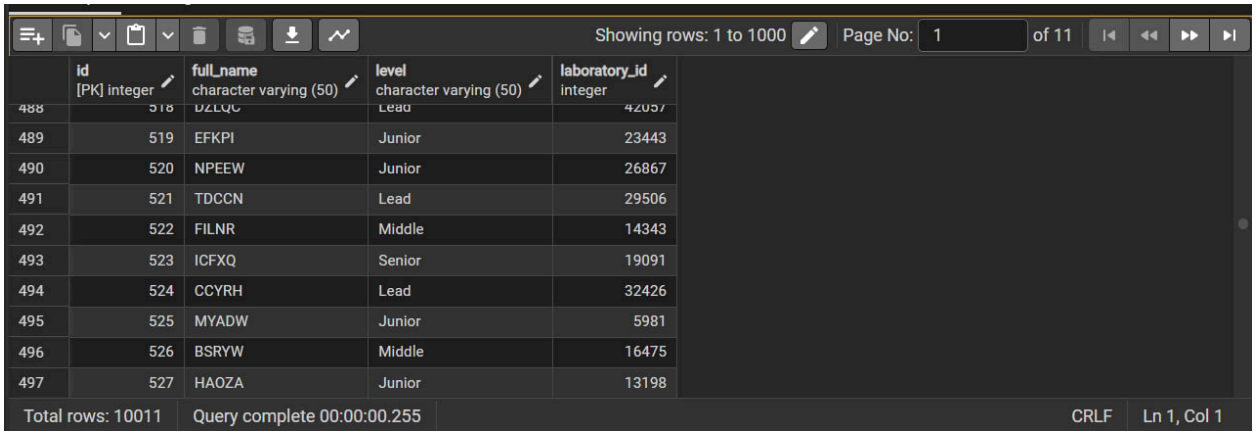
	id [PK] integer	lab_name character varying (100)
517	521	IMQ-R
518	522	KYV-L
519	523	GSY-R
520	524	USM-I
521	525	OMH-O
522	526	HXN-R
523	527	SAB-O
524	528	UCC-I
525	529	MYN-L
526	530	OCO-I

Запит SQL:

```
WITH gen AS (
   SELECT
      chr(65 + trunc(random()*26)::int) ||
      chr(65 + trunc(random()*26)::int) ||
      chr(65 + trunc(random()*26)::int) ||
      '-' ||
      (ARRAY['L','O','I','R'])[floor(random()*4)::int + 1] AS lab_name
   FROM generate_series(1, %s)
)
INSERT INTO laboratory(lab_name)
SELECT DISTINCT lab_name
FROM gen
WHERE lab_name NOT IN (SELECT lab_name FROM laboratory);
```

Такі ж дії проведемо для других таблиць, створимо там різну кількість даних.

```
Input number of the option: 5


        Task 2: choose what to generate:

        1. generate_labs

        2. generate_researchers

        3. generate_objects

        4. generate_object_types

Input number of the option: 2
Enter number of researcher records to generate: 10000
[TASK2] Generating 10000 researchers...
[TASK2] Researchers inserted (approx): 10000
```



