A black and white drawing of a building

Description automatically generated

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 і засоби управління**

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

*PostgreSQL ”*

Виконав:

студент ІІI курсу

групи КВ-12

Левчук І. В.

Перевірив:

Павловский В. І.

Київ – 2023

**Мета роботи:** здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних

**Виконання роботи**

**Опис сутностей**

Згідно цієї області для побудови бази даних обраної області, були виділені такі сутності:

* Звіт (Report) з атрибутами: ідентифікатор звіту, дата заповнення, ідентифікатор регіону, ідентифікатор екологічного показника. Призначення: збереження даних щодо звітів.
* Регіон (Region) з атрибутами: ідентифікатор регіону, назва, площа. Призначення: збереження даних щодо регіонів.
* Екологічний Показник (Indicator) з атрибутами: ідентифікатор екологічного показника, назва, одиниці виміру. Призначення: збереження даних щодо занять.
* Спостерігач (Observer) з атрибутами: ідентифікатор Спостерігача, ім'я, Прізвище, Адреса. Призначення: збереження даних щодо викладачів.

**Опис зв’язків**

Зв’язок «Регіон» - «Звіт» є зв’язком 1:N. Один регіон може мати багато звітів, але кожен звіт належить лише одному регіону.

Зв’язок «Звіт» - «Екологічні показники» є зв’язком N:M. Кожен Звіт може містити багато Екологічних Показників, і кожен Показник може бути включений у багато Звітів.

Зв’язок «Регіон» - «Спостерігач» є зв’язком 1:N. Один регіон може мати багато спостерігачів, але кожен спостерігач належить лише одному регіону.

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

Изображение выглядит как диаграмма, линия, текст, чек

Автоматически созданное описание

Рисунок 1 – ER-діаграма, побудована за нотацією Чена

**Перетворення концептуальної моделі у логічну модель та схему бази даних**

Сутність Report перетворено в таблицю Report з первинним ключем report\_id та атрибутами submition\_date та зовнішніми ключами region\_id.

Сутність Region перетворено в таблицю Region з первинним ключем region\_id та атрибутами name, area.

Сутність Indicator перетворено в таблицю Indicator з первинним ключем indicator\_id та атрибутами name, Unit\_of\_Measurement.

Сутність Observer перетворено в таблицю Observer з первинним ключем observer\_id та атрибутами address, first\_name, last\_name.

Сутність Region\_Observer\_Report перетворено в таблицю Region\_Observer\_Report з первинним ключем reg\_obs\_rep\_id та атрибутами

Оскільки в логічній моделі безпосередній зв’язок N:M є неможливим, а в концептуальній моделі він існує між сутностями Indicator i Report, то для його реалізації було створено таблицю Indicator\_Report, з первинним ключем ind\_rep\_id, та зовнішніми ключами indicator\_id i report\_id.

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

Изображение выглядит как текст, снимок экрана, Шрифт, Параллельный

Автоматически созданное описание

Рисунок 2 – Схема бази даних

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також стороння бібліотека, що надає API для доступу до PostgreSQL – psycopg2.

**Шаблон проектування**

MVC - Шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходять у папці Models.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

**Структура програми та її опис**

****

**Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание**

Рисунок 3 - Структура програми

Програма умовно поділена на 7 модулів:

1. файл Controller.py;
2. файл Model.py;
3. файл View.py.

Модулі, як видно з їх назв, повністю відповідають використаному патерну MVC.

У файлі Model (.py) описаний клас моделі, що займається регулюванням підключення до бази даних, та виконанням низькорівневих запитів до неї.

У файлі Controller (.py) описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі View (.py) описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

**Структура меню програми**

Меню для користувача складається з 8-ми пунктів (Рисунок 4).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 4 - Меню програми

Також для всіх таблиць (окрім звязків) реалізовано окреме підменю яке

викликається після обрання відповідної таблиці у головному меню(Рисунки 5-8).

**Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание** **Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание**

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Рисунки 5-8 - Підменю програми

У цих підменю доступні операції: додавання, перегляду, редагування та

вилученя даних у таблицях бази даних.

Для пункту 5(Генерування даних) також передбачене підменю в якому

можна обрати таблицю для якої , ви хочете згенерувати дані, попередньо

обравши кількість даних дял генерації(Рисунок 9).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 9 - Підменю генераціі

Для пункту 6(Пошук) було створено 3 запити , які реалізують пошук у

таблицях. Для цього є відповідне підменю з запитами(Рисунок 10).

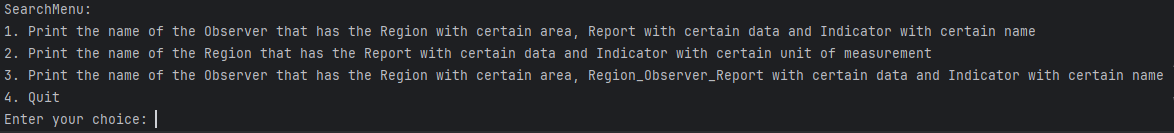
****

Рисунок 10 - Підменю пошуку

**Повний текст програми:**

Модуль сontroller.py

import time  
from model import Model  
from view import View  
  
  
class Controller\_Indicator:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
  
 def run(self):  
 while True:  
 choice = self.show\_submenu()  
 if choice == '1':  
 self.add\_Indicator()  
 elif choice == '2':  
 self.view\_Indicator()  
 elif choice == '3':  
 self.update\_Indicator()  
 elif choice == '4':  
 self.delete\_Indicator()  
 elif choice == '5':  
 break  
  
