

Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут»

**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 III курсу

групи: КВ-22

Кобан І.С.

Перевірив:

Павловський В. І

**Київ 2024**

## Хід роботи

### Опис сутностей предметної області

- **Дослідник (Researcher)**

Атрибути: ідентифікатор дослідника, ім'я, прізвище, спеціалізація, електронна пошта.

Призначення: збереження даних щодо дослідників, які беруть участь у наукових проєктах, статтях та експериментах.

- **Дослідницький проєкт (Research Project)**

Атрибути: ідентифікатор проєкту, назва, дата початку, дата завершення, опис проєкту.

Призначення: збереження інформації про наукові дослідження, їх тривалість і цілі.

- **Експеримент (Experiment)**

Атрибути: ідентифікатор експерименту, назва, опис та дата проведення.

Призначення: збереження даних про експерименти, які здійснювались в ході дослідницького проєкту.

- **Публікація (Publication)**

Атрибути: ідентифікатор публікації, назва, дата публікації, назва журналу.

Призначення: збереження даних про наукові публікації пов'язані з дослідженням.

### Опис зв'язків між сутностями предметної області

Зв'язок «Дослідник» - «Дослідницький проєкт» є зв'язком N:M. Один дослідник може брати участь у кількох проєктах або не брати в жодному, і один проєкт може залучати кількох дослідників або не мати жодного (на початковій стадії).

Зв'язок «Дослідник» - «Публікація» є зв'язком N:M. Один дослідник може бути автором або багатьох публікацій або жодної, і кожна публікація повинна мати принаймі одного автора (дослідника).

Зв'язок «Дослідник» - «Експеримент» є зв'язком N:M. Один дослідник може брати участь або у багатьох експериментах або у жодному, і кожен експеримент повинен мати принаймі одного виконавця (дослідника).

Зв'язок «Дослідницький проєкт» - «Публікація» є зв'язком 1:N. Один проєкт може мати багато публікацій або не мати жодної, якщо дослідження ще триває, і кожна публікація належить лише до одного дослідницького проєкту.

Зв'язок «Дослідницький проєкт» - «Експеримент» є зв'язком 1:N. Один проєкт може мати багато експериментів в ході виконання дослідження або не мати жодного, якщо це теоретичне, або наприклад історичне дослідження, і кожен експеримент належить лише до одного дослідницького проєкту.