Showing rows: 1 to 1000 | Page No: 1 of 11

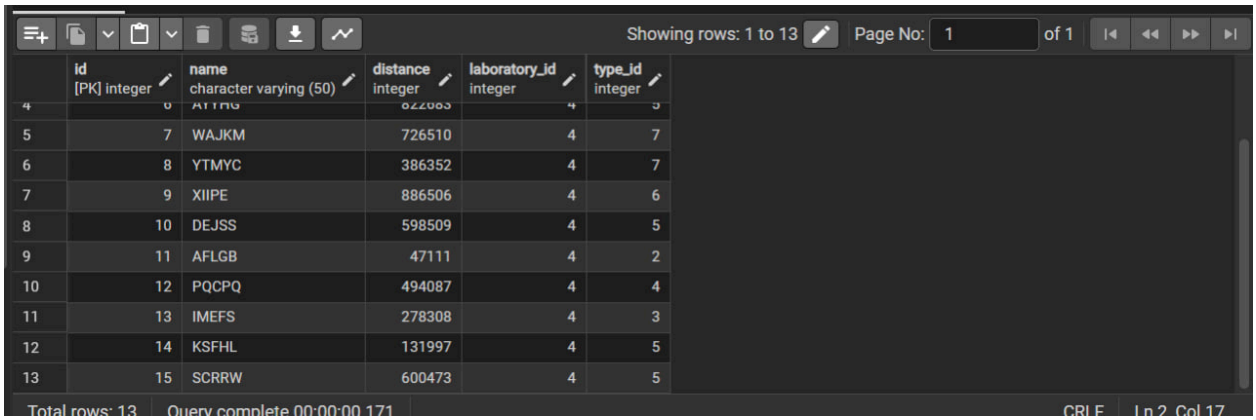
	id [PK] integer	full_name character varying (50)	level character varying (50)	laboratory_id integer
488	518	DZLQC	Lead	42057
489	519	EFKPI	Junior	23443
490	520	NPEEW	Junior	26867
491	521	TDCCN	Lead	29506
492	522	FILNR	Middle	14343
493	523	ICFXQ	Senior	19091
494	524	CCYRH	Lead	32426
495	525	MYADW	Junior	5981
496	526	BSRYW	Middle	16475
497	527	HAOZA	Junior	13198

Total rows: 10011 | Query complete 00:00:00.255 | CRLF | Ln 1, Col 1

SQL запит:

```
WITH params AS (
   SELECT %s::int[] AS lab_ids, %s::int AS n
),
gen AS (
   SELECT
      -- Random full name: 5 uppercase letters
      chr(65 + floor(random()*26)::int) ||
      chr(65 + floor(random()*26)::int) ||
      chr(65 + floor(random()*26)::int) ||
      chr(65 + floor(random()*26)::int) ||
      chr(65 + floor(random()*26)::int) AS full_name,
```

```
        -- Random research level
        (ARRAY['Junior','Middle','Senior','Lead'])[floor(random()*4)::int + 1] AS
level,

        -- Random lab id chosen from provided lab_ids array
        params.lab_ids[floor(random() * array_length(params.lab_ids, 1))::int + 1]
AS lab_id

    FROM params, generate_series(1, params.n)
)
INSERT INTO researcher(full_name, level, laboratory_id)
SELECT full_name, level, lab_id
FROM gen;
```

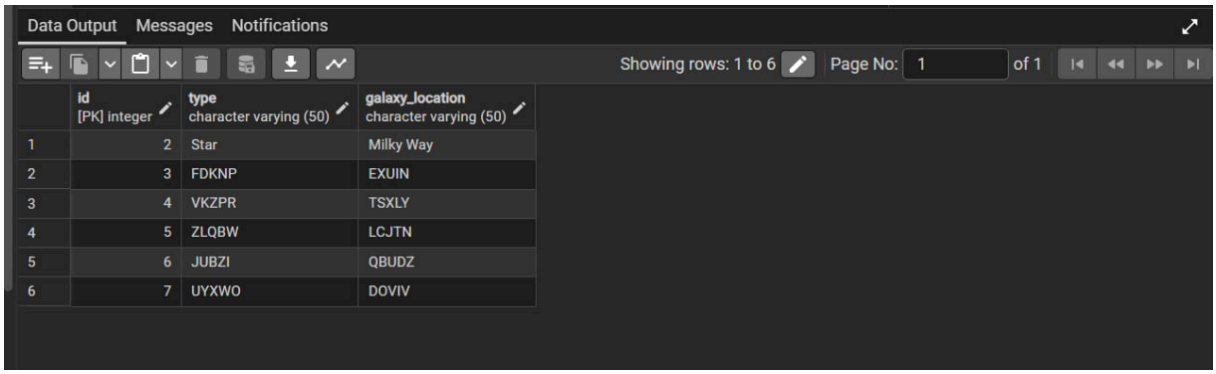


	id [PK] integer	name character varying (50)	distance integer	laboratory_id integer	type_id integer
5	7	WAJKM	726510	4	7
6	8	YTMYC	386352	4	7
7	9	XIIPE	886506	4	6
8	10	DEJSS	598509	4	5
9	11	AFLGB	47111	4	2
10	12	PQCPQ	494087	4	4
11	13	IMEFS	278308	4	3
12	14	KSFHL	131997	4	5
13	15	SCRRW	600473	4	5