 def show\_submenu(self):  
 self.view.show\_message("\nSub Menu Indicator :")  
 self.view.show\_message("1. Add Indicator")  
 self.view.show\_message("2. View Indicator")  
 self.view.show\_message("3. Update Indicator")  
 self.view.show\_message("4. Delete Indicator")  
 self.view.show\_message("5. Quit")  
 return input("Enter your choice: ")  
  
 def add\_Indicator(self):  
 name, unit\_of\_measurement = self.view.get\_Indicator\_input()  
  
 if len(name) > 50 or len(unit\_of\_measurement) > 50:  
 self.view.show\_message("Error: The values of the columns exceed their length limit.")  
 return  
  
 result\_message = self.model.add\_Indicator(name, unit\_of\_measurement)  
 self.view.show\_message(result\_message)  
  
 def view\_Indicator(self):  
 indicators = self.model.get\_all\_Indicator()  
 self.view.show\_Indicator(indicators)  
  
 def update\_Indicator(self):  
 indicator\_id = self.view.get\_id()  
 if not indicator\_id.isdigit():  
 self.view.show\_message("Error: 'Indicator\_id' must be a number.")  
 return  
  
 name, unit\_of\_measurement = self.view.get\_Indicator\_update()  
  
 if len(name) > 50 or len(unit\_of\_measurement) > 50:  
 self.view.show\_message("Error: The values of the columns exceed their length limit.")  
 return  
  
 result\_message = self.model.update\_Indicator(indicator\_id, name, unit\_of\_measurement)  
 self.view.show\_message(result\_message)  
  
 def delete\_Indicator(self):  
 indicator\_id = self.view.get\_id()  
 if not indicator\_id.isdigit():  
 self.view.show\_message("Error: 'Indicator\_id' must be a number.")  
 return  
  
 result\_message = self.model.delete\_Indicator(indicator\_id)  
 self.view.show\_message(result\_message)  
  
class Controller\_Region:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
  
 def run(self):  
 while True:  
 choice = self.show\_submenu()  
 if choice == '1':  
 self.add\_Region()  
 elif choice == '2':  
 self.view\_Region()  
 elif choice == '3':  
 self.update\_Region()  
 elif choice == '4':  
 self.delete\_Region()  
 elif choice == '5':  
 break  
  
 def show\_submenu(self):  
 self.view.show\_message("\nSub Menu Region:")  
 self.view.show\_message("1. Add Region")  
 self.view.show\_message("2. View Regions")  
 self.view.show\_message("3. Update Region")  
 self.view.show\_message("4. Delete Region")  
 self.view.show\_message("5. Quit")  
 return input("Enter your choice: ")  
  
 def add\_Region(self):  
 name, area = self.view.get\_Region\_input()  
  
 if not area.isdigit():  
 self.view.show\_message("Error: 'Region\_id' and 'Area' must be numbers.")  
 return  
  
 if len(name) > 30:  
 self.view.show\_message("Error: The value of the 'Name' column exceeds 30 characters.")  
 return  
  
 result\_message = self.model.add\_Region(name, area)  
 self.view.show\_message(result\_message)  
  
 def view\_Region(self):  
 regions = self.model.get\_all\_Region()  
 self.view.show\_Region(regions)  
  
 def update\_Region(self):  
 region\_id = self.view.get\_id()  
 if not region\_id.isdigit():  
 self.view.show\_message("Error: 'Region\_id' must be a number.")  
 return  
  
 name, area = self.view.get\_Region\_update()  
 if not area.isdigit():  
 self.view.show\_message("Error: 'Area' must be a number.")  
 return  
  
 if len(name) > 30:  
 self.view.show\_message("Error: The value of the 'Name' column exceeds 30 characters.")  
 return  
  
 result\_message = self.model.update\_Region(region\_id, name, area)  
 self.view.show\_message(result\_message)  
  
 def delete\_Region(self):  
 region\_id = self.view.get\_id()  
 if not region\_id.isdigit():  
 self.view.show\_message("Error: 'Region\_id' must be a number.")  
 return  
  
 result\_message = self.model.delete\_Region(region\_id)  
 self.view.show\_message(result\_message)  
  
class Controller\_Report:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
  
 def run(self):  
 while True:  
 choice = self.show\_submenu()  
 if choice == '1':  
 self.add\_Report()  
 elif choice == '2':  
 self.view\_Report()  
 elif choice == '3':  
 self.update\_Report()  
 elif choice == '4':  
 self.delete\_Report()  
 elif choice == '5':  
 break  
  
 def show\_submenu(self):  
 self.view.show\_message("\nSub Menu Report:")  
 self.view.show\_message("1. Add Report")  
 self.view.show\_message("2. View Reports")  
 self.view.show\_message("3. Update Report")  
 self.view.show\_message("4. Delete Report")  
 self.view.show\_message("5. Quit")  
 return input("Enter your choice: ")  
  
 def add\_Report(self):  
 submission\_date = self.view.get\_Report\_input()  
  
 result\_message = self.model.add\_Report(submission\_date)  
 self.view.show\_message(result\_message)  
  
 def view\_Report(self):  
 reports = self.model.get\_all\_Report()  
 self.view.show\_Report(reports)  
  
 def update\_Report(self):  
 report\_id = self.view.get\_id()  
 if not report\_id.isdigit():  
 self.view.show\_message("Error: 'Report\_id' must be a number.")  
 return  
  
 submission\_date = self.view.get\_Report\_update()  
 result\_message = self.model.update\_Report(report\_id, submission\_date)  
 self.view.show\_message(result\_message)  
  