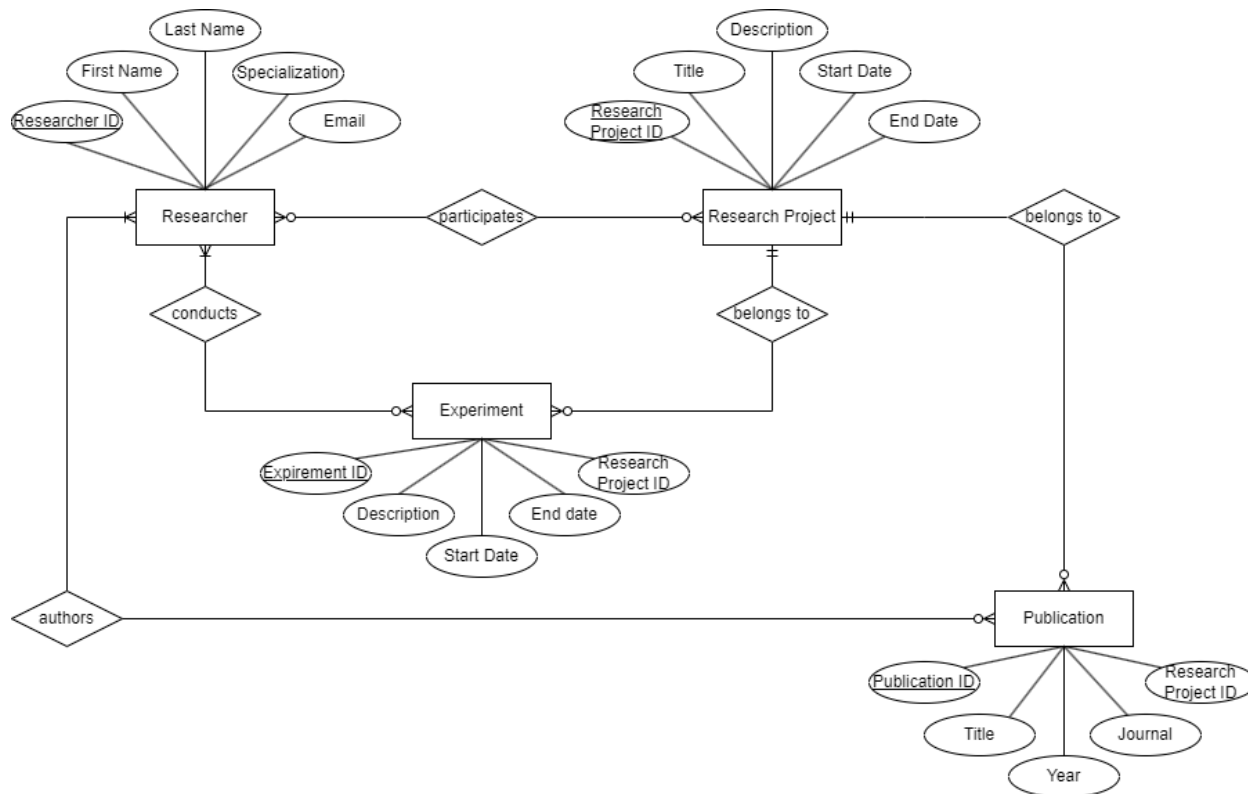


Рисунок 1 – діаграма ER-моделі, побудована за нотацією Чена

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

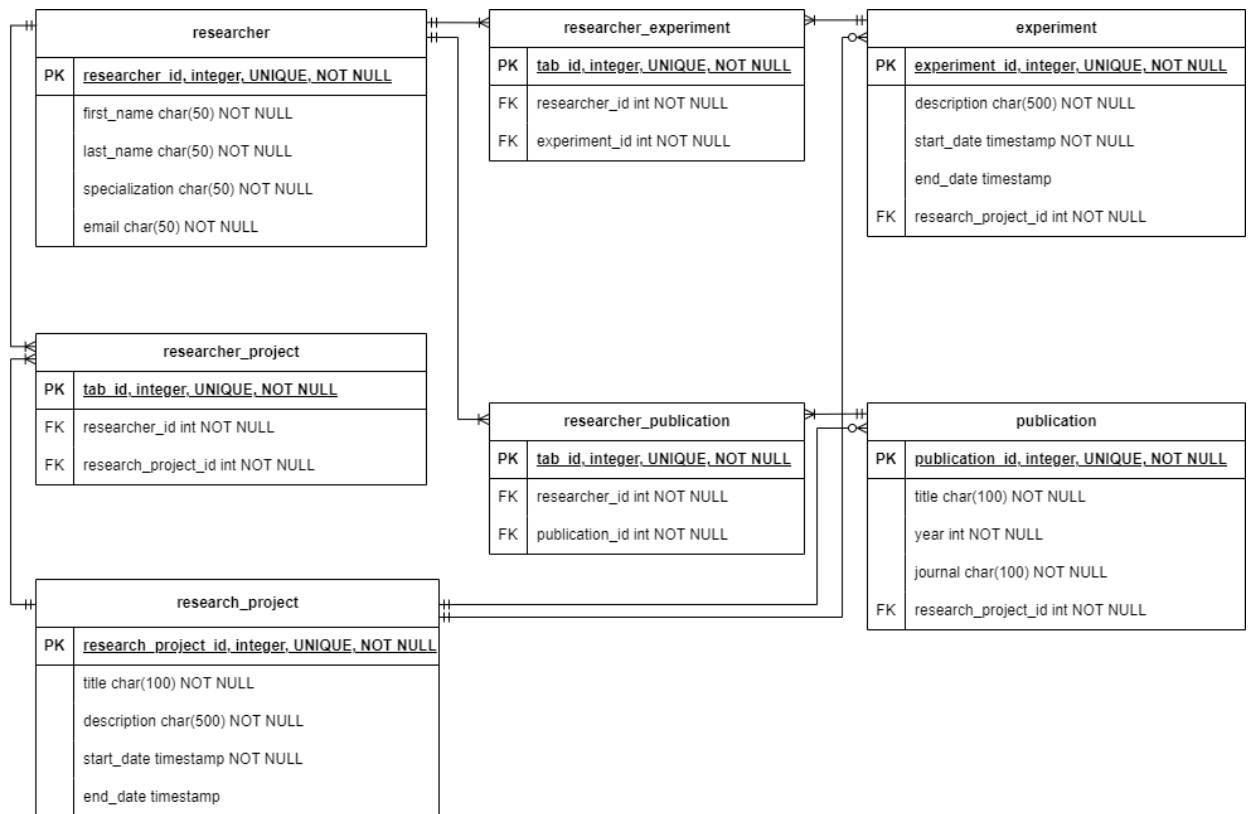


Рисунок 2 – Логічна модель «Сутність-зв'язок»

## Середовище та компоненти розробки

У процесі розробки було використано мову програмування Python та інтегроване середовище розробки Visual Studio Code. Для взаємодії з базою даних PostgreSQL застосовувалася бібліотека `psycopg2`, яка забезпечує зручний API для виконання SQL-запитів.

## Структура програми та її опис

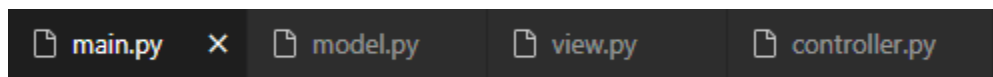


Рисунок 3 – структура програми

У файлі `main.py` відбувається ініціалізація та запуск контролера. Цей файл є точкою входу до програми, де створюється екземпляр класу `Controller` та викликається його метод `run()`, який керує потоком виконання програми.