SQL запит:

```
WITH params AS (
    SELECT %s::int[] AS lab_ids,
           %s::int[] AS type_ids,
           %s::int    AS n
),
gen AS (
    SELECT
        -- random object name
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
```

```
        chr(65 + floor(random()*26)::int) AS name,

        -- random distance between 1,000 and 1,000,000,000
        (random() * 999999000 + 1000)::int AS distance,

        -- random lab
        params.lab_ids[
            floor(random() * array_length(params.lab_ids, 1))::int + 1
        ] AS lab_id,

        -- random type
        params.type_ids[
            floor(random() * array_length(params.type_ids, 1))::int + 1
        ] AS type_id

    FROM params, generate_series(1, params.n)
)
INSERT INTO object(name, distance, laboratory_id, type_id)
SELECT name, distance, lab_id, type_id
FROM gen;
```



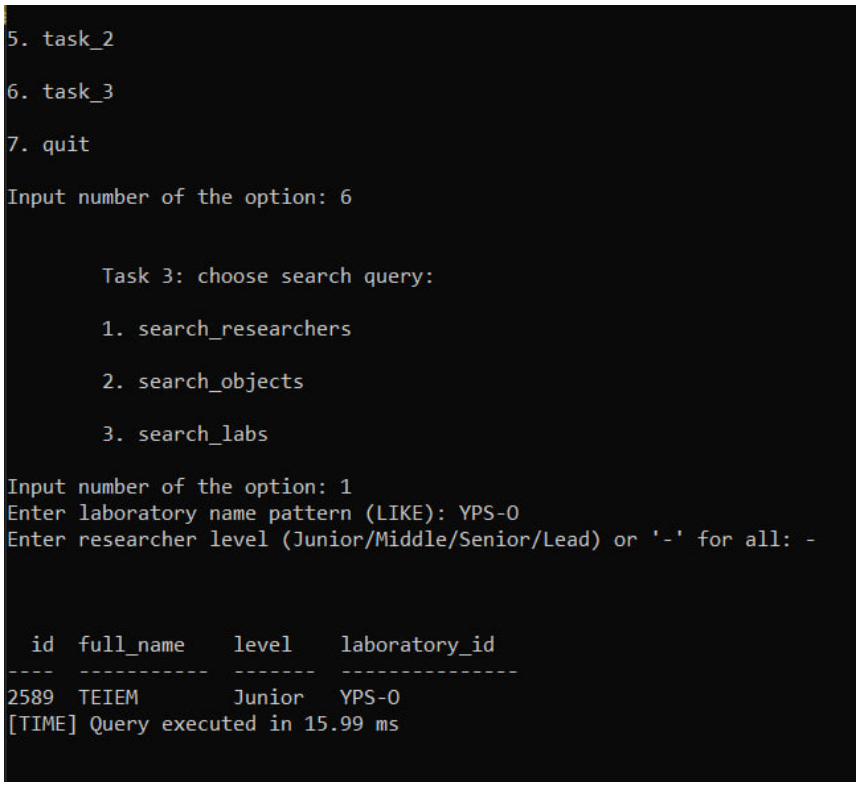
	id [PK] integer	type character varying (50)	galaxy_location character varying (50)
1	2	Star	Milky Way
2	3	FDKNP	EXUIN
3	4	VKZPR	TSXLY
4	5	ZLQBW	LCJTN
5	6	JUBZI	QBUDZ
6	7	UYXWO	DOVIV

SQL запит:

```
WITH gen AS (
    SELECT
        -- random type name (5 LETTERS)
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
```

```
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int)     AS type,

        -- random location (5 LETTERS)
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int) ||
        chr(65 + floor(random()*26)::int)     AS galaxy_location
    FROM generate_series(1, %s)
)
INSERT INTO object_type(type, galaxy_location)
SELECT type, galaxy_location FROM gen;
```