 def delete\_Report(self):  
 report\_id = self.view.get\_id()  
 if not report\_id.isdigit():  
 self.view.show\_message("Error: 'Report\_id' must be a number.")  
 return  
  
 result\_message = self.model.delete\_Report(report\_id)  
 self.view.show\_message(result\_message)  
  
class Controller\_Observer:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
  
 def run(self):  
 while True:  
 choice = self.show\_submenu()  
 if choice == '1':  
 self.add\_Observer()  
 elif choice == '2':  
 self.view\_Observer()  
 elif choice == '3':  
 self.update\_Observer()  
 elif choice == '4':  
 self.delete\_Observer()  
 elif choice == '5':  
 break  
  
 def show\_submenu(self):  
 self.view.show\_message("\nObserver Menu:")  
 self.view.show\_message("1. Add Observer")  
 self.view.show\_message("2. View Observers")  
 self.view.show\_message("3. Update Observer")  
 self.view.show\_message("4. Delete Observer")  
 self.view.show\_message("5. Quit")  
 return input("Enter your choice: ")  
  
 def add\_Observer(self):  
 region\_id, first\_name, last\_name, address = self.view.get\_Observer\_input()  
  
 if len(first\_name) > 50 or len(last\_name) > 50 or len(address) > 50:  
 self.view.show\_message("Error: The values of the columns exceed their length limit.")  
 return  
  
 result\_message = self.model.add\_Observer(region\_id, first\_name, last\_name, address)  
 self.view.show\_message(result\_message)  
  
 def view\_Observer(self):  
 observers = self.model.get\_all\_Observer()  
 self.view.show\_Observer(observers)  
  
 def update\_Observer(self):  
 observer\_id = self.view.get\_id()  
 if not observer\_id.isdigit():  
 self.view.show\_message("Error: 'Observer\_id' must be a number.")  
 return  
  
 first\_name, last\_name, address, region\_id = self.view.get\_Observer\_update()  
  
 if len(first\_name) > 50 or len(last\_name) > 50 or len(address) > 50:  
 self.view.show\_message("Error: The values of the columns exceed their length limit.")  
 return  
  
 if not region\_id.isdigit():  
 self.view.show\_message("Error: 'Region\_id' must be a number.")  
 return  
  
 result\_message = self.model.update\_Observer(observer\_id, region\_id, first\_name, last\_name, address)  
 self.view.show\_message(result\_message)  
  
 def delete\_Observer(self):  
 observer\_id = self.view.get\_id()  
 if not observer\_id.isdigit():  
 self.view.show\_message("Error: 'Observer\_id' must be a number.")  
 return  
  
 result\_message = self.model.delete\_Observer(observer\_id)  
 self.view.show\_message(result\_message)  
  
class Controller:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
 self.Controller\_Observer = Controller\_Observer()  
 self.Controller\_Indicator = Controller\_Indicator()  
 self.Controller\_Region = Controller\_Region()  
 self.Controller\_Report = Controller\_Report()  
  
 def run(self):  
 while True:  
 choice = self.show\_menu()  
 if choice == '1':  
 self.Controller\_Observer.run()  
 if choice == '2':  
 self.Controller\_Indicator.run()  
 if choice == '3':  
 self.Controller\_Region.run()  
 if choice == '4':  
 self.Controller\_Report.run()  
 if choice == '5':  
 self.Generate()  
 if choice == '6':  
 self.Search()  
 if choice == '7':  
 self.Reset()  
 if choice == '8':  
 break  
  
 def show\_menu(self):  
 self.view.show\_message("\nMenu:")  
 self.view.show\_message("1. Observer")  
 self.view.show\_message("2. Indicator")  
 self.view.show\_message("3. Region")  
 self.view.show\_message("4. Report")  
 self.view.show\_message("5. Generate")  
 self.view.show\_message("6. Search")  
 self.view.show\_message("7. Reset all")  
 self.view.show\_message("8. Quit")  
 return input("Enter your choice: ")  
  
 def show\_submenu(self):  
 self.view.show\_message("\nSubMenu:")  
 self.view.show\_message("1. Observer")  
 self.view.show\_message("2. Indicator")  
 self.view.show\_message("3. Region")  
 self.view.show\_message("4. Report")  
 self.view.show\_message("5. Indicator\_Report")  
 self.view.show\_message("6. Observer\_Region\_Report")  
 self.view.show\_message("7. Quit")  
 return input("Enter your choice: ")  
  
 def show\_searchmenu(self):  
 self.view.show\_message("\nSearchMenu:")  
 self.view.show\_message("1. Print the name of the Observer that has the Region with certain area, Report with certain data and Indicator with certain name")  
 self.view.show\_message("2. Print the name of the Region that has the Report with certain data and Indicator with certain unit of measurement")  
 self.view.show\_message("3. Print the name of the Observer that has the Region with certain area, Region\_Observer\_Report with certain data and Indicator with certain name")  
 self.view.show\_message("4. Quit")  
 return input("Enter your choice: ")  
  
 def Generate(self):  
 while True:  
 self.view.show\_message("Select the amount of data to generate.")  
 number = self.view.get\_number\_for\_generation()  
 if not number.isdigit():  
 self.view.show\_message("Error: 'number' must be a number.")  
 return  
 choice = self.show\_submenu()  
 if choice == '1':  
 self.model.generate\_observer(number)  
 elif choice == '2':  
 self.model.generate\_indicator(number)  
 elif choice == '3':  
 self.model.generate\_region(number)  
 elif choice == '4':  
 self.model.generate\_report(number)  
 elif choice == '5':  
 self.model.generate\_indicator\_report(number)  
 elif choice == '6':  
 self.model.generate\_observer\_region\_report(number)  
 elif choice == '7':  
 break  
 self.view.show\_message("Randomize data added successfully!")  
  