У файлі `model.py` описаний клас `Model`, який відповідає за управління підключенням до бази даних та виконанням низькорівневих запитів до неї. Модель забезпечує доступ до даних, виконує операції додавання, оновлення, видалення та генерації даних, а також відповідає за валідацію введених користувачем даних.

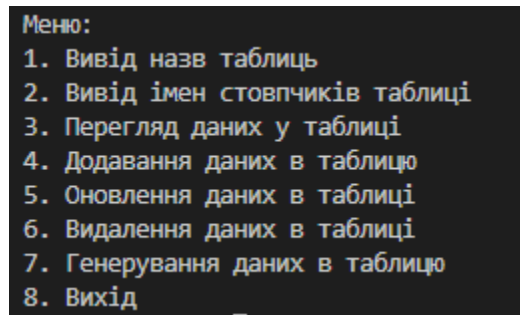
У файлі `controller.py` реалізовано клас `Controller`, який відповідає за управління взаємодією між моделлю та представленням. Контролер обробляє запити користувача, викликає відповідні методи моделі для виконання необхідних операцій та оновлює представлення на основі результатів цих операцій.

У файлі `view.py` описаний клас `View`, який відповідає за відображення результатів виконання різних дій користувача на екрані консолі. Цей компонент забезпечує зручний інтерфейс для взаємодії користувача з програмою, відображаючи меню, результати операцій та повідомлення про успіх або помилки.

Отже, структура програми чітко відповідає шаблону MVC.