Дані створені, тепер можна використати зручний перегляд великої кількості даних за вибраними користувачем параметрами у неї. Для початку спробуємо в кучі даних знайти за лабораторією, якогось науковця. Додано тут зручну функцію для того, щоб знаходити дані за одним параметром або заразу за всіма. Щоб не звертатися до якогось стовпця можна написати знак “-”.



```
6. task_3

7. quit

Input number of the option: 6


        Task 3: choose search query:

        1. search_researchers

        2. search_objects

        3. search_labs

Input number of the option: 1
Enter laboratory name pattern (LIKE): ZBW-L
Enter researcher level (Junior/Middle/Senior/Lead) or '-' for all: -


id    full_name    level    laboratory_id
----  -----------  -------  --------------
[TIME] Query executed in 16.81 ms
```

```
Input number of the option: 6


        Task 3: choose search query:

        1. search_researchers

        2. search_objects

        3. search_labs

Input number of the option: 1
Enter laboratory name pattern (LIKE): Лабораторія НАУ
Enter researcher level (Junior/Middle/Senior/Lead) or '-' for all: Junior



  id  full_name    level    laboratory_id
----  -----------  -------  --------------
  37  EQWRV        Junior   Лабораторія НАУ
  40  PCJKL        Junior   Лабораторія НАУ
[TIME] Query executed in 9.39 ms
```

SQL запит:

```
SELECT r.id, r.full_name, r.level, l.lab_name
FROM researcher r
JOIN laboratory l ON r.laboratory_id = l.id
WHERE
   (%s = '-' OR l.lab_name LIKE %s)
   AND (%s = '-' OR r.level = %s)
ORDER BY r.id;
```

Тепер спробуємо щось знайти в таблиці object та laboratory.

```
Input number of the option: 6


        Task 3: choose search query:

        1. search_researchers

        2. search_objects

        3. search_labs

Input number of the option: 2
Enter laboratory name pattern (LIKE): Лабораторія НАУ
Enter object type pattern (LIKE): Star


  id  name          distance  laboratory_id    type    galaxy_location
----  ------  --------------  ---------------  ------  -----------------
   3  Sun          1.496e+08  Лабораторія НАУ  Star    Milky Way
   4  EFAWW   678975          Лабораторія НАУ  Star    Milky Way
   5  LFWJP   391264          Лабораторія НАУ  Star    Milky Way
  11  AFLGB    47111          Лабораторія НАУ  Star    Milky Way
[TIME] Query executed in 6.05 ms
```

SQL запит:

```
SELECT o.id, o.name, CAST(o.distance AS numeric), l.lab_name, t.type,
t.galaxy_location
 FROM object o
 JOIN laboratory l ON o.laboratory_id = l.id
 JOIN object_type t ON o.type_id = t.id
```

```
WHERE
   (%s = '' OR %s = '-' OR l.lab_name LIKE %s)
AND
   (%s = '' OR %s = '-' OR t.type LIKE %s)
ORDER BY o.id;
```

```
Input number of the option: 6


        Task 3: choose search query:

        1. search_researchers

        2. search_objects

        3. search_labs

Input number of the option: 3
Enter researcher name (LIKE) or '-' for all: Alex
Enter researcher level (Junior/Middle/Senior/Lead) or '-' for all: -
Enter object name (LIKE) or '-' for all: -


  id  lab_name
----  ---------------
   4  Лабораторія НАУ
[TIME] Query executed in 0.000 ms
```

```
Input number of the option: 6


        Task 3: choose search query:

        1. search_researchers

        2. search_objects

        3. search_labs

Input number of the option: 3
Enter researcher name (LIKE) or '-' for all: -
Enter researcher level (Junior/Middle/Senior/Lead) or '-' for all: -
Enter object name (LIKE) or '-' for all: BNACB



  id  lab_name
----  ----------
3484  ZYP-I
[TIME] Query executed in 5.074 ms
```

SQL запит:

```
SELECT DISTINCT
   l.id,
   l.lab_name
FROM laboratory l
LEFT JOIN researcher r ON r.laboratory_id = l.id
LEFT JOIN object o ON o.laboratory_id = l.id
WHERE
   (%s = '' OR %s = '-' OR r.full_name LIKE %s)
   AND (%s = '' OR %s = '-' OR r.level = %s)
   AND (%s = '' OR %s = '-' OR o.name LIKE %s)
ORDER BY l.id;
```