  
 def Reset(self):  
 self.model.reset()  
 self.view.show\_message("Reset complete successfully!")  
  
 def Search(self):  
 while True:  
 self.view.show\_message("Enter the number of the request to search.")  
 choice = self.show\_searchmenu()  
 if not choice.isdigit():  
 self.view.show\_message("Error: 'number' must be a number.")  
 return  
 start\_time = time.time()  
 if choice == '1':  
 search\_results , column\_description = self.model.search\_request1()  
 elif choice == '2':  
 search\_results, column\_description = self.model.search\_request2()  
 elif choice == '3':  
 search\_results, column\_description = self.model.search\_request3()  
 elif choice == '4':  
 break  
  
 end\_time = time.time()  
 execution\_time = (end\_time - start\_time) \* 1000  
  
 self.view.format\_and\_print\_results(search\_results, column\_description)  
 self.view.show\_message(f"Search request successfully done! Execution time: {execution\_time:.2f} milliseconds")

Модуль model.py

import psycopg2  
  
  
class Model:  
 def \_\_init\_\_(self):  
 self.conn = psycopg2.connect(  
 dbname='RGR',  
 user='postgres',  
 password='Qs2qwer',  
 host='localhost',  
 port=5432  
 )  
 self.create\_table\_Indicator()  
 self.create\_table\_Region()  
 self.create\_table\_Report()  
 self.create\_table\_Observer()  
 self.create\_table\_Indicator\_Report()  
 self.create\_table\_Region\_Observer\_Report()  
 def create\_table\_Indicator(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Indicator" (  
 indicator\_id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 unit\_of\_measurement TEXT NOT NULL  
 )  
 ''')  
  
 # Перевірка наявності таблиці  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Indicator')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Якщо таблиця не існує, то створити її  
 c.execute('''  
 CREATE TABLE "Indicator" (  
 indicator\_id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 unit\_of\_measurement TEXT NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def create\_table\_Indicator\_Report(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Indicator\_Report" (  
 ind\_rep\_id SERIAL PRIMARY KEY,  
 report\_id INTEGER NOT NULL,  
 indicator\_id INTEGER NOT NULL,  
 CONSTRAINT "Indicator\_Report\_indicator\_id\_fkey" FOREIGN KEY (indicator\_id)  
 REFERENCES "Indicator" (indicator\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION,  
 CONSTRAINT "Indicator\_Report\_report\_id\_fkey" FOREIGN KEY (report\_id)  
 REFERENCES "Report" (report\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
 )  
 ''')  
  
 # Перевірка наявності таблиці  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Indicator\_Report')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Якщо таблиця не існує, то створити її  
 c.execute('''  
 CREATE TABLE "Indicator\_Report" (  
 ind\_rep\_id SERIAL PRIMARY KEY,  
 report\_id INTEGER NOT NULL,  
 indicator\_id INTEGER NOT NULL,  
 CONSTRAINT "Indicator\_Report\_indicator\_id\_fkey" FOREIGN KEY (indicator\_id)  
 REFERENCES "Indicator" (indicator\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION,  
 CONSTRAINT "Indicator\_Report\_report\_id\_fkey" FOREIGN KEY (report\_id)  
 REFERENCES "Report" (report\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
 )  
 ''')  
  
 self.conn.commit()  
  
 def create\_table\_Observer(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Observer" (  
 observer\_id SERIAL PRIMARY KEY,  
 region\_id INTEGER NOT NULL,  
 first\_name TEXT NOT NULL,  
 last\_name TEXT NOT NULL,  
 address TEXT NOT NULL,  
 CONSTRAINT "Observer\_region\_id\_fkey" FOREIGN KEY (region\_id)  
 REFERENCES "Region" (region\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
 )  
 ''')  
  
 # Перевірка наявності таблиці  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Observer')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Якщо таблиця не існує, то створити її  
 c.execute('''  
 CREATE TABLE "Observer" (  
 observer\_id SERIAL PRIMARY KEY,  
 region\_id INTEGER NOT NULL,  
 first\_name TEXT NOT NULL,  
 last\_name TEXT NOT NULL,  
 address TEXT NOT NULL,  
 CONSTRAINT "Observer\_region\_id\_fkey" FOREIGN KEY (region\_id)  
 REFERENCES "Region" (region\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
 )  
 ''')  
  
 self.conn.commit()  
  
 def create\_table\_Region(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Region" (  
 region\_id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 area INTEGER NOT NULL  
 )  
 ''')  
  
 # Перевірка наявності таблиці  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Region')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Якщо таблиця не існує, то створити її  
 c.execute('''  
 CREATE TABLE "Region" (  
 region\_id SERIAL PRIMARY KEY,  
 name TEXT NOT NULL,  
 area INTEGER NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def create\_table\_Report(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Report" (  
 report\_id SERIAL PRIMARY KEY,  
 submission\_date DATE NOT NULL  
 )  
 ''')  
  