### Структура меню програми

На рисунку 4 зображено меню користувача, яке складається із восьми пунктів.



```
Меню:  
1. Вивід назв таблиць  
2. Вивід імен стовпчиків таблиці  
3. Перегляд даних у таблиці  
4. Додавання даних в таблицю  
5. Оновлення даних в таблиці  
6. Видалення даних в таблиці  
7. Генерування даних в таблицю  
8. Вихід
```

Рисунок 4 – структура меню користувача

Програма надає користувачу інтуїтивно зрозуміле текстове меню для взаємодії з базою даних. Нижче наведено опис функціональності кожного пункту меню, зосереджуючись на конкретних методах, що їх реалізують.

**Фрагмент коду (controller.py) в якому реалізовано головний цикл програми з вибором пункту меню**

```
def run(self):
    while True:
        choice = self.view.display_menu()
        if choice == '1':
            self.view_tables()
        elif choice == '2':
            self.view_columns()
        elif choice == '3':
            self.view_table_data()
        elif choice == '4':
            self.add_data()
        elif choice == '5':
            self.update_data()
        elif choice == '6':
            self.delete_data()
        elif choice == '7':
            self.generate_data()
        elif choice == '8':
            self.model.close_connection()
            self.view.display_message("Вихід з програми.")
            break
        else:
            self.view.display_message("Невірний вибір. Спробуйте ще раз.")
```

**Фрагмент коду (файл model.py) в якому наведено реалізацію методів додавання, оновлення, видалення та генерації даних в базі даних**

```
def insert_data(self, table_name, columns, values):
    try:
        if len(columns) != len(values):
            raise ValueError("Кількість стовпців не відповідає кількості значень.")

        for column, value in zip(columns, values):
            if column == f'{table_name.lower()}_id':
                try:
                    value = int(value)
                except ValueError:
                    raise ValueError(f"Значення ідентифікатора повинно бути цілим числом для стовпця {column}.")
```

```

        self.cursor.execute(f'SELECT {column} FROM {table_name}')
        existing_identifiers = [item for sublist in
self.cursor.fetchall() for item in sublist]
        if value in existing_identifiers:
            raise ValueError("Ідентифікатор вже існує.")
            elif column.endswith('_id') and column !=
f'{table_name.lower()}_id':
                try:
                    value = int(value)
                except ValueError:
                    raise ValueError(f"Значення для {column} повинно
бути цілим числом.")

        referenced_table = column[:-3]
        self.cursor.execute(f'SELECT {referenced_table}_id FROM
{referenced_table}')
        referenced_values = [item for sublist in
self.cursor.fetchall() for item in sublist]
        if value not in referenced_values:
            raise ValueError(f"Значення зовнішнього ключа для
{column} не існує.")

        columns_str = ', '.join(columns)
        placeholders = ', '.join(['%s'] * len(values))
        query = f"INSERT INTO {table_name} ({columns_str}) VALUES
({placeholders})"
        self.cursor.execute(query, values)
        self.connection.commit()
    except IntegrityError as e:
        self.connection.rollback()
        raise ValueError(f"Помилка вставки даних (можливо, порушення
обмежень цілісності): {e}")
    except Exception as e:
        self.connection.rollback()
        raise e