Весь код модуля Model:

```
from psycopg2 import connect
import time

class Model:
    def __init__(self):
        self.connection = connect(
            database="postgres",
            user="postgres",
            password="1234",
            host="localhost",
            port="5432",
        )

        # ======== INSERT QUERIES ========
        self.insert_queries = {
            "laboratory": """INSERT INTO laboratory(lab_name) VALUES (%s)""",
            "researcher": """INSERT INTO researcher(full_name, level, laboratory_id) VALUES (%s,
%s, %s)""",
            "object_type": """INSERT INTO object_type(type, galaxy_location) VALUES (%s, %s)""",
            "object": """INSERT INTO object(name, distance, laboratory_id, type_id) VALUES (%s,
%s, %s, %s)""",
        }

        # ======== READ QUERIES ========
        self.read_queries = {
            "laboratory": "SELECT id, lab_name FROM laboratory",
            "researcher": "SELECT r.id, r.full_name, r.level, l.lab_name "
                          "FROM researcher AS r "
                          "JOIN laboratory AS l ON r.laboratory_id = l.id",
            "object": "SELECT o.id, o.name, o.distance, l.lab_name, t.type, t.galaxy_location "
                      "FROM object AS o "
                      "JOIN laboratory AS l ON o.laboratory_id = l.id "
                      "JOIN object_type AS t ON o.type_id = t.id",
            "object_type": "SELECT id, type, galaxy_location FROM object_type",
        }

        # ======== DELETE QUERIES ========
        self.delete_queries = {
            "laboratory": "DELETE FROM laboratory WHERE id = %s",
            "researcher": "DELETE FROM researcher WHERE id = %s",
            "object": "DELETE FROM object WHERE id = %s",
            "object_type": "DELETE FROM object_type WHERE id = %s",
```

```
    }

    # ======== UPDATE QUERIES ========
    self.update_queries = {
        "laboratory": {
            "lab_name": "UPDATE laboratory SET lab_name = %s WHERE id = %s",
        },
        "researcher": {
            "full_name": "UPDATE researcher SET full_name = %s WHERE id = %s",
            "level": "UPDATE researcher SET level = %s WHERE id = %s",
            "laboratory_id": "UPDATE researcher SET laboratory_id = %s WHERE id = %s",
        },
        "object_type": {
            "type": "UPDATE object_type SET type = %s WHERE id = %s",
            "galaxy_location": "UPDATE object_type SET galaxy_location = %s WHERE id = %s",
        },
        "object": {
            "name": "UPDATE object SET name = %s WHERE id = %s",
            "distance": "UPDATE object SET distance = %s WHERE id = %s",
            "laboratory_id": "UPDATE object SET laboratory_id = %s WHERE id = %s",
            "type_id": "UPDATE object SET type_id = %s WHERE id = %s",
        },
    }

# ======== BASIC METHODS ========

def disconnect(self):
    if self.connection and self.connection.closed == 0:
        self.connection.close()

def _execute_select(self, query: str, data=None) -> list:
    cur = self.connection.cursor()
    try:
        cur.execute(query, data or ())
        rows = cur.fetchall()
        cur.close()
        return rows
    except Exception as e:
        print(f"\n Unexpected error in SELECT: {type(e).__name__} {e}\n")
        self.connection.rollback()
        cur.close()
        return []

def _execute_modify(self, query: str, data: tuple):
```

```
    cur = self.connection.cursor()
    try:
        print(f"[DEBUG] Executing SQL: {query}")
        print(f"[DEBUG] With data: {data}")

        cur.execute(query, data)
        self.connection.commit()

        affected = cur.rowcount
        cur.close()
        return affected

    except Exception as e:
        print(f"\n Unexpected error in modify-query: {type(e).__name__} {e}\n")
        self.connection.rollback()     # ?? CRITICAL FIX
        cur.close()
        return 0


# ======== CRUD OPERATIONS ========

## CREATE
def create_laboratory(self, lab_name):
    self._execute_modify(self.insert_queries["laboratory"], (lab_name,))

def create_researcher(self, full_name, level, laboratory_id):
    self._execute_modify(self.insert_queries["researcher"], (full_name, level, laboratory_id))

def create_object_type(self, type_name, galaxy_location):
    self._execute_modify(self.insert_queries["object_type"], (type_name, galaxy_location))

def create_object(self, name, distance, laboratory_id, type_id):
    self._execute_modify(self.insert_queries["object"], (name, distance, laboratory_id, type_id))

## READ
def read(self, table_name):