 # Перевірка наявності таблиці  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Report')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Якщо таблиця не існує, то створити її  
 c.execute('''  
 CREATE TABLE "Report" (  
 report\_id SERIAL PRIMARY KEY,  
 submission\_date DATE NOT NULL  
 )  
 ''')  
  
 self.conn.commit()  
  
 def create\_table\_Region\_Observer\_Report(self):  
 c = self.conn.cursor()  
 c.execute('''  
 CREATE TABLE IF NOT EXISTS "Region\_Observer\_Report" (  
 reg\_ob\_rep\_id SERIAL PRIMARY KEY,  
 region\_id INTEGER NOT NULL,  
 report\_id INTEGER NOT NULL,  
 observer\_id INTEGER NOT NULL,  
 date DATE NOT NULL,  
 CONSTRAINT "Region\_Observer\_Report\_observer\_id\_fkey" FOREIGN KEY (observer\_id)  
 REFERENCES "Observer" (observer\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION,  
 CONSTRAINT "Region\_Observer\_Report\_region\_id\_fkey" FOREIGN KEY (region\_id)  
 REFERENCES "Region" (region\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION,  
 CONSTRAINT "Region\_Observer\_Report\_report\_id\_fkey" FOREIGN KEY (report\_id)  
 REFERENCES "Report" (report\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
 )  
 ''')  
  
 # Check if the table exists  
 c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Region\_Observer\_Report')")  
 table\_exists = c.fetchone()[0]  
  
 if not table\_exists:  
 # Table does not exist, so create it  
 c.execute('''  
 CREATE TABLE "Region\_Observer\_Report" (  
 reg\_ob\_rep\_id SERIAL PRIMARY KEY,  
 region\_id INTEGER NOT NULL,  
 report\_id INTEGER NOT NULL,  
 observer\_id INTEGER NOT NULL,  
 date DATE NOT NULL,  
 CONSTRAINT "Region\_Observer\_Report\_observer\_id\_fkey" FOREIGN KEY (observer\_id)  
 REFERENCES "Observer" (observer\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION,  
 CONSTRAINT "Region\_Observer\_Report\_region\_id\_fkey" FOREIGN KEY (region\_id)  
 REFERENCES "Region" (region\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION,  
 CONSTRAINT "Region\_Observer\_Report\_report\_id\_fkey" FOREIGN KEY (report\_id)  
 REFERENCES "Report" (report\_id) MATCH SIMPLE  
 ON UPDATE NO ACTION  
 ON DELETE NO ACTION  
 )  
 ''')  
  
 self.conn.commit()  
  
 def add\_Indicator(self, Name, Unit\_of\_measurement):  
 c = self.conn.cursor()  
 try:  
 c.execute('INSERT INTO "Indicator" ("name", "unit\_of\_measurement") VALUES (%s, %s) RETURNING "indicator\_id"', (Name, Unit\_of\_measurement))  
 indicator\_id = c.fetchone()[0]  
 self.conn.commit()  
 return f"Indicator added successfully with ID: {indicator\_id}"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def get\_all\_Indicator(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM "Indicator"')  
 return c.fetchall()  
  
 def update\_Indicator(self, Indicator\_id, Name, Unit\_of\_measurement):  
 c = self.conn.cursor()  
 try:  
 c.execute('UPDATE "Indicator" SET "name"=%s, "unit\_of\_measurement"=%s WHERE "indicator\_id"=%s', (Name, Unit\_of\_measurement, Indicator\_id))  
 self.conn.commit()  
 return "Indicator updated successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def delete\_Indicator(self, Indicator\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM "Indicator" WHERE "indicator\_id"=%s', (Indicator\_id,))  
 self.conn.commit()  
 return "Indicator deleted successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def add\_Report(self, Name):  
 c = self.conn.cursor()  
 try:  
 c.execute('INSERT INTO "Report" ("Name", "submission\_date") VALUES (%s, CURRENT\_DATE) RETURNING "report\_id"', (Name,))  
 report\_id = c.fetchone()[0]  
 self.conn.commit()  
 return f"Report added successfully with ID: {report\_id}"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def get\_all\_Report(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM "Report"')  
 return c.fetchall()  
  
 def update\_Report(self, Report\_id, Name):  
 c = self.conn.cursor()  
 try:  
 c.execute('UPDATE "Report" SET "Name"=%s WHERE "report\_id"=%s', (Name, Report\_id))  
 self.conn.commit()  
 return "Report updated successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def delete\_Report(self, Report\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM "Report" WHERE "report\_id"=%s', (Report\_id,))  
 self.conn.commit()  
 return "Report deleted successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def add\_Region(self, Name, Area):  
 c = self.conn.cursor()  
 try:  
 c.execute('INSERT INTO "Region" ("Name", "area") VALUES (%s, %s) RETURNING "region\_id"', (Name, Area))  
 region\_id = c.fetchone()[0]  
 self.conn.commit()  
 return f"Region added successfully with ID: {region\_id}"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def get\_all\_Region(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM "Region"')  
 return c.fetchall()  
  
 def update\_Region(self, Region\_id, Name, Area):  
 c = self.conn.cursor()  
 try:  
 c.execute('UPDATE "Region" SET "Name"=%s, "area"=%s WHERE "region\_id"=%s', (Name, Area, Region\_id))  
 self.conn.commit()  
 return "Region updated successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def delete\_Region(self, Region\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM "Region" WHERE "region\_id"=%s', (Region\_id,))  
 self.conn.commit()  
 return "Region deleted successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def add\_Observer(self, Region\_id, First\_name, Last\_name, Address):  
 c = self.conn.cursor()  
 try:  
 c.execute(  
 'INSERT INTO "Observer" ("region\_id", "first\_name", "last\_name", "address") VALUES (%s, %s, %s, %s) RETURNING "observer\_id"',  
 (Region\_id, First\_name, Last\_name, Address)  
 )  
 observer\_id = c.fetchone()[0]  
 self.conn.commit()  
 return f"Observer added successfully with ID: {observer\_id}"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def get\_all\_Observer(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM "Observer"')  
 return c.fetchall()  
  