```

Дана функція відповідає за додавання нового запису в обрану таблицю. Вона приймає введені користувачем значення для необхідних стовпчиків та перевіряє їх на коректність. Зокрема, функція перевіряє, чи є введений ідентифікатор унікальним, а також чи існують всі необхідні зовнішні ключі у відповідних зв'язаних таблицях. Якщо всі перевірки проходять успішно, дані передаються моделі для вставки в базу даних. У випадку успішної вставки користувач отримує підтвердження, а при виникненні помилок – відповідні повідомлення про природу помилки.

```

def update_data(self, table_name, column, row_id, new_value):
    identifier_column = f'{table_name.lower()}_id'
    is_unique_identifier = identifier_column == column

    if is_unique_identifier:
        try:
            val_id = int(new_value)
        except ValueError:
            raise ValueError("Значення ідентифікатора повинно бути цілим
числом.")

        self.cursor.execute(f'SELECT {identifier_column} FROM
{table_name}')
        existing_identifiers = [item for sublist in
self.cursor.fetchall() for item in sublist]
        if val_id in existing_identifiers:
            raise ValueError("Ідентифікатор вже існує.")
    elif column.endswith('_id') and column != f'{table_name.lower()}_id':
        try:
            val_id = int(new_value)
        except ValueError:
            raise ValueError(f"Значення для {column} повинно бути цілим
числом.")

        referenced_table = column[:-3]
        self.cursor.execute(f'SELECT {referenced_table}_id FROM
{referenced_table}')
        referenced_values = [item for sublist in self.cursor.fetchall()
for item in sublist]
        if val_id not in referenced_values:
            raise ValueError(f"Значення зовнішнього ключа для {column}
не існує.")

        try:
            query = f"UPDATE {table_name} SET {column} = %s WHERE
{identifier_column} = %s"
            self.cursor.execute(query, (new_value, row_id))
            if self.cursor.rowcount == 0:
                raise ValueError(f"Рядок з id {row_id} не знайдено в таблиці
{table_name}.")
            self.connection.commit()
        except IntegrityError as e:
            self.connection.rollback()
            raise ValueError(f"Помилка оновлення даних (можливо, порушення
обмежень цілісності): {e}")

```



```

except Exception as e:
    self.connection.rollback()
    raise e

```

Ця функція дозволяє змінювати існуючий запис в обраній таблиці. Користувач вводить ідентифікатор рядка, який потрібно оновити, а також нові значення для вибраних стовпчиків. Функція перевіряє, чи існує вказаний рядок у таблиці та чи відповідають нові дані вимогам (наприклад, унікальність ідентифікатора, наявність валідних зовнішніх ключів). Після проходження всіх перевірок, оновлені дані передаються моделі для виконання операції оновлення в базі даних. Якщо оновлення проходить успішно, користувач отримує повідомлення про успіх, а у випадку помилок – відповідне повідомлення з описом проблеми.

```

def delete_data(self, table_name, row_id):
    try:
        identifier_column = f'{table_name.lower()}_id'
        row_id = int(row_id)
        query = f"DELETE FROM {table_name} WHERE {identifier_column} =
%s"

        self.cursor.execute(query, (row_id,))
        if self.cursor.rowcount == 0:
            raise ValueError(f"Рядок з id {row_id} не знайдено в таблиці
{table_name}.")
        self.connection.commit()
    except ValueError as ve:
        self.connection.rollback()
        raise ve
    except IntegrityError as e:
        self.connection.rollback()
        raise ValueError(f"Помилка видалення даних (можливо, є залежні
записи): {e}")
    except Exception as e:
        self.connection.rollback()
        raise e

```

Функція відповідає за видалення конкретного запису з обраної таблиці. Користувач вводить ідентифікатор рядка, який необхідно видалити. Перед виконанням операції видалення функція перевіряє, чи існує цей рядок у таблиці та чи немає залежних записів у інших таблицях, що можуть порушити цілісність даних (через зовнішні ключі). Якщо всі умови виконані, запит на видалення передається моделі, яка здійснює видалення запису з бази даних.