    return self._execute_select(self.read_queries[table_name])

## UPDATE
def update_laboratory_field(self, lab_id, new_name):
    query = self.update_queries["laboratory"]["lab_name"]
    affected = self._execute_modify(query, (new_name, lab_id))
    return affected
```

```
def update_researcher_field(self, researcher_id, field, new_value):
    query = self.update_queries["researcher"].get(field)
    if not query:
        raise ValueError(f"Unknown field for researcher: {field}")
    affected = self._execute_modify(query, (new_value, researcher_id))
    return affected

def update_object_type_field(self, type_id, field, new_value):
    query = self.update_queries["object_type"].get(field)
    if not query:
        raise ValueError(f"Unknown field for object_type: {field}")
    affected = self._execute_modify(query, (new_value, type_id))
    return affected


def update_object_field(self, object_id, field, new_value):
    query = self.update_queries["object"].get(field)
    if not query:
        raise ValueError(f"Unknown field for object: {field}")
    affected = self._execute_modify(query, (new_value, object_id))
    return affected
## DELETE
def delete(self, table_name, record_id):
    self._execute_modify(self.delete_queries[table_name], (record_id,))

# ======== DELETE METHODS ========

def delete_laboratory(self, lab_id):
    affected = self._execute_modify(self.delete_queries["laboratory"], (lab_id,))
    if affected == 0:
        print(f"[INFO] No laboratory with id={lab_id} nothing deleted.")
    else:
        print(f"[SUCCESS] Laboratory id={lab_id} deleted successfully.")
    return affected


def delete_researcher(self, researcher_id):
    affected = self._execute_modify(self.delete_queries["researcher"], (researcher_id,))
    if affected == 0:
        print(f"[INFO] No researcher with id={researcher_id} nothing deleted.")
    else:
        print(f"[SUCCESS] Researcher id={researcher_id} deleted successfully.")
    return affected
```

```
def delete_object(self, object_id):
    affected = self._execute_modify(self.delete_queries["object"], (object_id,))
    if affected == 0:
        print(f"[INFO] No object with id={object_id} nothing deleted.")
    else:
        print(f"[SUCCESS] Object id={object_id} deleted successfully.")
    return affected


def delete_object_type(self, type_id):
    affected = self._execute_modify(self.delete_queries["object_type"], (type_id,))
    if affected == 0:
        print(f"[INFO] No object_type with id={type_id} nothing deleted.")
    else:
        print(f"[SUCCESS] Object type id={type_id} deleted successfully.")
    return affected


def generate_laboratories(self, n: int):
    query = """
    WITH gen AS (
        SELECT
            chr(65 + trunc(random()*26)::int) ||
            chr(65 + trunc(random()*26)::int) ||
            chr(65 + trunc(random()*26)::int) ||
            '-' ||
            (ARRAY['L','O','I','R'])[floor(random()*4)::int + 1] AS lab_name
        FROM generate_series(1, %s)
    )
    INSERT INTO laboratory(lab_name)
    SELECT DISTINCT lab_name
    FROM gen
    WHERE lab_name NOT IN (SELECT lab_name FROM laboratory);
    """

    cur = self.connection.cursor()
    cur.execute(query, (n,))
    self.connection.commit()

    rowcount = cur.rowcount
    cur.close()
    return rowcount
```

```
def generate_researchers(self, n: int):
    # Get all existing laboratory IDs
    lab_query = "SELECT id FROM laboratory;"
    lab_ids = self._execute_select(lab_query)

    if not lab_ids:
        print("[ERROR] Cannot generate researchers: no laboratories exist.")
        return 0

    flat_ids = [row[0] for row in lab_ids]

    sql = """
    WITH params AS (
        SELECT %s::int[] AS lab_ids, %s::int AS n
    ),
    gen AS (
        SELECT
            -- Random full name: 5 uppercase letters
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) AS full_name,

            -- Random research level
            (ARRAY['Junior','Middle','Senior','Lead'])[floor(random()*4)::int + 1] AS level,

            -- Random lab id chosen from provided lab_ids array
            params.lab_ids[floor(random() * array_length(params.lab_ids, 1))::int + 1] AS lab_id

        FROM params, generate_series(1, params.n)
    )
    INSERT INTO researcher(full_name, level, laboratory_id)
    SELECT full_name, level, lab_id
    FROM gen;
    """

    cur = self.connection.cursor()
    # pass (lab_ids_array, n) — note the order matches params in SQL
    cur.execute(sql, (flat_ids, n))
    self.connection.commit()
    affected = cur.rowcount
    cur.close()