 def update\_Observer(self, Observer\_id, Region\_id, First\_name, Last\_name, Address):  
 c = self.conn.cursor()  
 try:  
 c.execute(  
 'UPDATE "Observer" SET "region\_id"=%s, "first\_name"=%s, "last\_name"=%s, "address"=%s WHERE "observer\_id"=%s',  
 (Region\_id, First\_name, Last\_name, Address, Observer\_id)  
 )  
 self.conn.commit()  
 return "Observer updated successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def delete\_Observer(self, Observer\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM "Observer" WHERE "observer\_id"=%s', (Observer\_id,))  
 self.conn.commit()  
 return "Observer deleted successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def add\_Indicator\_Report(self, report\_id, indicator\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute(  
 'INSERT INTO "Indicator\_Report" ("report\_id", "indicator\_id") VALUES (%s, %s) RETURNING "ind\_rep\_id"',  
 (report\_id, indicator\_id)  
 )  
 ind\_rep\_id = c.fetchone()[0]  
 self.conn.commit()  
 return f"Indicator\_Report added successfully with ID: {ind\_rep\_id}"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def get\_all\_Indicator\_Report(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM "Indicator\_Report"')  
 return c.fetchall()  
  
 def update\_Indicator\_Report(self, ind\_rep\_id, report\_id, indicator\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute(  
 'UPDATE "Indicator\_Report" SET "report\_id"=%s, "indicator\_id"=%s WHERE "ind\_rep\_id"=%s',  
 (report\_id, indicator\_id, ind\_rep\_id)  
 )  
 self.conn.commit()  
 return "Indicator\_Report updated successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def delete\_Indicator\_Report(self, ind\_rep\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM "Indicator\_Report" WHERE "ind\_rep\_id"=%s', (ind\_rep\_id,))  
 self.conn.commit()  
 return "Indicator\_Report deleted successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def add\_Region\_Observer\_Report(self, region\_id, report\_id, observer\_id, date):  
 c = self.conn.cursor()  
 try:  
 c.execute(  
 'INSERT INTO "Region\_Observer\_Report" ("region\_id", "report\_id", "observer\_id", "date") VALUES (%s, %s, %s, %s) RETURNING "reg\_ob\_rep\_id"',  
 (region\_id, report\_id, observer\_id, date)  
 )  
 reg\_ob\_rep\_id = c.fetchone()[0]  
 self.conn.commit()  
 return f"Region\_Observer\_Report added successfully with ID: {reg\_ob\_rep\_id}"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def get\_all\_Region\_Observer\_Report(self):  
 c = self.conn.cursor()  
 c.execute('SELECT \* FROM "Region\_Observer\_Report"')  
 return c.fetchall()  
  
 def update\_Region\_Observer\_Report(self, reg\_ob\_rep\_id, region\_id, report\_id, observer\_id, date):  
 c = self.conn.cursor()  
 try:  
 c.execute(  
 'UPDATE "Region\_Observer\_Report" SET "region\_id"=%s, "report\_id"=%s, "observer\_id"=%s, "date"=%s WHERE "reg\_ob\_rep\_id"=%s',  
 (region\_id, report\_id, observer\_id, date, reg\_ob\_rep\_id)  
 )  
 self.conn.commit()  
 return "Region\_Observer\_Report updated successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
  
 def delete\_Region\_Observer\_Report(self, reg\_ob\_rep\_id):  
 c = self.conn.cursor()  
 try:  
 c.execute('DELETE FROM "Region\_Observer\_Report" WHERE "reg\_ob\_rep\_id"=%s', (reg\_ob\_rep\_id,))  
 self.conn.commit()  
 return "Region\_Observer\_Report deleted successfully!"  
 except psycopg2.IntegrityError as e:  
 self.conn.rollback()  
 error\_message = str(e)  
 return "Error: " + error\_message  
 finally:  
 c.close()  
 c.close()  
  
 def generate\_indicator(self, number):  
 c = self.conn.cursor()  
 c.execute(  
 '''  
 DO $$  
 DECLARE  
 indicator\_id INT;  
 BEGIN  
 FOR indicator\_id IN 1..%s  
 LOOP  
 INSERT INTO public."Indicator" ("indicator\_id", "name", "unit\_of\_measurement")   
 VALUES (  
 nextval('"Indicator\_indicator\_id\_seq"'),  
 'Indicator' || currval('"Indicator\_indicator\_id\_seq"'),  
 'Unit' || currval('"Indicator\_indicator\_id\_seq"')  
 );  
 END LOOP;  
 END $$;  
 ''',  
 (number,)  
 )  
 self.conn.commit()  
  
 def generate\_region(self, number):  
 c = self.conn.cursor()  
 c.execute(  
 '''  
 DO $$  
 DECLARE  
 i INT;  
 BEGIN  
 FOR i IN 1..%s  
 LOOP  
 INSERT INTO public."Region" ("name", "area")   
 VALUES (  
 'Region' || i,  
 floor(random() \* 1000)  
 );  
 END LOOP;  
 END $$;  
 ''',  
 (number,)  
 )  
 self.conn.commit()  
  
 def generate\_report(self, number):  
 c = self.conn.cursor()  
 c.execute(  
 '''  
 DO $$  
 DECLARE  
 report\_id INT;  
 BEGIN  
 FOR report\_id IN 1..%s  
 LOOP  
 INSERT INTO public."Report" ("report\_id", "submission\_date")   
 VALUES (  
 nextval('"Report\_report\_id\_seq"'),  
 CURRENT\_DATE - (floor(random() \* 365) || ' days')::INTERVAL  
 );  
 END LOOP;  
 END $$;  
 ''',  
 (number,)  
 )  
 self.conn.commit()  
  
 def generate\_observer(self, number):  
 c = self.conn.cursor()  
 c.execute(  
 '''  
 DO $$  
 DECLARE  
 observer\_id INT;  
 region\_count INT;  
 BEGIN  
 SELECT COUNT(\*) FROM public."Region" INTO region\_count;  
  
 FOR observer\_id IN 1..%s  
 LOOP  
 INSERT INTO public."Observer" ("region\_id", "first\_name", "last\_name", "address")   
 VALUES (  
 (random() \* region\_count)::INT + 1,  
 'First' || observer\_id,  
 'Last' || observer\_id,  
 'Address' || observer\_id  
 );  
 END LOOP;  
 END $$;  
 ''',  
 (number,)  
 )  
 self.conn.commit()  
  
 def generate\_indicator\_report(self, number):  
 c = self.conn.cursor()  
 c.execute(  
 '''  
 INSERT INTO public."Indicator\_Report" ("indicator\_id", "report\_id")  
 SELECT i."indicator\_id", r."report\_id"  
 FROM public."Indicator" i  
 CROSS JOIN public."Report" r  
 LIMIT %s;  
 ''',  
 (number,)  
 )  
  
 self.conn.commit()  
  
 def generate\_observer\_region\_report(self, number):  
 c = self.conn.cursor()  
 c.execute(  
 '''  
 INSERT INTO public."Region\_Observer\_Report" ("region\_id", "report\_id", "observer\_id", "date")  
 SELECT r."region\_id", rep."report\_id", obs."observer\_id", CURRENT\_DATE - (CAST(random() \* 365 AS INTEGER))  
 FROM public."Region" r  
 CROSS JOIN public."Report" rep  
 CROSS JOIN public."Observer" obs  
 LIMIT %s;  
 ''',  
 (number,)  
 )  
  
 self.conn.commit()  
  
 def reset(self):  
 c = self.conn.cursor()  
 c.execute(  
 'DELETE FROM public."Indicator\_Report"; DELETE FROM public."Observer"; DELETE FROM public."Report"; DELETE FROM public."Region\_Observer\_Report"; DELETE FROM public."Region"; DELETE FROM public."Indicator"; SELECT setval(\'indicator\_id\', 0); SELECT setval(\'region\_id\', 0); SELECT setval(\'report\_id\', 0); SELECT setval(\'observer\_id\', 0); SELECT setval(\'reg\_ob\_rep\_id\', 0); SELECT setval(\'ind\_rep\_id\', 0);')  
 self.conn.commit()  
  
 def search\_request1(self):  
 c = self.conn.cursor()  
 c.execute('''  
 SELECT o.first\_name, o.last\_name  
 FROM public."Observer" o  
 JOIN public."Region" r ON o.region\_id = r.region\_id  
 JOIN public."Region\_Observer\_Report" ror ON r.region\_id = ror.region\_id  
 JOIN public."Report" rep ON ror.report\_id = rep.report\_id  
 JOIN public."Indicator\_Report" ir ON rep.report\_id = ir.report\_id  
 JOIN public."Indicator" i ON ir.indicator\_id = i.indicator\_id  
 WHERE r.area >= 500   
 AND rep.submission\_date BETWEEN '2023-01-01' AND '2023-12-31'  
 AND i.name LIKE '%specified\_indicator%'  
 GROUP BY o.first\_name, o.last\_name;  
 ''')  
 return c.fetchall(), c.description  
  
 def search\_request2(self):  
 c = self.conn.cursor()  
 c.execute('''  
 SELECT DISTINCT r.name  
 FROM public."Region" r  
 JOIN public."Region\_Observer\_Report" ror ON r.region\_id = ror.region\_id  
 JOIN public."Report" rep ON ror.report\_id = rep.report\_id  
 JOIN public."Indicator\_Report" ir ON rep.report\_id = ir.report\_id  
 JOIN public."Indicator" i ON ir.indicator\_id = i.indicator\_id  
 WHERE rep.submission\_date BETWEEN '2023-01-01' AND '2023-12-31'  
 AND i.unit\_of\_measurement LIKE '%specified\_unit%';  
 ''')  
 return c.fetchall(), c.description  
  
 def search\_request3(self):  
 c = self.conn.cursor()  
 c.execute('''SELECT o.first\_name, o.last\_name  
 FROM public."Observer" o  
 JOIN public."Region" r ON o.region\_id = r.region\_id  
 JOIN public."Region\_Observer\_Report" ror ON r.region\_id = ror.region\_id  
 JOIN public."Report" rep ON ror.report\_id = rep.report\_id  
 JOIN public."Indicator\_Report" ir ON rep.report\_id = ir.report\_id  
 JOIN public."Indicator" i ON ir.indicator\_id = i.indicator\_id  
 WHERE r.area >= 300 AND i.name LIKE '%specified\_indicator%'  
 AND ror.date BETWEEN '2023-01-01' AND '2023-12-31';  
 ''')  
 return c.fetchall(), c.description