```

```
    return affected


def generate_objects(self, n: int):
    # get lab IDs
    lab_ids = self._execute_select("SELECT id FROM laboratory")
    if not lab_ids:
        print("[ERROR] Cannot generate objects: no laboratories exist.")
        return 0

    flat_labs = [row[0] for row in lab_ids]

    # get type IDs
    type_ids = self._execute_select("SELECT id FROM object_type")
    if not type_ids:
        print("[ERROR] Cannot generate objects: no object types exist.")
        return 0

    flat_types = [row[0] for row in type_ids]

    sql = """
    WITH params AS (
        SELECT %s::int[] AS lab_ids,
               %s::int[] AS type_ids,
               %s::int    AS n
    ),
    gen AS (
        SELECT
            -- random object name
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) AS name,

            -- random distance between 1,000 and 1,000,000,000
            (random() * 999999000 + 1000)::int AS distance,

            -- random lab
            params.lab_ids[
                floor(random() * array_length(params.lab_ids, 1))::int + 1
            ] AS lab_id,

            -- random type
```

```
            params.type_ids[
                floor(random() * array_length(params.type_ids, 1))::int + 1
            ] AS type_id

        FROM params, generate_series(1, params.n)
    )
    INSERT INTO object(name, distance, laboratory_id, type_id)
    SELECT name, distance, lab_id, type_id
    FROM gen;
    """

    cur = self.connection.cursor()
    cur.execute(sql, (flat_labs, flat_types, n))
    self.connection.commit()

    affected = cur.rowcount
    cur.close()
    return affected


def generate_object_types(self, n: int):

    sql = """
    WITH gen AS (
        SELECT
            -- random type name (5 LETTERS)
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int)     AS type,

            -- random location (5 LETTERS)
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int) ||
            chr(65 + floor(random()*26)::int)     AS galaxy_location
        FROM generate_series(1, %s)
    )
    INSERT INTO object_type(type, galaxy_location)
    SELECT type, galaxy_location FROM gen;
    """
```

```
        cur = self.connection.cursor()
        cur.execute(sql, (n,))
        self.connection.commit()

        affected = cur.rowcount
        cur.close()
        return affected

    def search_researchers(self, lab_like, level):
        sql = """
        SELECT r.id, r.full_name, r.level, l.lab_name
        FROM researcher r
        JOIN laboratory l ON r.laboratory_id = l.id
        WHERE
            (%s = '-' OR l.lab_name LIKE %s)
            AND (%s = '-' OR r.level = %s)
        ORDER BY r.id;
        """

        args = [
            lab_like, f"%{lab_like}%",
            level, level
        ]

        t0 = time.time()
        cur = self.connection.cursor()
        cur.execute(sql, args)
        rows = cur.fetchall()
        ms = (time.time() - t0) * 1000
        cur.close()

        return rows, ms

    def search_objects(self, lab_like, type_like):
        sql = """
            SELECT o.id, o.name, CAST(o.distance AS numeric), l.lab_name, t.type,
t.galaxy_location
            FROM object o
            JOIN laboratory l ON o.laboratory_id = l.id
            JOIN object_type t ON o.type_id = t.id
            WHERE
                (%s = '' OR %s = '-' OR l.lab_name LIKE %s)
            AND
                (%s = '' OR %s = '-' OR t.type LIKE %s)
```

```
        ORDER BY o.id;
    """
    t0 = time.time()
    cur = self.connection.cursor()

    args = [
        lab_like, lab_like, f"%{lab_like}%",
        type_like, type_like, f"%{type_like}%"
    ]

    cur.execute(sql, args)
    rows = cur.fetchall()
    ms = (time.time() - t0) * 1000
    cur.close()
    return rows, ms

def search_labs(self, rname_like, level, obj_like):
    sql = """
    SELECT DISTINCT
        l.id,
        l.lab_name
    FROM laboratory l
    LEFT JOIN researcher r ON r.laboratory_id = l.id
    LEFT JOIN object o ON o.laboratory_id = l.id
    WHERE
        (%s = '' OR %s = '-' OR r.full_name LIKE %s)
        AND (%s = '' OR %s = '-' OR r.level = %s)
        AND (%s = '' OR %s = '-' OR o.name LIKE %s)
    ORDER BY l.id;
    """

    args = [
        rname_like, rname_like, f"%{rname_like}%",
        level, level, level,
        obj_like, obj_like, f"%{obj_like}%"
    ]

    import time
    t0 = time.time()
    cur = self.connection.cursor()
    cur.execute(sql, args)
    rows = cur.fetchall()
    t = (time.time() - t0) * 1000
    cur.close()
```

```
return rows, t
```

Опис модуля Model

Модуль Model реалізує роботу з базою даних PostgreSQL та відповідає за виконання всіх операцій CRUD, генерацію тестових даних та реалізацію пошукових запитів. Він є частиною шаблону MVC і виконує роль шару доступу до даних.

1. Підключення до БД

У конструкторі класу виконується встановлення з'єднання з PostgreSQL через psycopg2.connect().
Також визначаються словники SQL-запитів:

insert_queries — запити INSERT

read_queries — SELECT

update_queries — UPDATE

delete_queries — DELETE

Це забезпечує централізоване зберігання SQL-команд.

2. Допоміжні методи

_execute_select(query, data)

Виконує SELECT-запити.
Повертає список рядків або порожній список у разі помилки (з rollback).

_execute_modify(query, data)

Виконує INSERT, UPDATE, DELETE.
Повертає кількість змінених рядків.
Передбачений rollback у випадку помилок, щоб не блокувати наступні запити.

disconnect()

Закриває з’єднання з БД.

3. Операції CREATE

Реалізовані методи для створення записів у всіх таблицях:

create_laboratory

create_researcher

create_object_type

create_object

Використовують _execute_modify().

4. Операції READ

Метод:

read(table_name)

Повертає всі записи з відповідної таблиці.
SQL-запити зібрані у словнику read_queries.

5. Операції UPDATE

Методи оновлення для кожної сутності:

update_laboratory_field

update_researcher_field

update_object_type_field

update_object_field

Кожен метод вибирає відповідний SQL-запит та виконує його через _execute_modify().

6. Операції DELETE

Методи видалення:

delete_laboratory

delete_researcher

delete_object

delete_object_type

Після виконання виводять інформаційне повідомлення про результат.

7. Генерація випадкових даних (Task 2)

Методи виконують масове створення даних SQL-запитами з CTE:

generate_laboratories(n) — створення випадкових назв лабораторій.

generate_researchers(n) — створення науковців з випадковими рівнями та випадковим lab_id.

generate_object_types(n) — випадкові типи об’єктів.

generate_objects(n) — створення об’єктів з випадковими відстанями, лабораторіями та типами.

Це дозволяє швидко наповнити базу тестовими наборами даних.

8. Пошукові функції (Task 3)

search_researchers(lab_like, level)

Повертає науковців за частиною назви лабораторії та рівнем.

search_objects(lab_like, type_like)

Пошук об’єктів за лабораторією та типом об’єкта.

search_labs(rname, level, obj_name)

Пошук лабораторій за трьома ознаками: ім’я науковця, рівень, ім’я об’єкта

Весь код модуля