Модуль view.py

class View:  
  
 def show\_Indicator(self, Indicator):  
 print("Indicators:")  
 for indicator in Indicator:  
 print(f"Indicator ID: {indicator[0]}, Name: {indicator[1]}, Unit of Measurement: {indicator[2]}")  
  
 def show\_Report(self, Report):  
 print("Reports:")  
 for report in Report:  
 print(f"Report ID: {report[0]}, Submission Date: {report[1]}")  
  
 def show\_Region(self, Region):  
 print("Regions:")  
 for region in Region:  
 print(f"Region ID: {region[0]}, Name: {region[1]}, Area: {region[2]}")  
  
 def show\_Observer(self, Observer):  
 print("Observers:")  
 for observer in Observer:  
 print(f"Observer ID: {observer[0]}, Region ID: {observer[1]}, First Name: {observer[2]}, "  
 f"Last Name: {observer[3]}, Address: {observer[4]}")  
  
 def show\_Indicator\_Report(self, Indicator\_Report):  
 print("Indicator Reports:")  
 for indicator\_Report in Indicator\_Report:  
 print(f"Indicator\_Report ID: {indicator\_Report[0]}, Report ID: {indicator\_Report[1]}, "  
 f"Indicator ID: {indicator\_Report[2]}")  
  
 def show\_Region\_Observer\_Report(self, Region\_Observer\_Report):  
 print("Region Observer Reports:")  
 for region\_Observer\_Report in Region\_Observer\_Report:  
 print(f"Region\_Observer\_Report ID: {region\_Observer\_Report[0]}, Region ID: {region\_Observer\_Report[1]}, "  
 f"Report ID: {region\_Observer\_Report[2]}, Observer ID: {region\_Observer\_Report[3]}, "  
 f"Date: {region\_Observer\_Report[4]}")  
  
 def get\_Indicator\_input(self):  
 Name = input("Enter Name: ")  
 Unit\_of\_measurement = input("Enter Unit of Measurement: ")  
 return Name, Unit\_of\_measurement  
  
 def get\_Indicator\_update(self):  
 Name = input("Enter Name: ")  
 Unit\_of\_measurement = input("Enter Unit of Measurement: ")  
 return Name, Unit\_of\_measurement  
  
 def get\_Report\_input(self):  
 Submission\_date = input("Enter Submission Date (YYYY-MM-DD): ")  
 return Submission\_date  
  
 def get\_Report\_update(self):  
 Submission\_date = input("Enter Submission Date (YYYY-MM-DD): ")  
 return Submission\_date  
  
 def get\_Region\_input(self):  
 Name = input("Enter Name: ")  
 Area = input("Enter Area: ")  
 return Name, Area  
  
 def get\_Region\_update(self):  
 Name = input("Enter Name: ")  
 Area = input("Enter Area: ")  
 return Name, Area  
  
 def get\_Observer\_input(self):  
 Region\_id = input("Enter Region ID: ")  
 First\_name = input("Enter First Name: ")  
 Last\_name = input("Enter Last Name: ")  
 Address = input("Enter Address: ")  
 return Region\_id, First\_name, Last\_name, Address  
  
 def get\_Observer\_update(self):  
 Region\_id = input("Enter Region ID: ")  
 First\_name = input("Enter First Name: ")  
 Last\_name = input("Enter Last Name: ")  
 Address = input("Enter Address: ")  
 return Region\_id, First\_name, Last\_name, Address  
  
 def get\_Indicator\_Report\_input(self):  
 Report\_id = input("Enter Report ID: ")  
 Indicator\_id = input("Enter Indicator ID: ")  
 return Report\_id, Indicator\_id  
  
 def get\_Indicator\_Report\_update(self):  
 Report\_id = input("Enter Report ID: ")  
 Indicator\_id = input("Enter Indicator ID: ")  
 return Report\_id, Indicator\_id  
  
 def get\_Region\_Observer\_Report\_input(self):  
 Region\_id = input("Enter Region ID: ")  
 Report\_id = input("Enter Report ID: ")  
 Observer\_id = input("Enter Observer ID: ")  
 Date = input("Enter Date (YYYY-MM-DD): ")  
 return Region\_id, Report\_id, Observer\_id, Date  
  
 def get\_Region\_Observer\_Report\_update(self):  
 Region\_id = input("Enter Region ID: ")  
 Report\_id = input("Enter Report ID: ")  
 Observer\_id = input("Enter Observer ID: ")  
 Date = input("Enter Date (YYYY-MM-DD): ")  
 return Region\_id, Report\_id, Observer\_id, Date  
  
 def get\_id(self):  
 return input("Enter ID: ")  
  
 def get\_number\_for\_generation(self):  
 return input("Enter number: ")  
  
 def format\_and\_print\_results(self, results, column\_description):  
 if not results:  
 self.show\_message("No results found.")  
 return  
  
 column\_names = [description[0] for description in column\_description]  
 max\_lengths = [len(name) for name in column\_names]  
 for row in results:  
 max\_lengths = [max(max\_lengths[i], len(str(value)) if value is not None else 0) for i, value in  
 enumerate(row)]  
  
 header = "|".join([name.ljust(max\_lengths[i]) for i, name in enumerate(column\_names)])  
 self.show\_message(header)  
  
 for row in results:  
 formatted\_row = "|".join(  
 [str(value).ljust(max\_lengths[i]) if value is not None else 'None'.ljust(max\_lengths[i]) for i, value in  
 enumerate(row)])  
 self.show\_message(formatted\_row)  
  
 def show\_message(self, message):  
 print(message)

**Скріншоти результатів виконання операції вставки запису в дочірню таблицю**

**Изображение выглядит как текст, снимок экрана

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

**Скріншоти результатів виконання операції редагування таблиці**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание** **Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

**Скріншоти результатів виконання операції видалення**

**Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание**

**Скріншоти результатів виконання операції генерування**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание**

**Скріншоти результатів виконання операції пошуку**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

<https://github.com/MrMoPo3/BDlabs/blob/main>