Після успішного видалення користувач отримує підтвердження, а у випадку помилок – відповідне повідомлення про причину невдалого видалення.

```
def generate_data(self, table_name, count):
    try:
        self.cursor.execute(
            """
            SELECT column_name, data_type
            FROM information_schema.columns
            WHERE table_name = %s AND table_schema = 'public'
            """,
            (table_name.lower(),)
        )
        columns_info = self.cursor.fetchall()

        if not columns_info:
            raise ValueError(f"Таблиця {table_name} не має стовпців або
неправильна назва таблиці.")

        self.cursor.execute(
            """
            SELECT kcu.column_name
            FROM information_schema.table_constraints tc
            JOIN information_schema.key_column_usage kcu
            ON tc.constraint_name = kcu.constraint_name
            AND tc.table_schema = kcu.table_schema
            WHERE tc.constraint_type = 'PRIMARY KEY'
            AND tc.table_name = %s
            AND tc.table_schema = 'public'
            """,
            (table_name.lower(),)
        )
        pk_info = self.cursor.fetchone()
        if pk_info:
            id_column = pk_info[0]
        else:
            raise ValueError(f"Таблиця {table_name} не має первинного
ключа.")

        for _ in range(count):
            insert_query = f'INSERT INTO {table_name} ('
            select_subquery = ""

            for column_info in columns_info:
                column_name = column_info[0]
```

```

        column_type = column_info[1]

        if column_name == id_column:
            select_subquery += f"(SELECT
COALESCE(MAX({id_column}), 0) + 1 FROM {table_name}),"
            elif column_name.endswith('_id') and column_name !=
id_column:
                related_table_name = column_name[:-3]
                select_subquery += f"(SELECT {related_table_name}_id
FROM {related_table_name} ORDER BY RANDOM() LIMIT 1),"
                elif column_type == 'integer':
                    if column_name.lower() == 'year':
                        select_subquery += '(2000 + FLOOR(RANDOM() *
100)), '
                    else:
                        select_subquery += 'FLOOR(RANDOM() * 100 + 1),'
                elif column_type in ['character varying', 'varchar']:
                    select_subquery += f"Random {column_name} ' ||
substr(md5(random()::text), 1, 5),"
                elif column_type == 'date':
                    if column_name == 'end_date':
                        select_subquery += "current_date + (FLOOR(RANDOM()
* 365))::int,"
                    else:
                        select_subquery += "current_date - (FLOOR(RANDOM()
* 365))::int,"
                elif column_type == 'timestamp with time zone':
                    if column_name == 'end_date':
                        select_subquery += "NOW() + (FLOOR(RANDOM() *
365) || ' days')::interval,"
                    else:
                        select_subquery += "NOW() - (FLOOR(RANDOM() *
365) || ' days')::interval,"
                    else:
                        select_subquery += 'NULL,'

            insert_query += f'{column_name},'

            insert_query = insert_query.rstrip(',') + f') VALUES
({select_subquery.rstrip(",")})'
            self.cursor.execute(insert_query)

        self.connection.commit()
    except Exception as e:
        self.connection.rollback()

```

`raise e`

Ця функція автоматично створює довільні записи для обраної таблиці. Користувач вводить назву таблиці та кількість записів, які потрібно згенерувати. Функція перевіряє коректність введених даних та викликає модель для створення та вставки випадкових записів у базу даних. Під час генерації даних забезпечується унікальність ідентифікаторів та відповідність зовнішніх ключів існуючим записам у зв'язаних таблицях. Після успішного завершення процесу користувач отримує повідомлення про кількість успішно створених записів або відповідне повідомлення про помилки що виникли.

### Повний код програми

#### Файл main.py:

```
from controller import Controller

if __name__ == "__main__":
    controller = Controller()
    controller.run()
```

#### Файл model.py:

```
import psycopg2
from psycopg2 import OperationalError, IntegrityError
import random

class Model:
    def __init__(self, db_name, user, password, host='localhost',
port='5432'):
        try:
            self.connection = psycopg2.connect(
                dbname=db_name, user=user, password=password, host=host,
port=port
            )
            self.cursor = self.connection.cursor()
        except OperationalError as e:
            raise OperationalError(f"Помилка підключення до бази даних:
{e}")

    def list_tables(self):
        try:
```

```

        self.cursor.execute("SELECT table_name FROM
information_schema.tables WHERE table_schema='public'")
        return self.cursor.fetchall()
    except Exception as e:
        raise Exception(f"Помилка отримання таблиць: {e}")

    def list_columns(self, table_name):
        try:
            self.cursor.execute(
                "SELECT column_name FROM information_schema.columns WHERE
table_name=%s AND table_schema='public'",
                (table_name.lower(),)
            )
            return self.cursor.fetchall()
        except Exception as e:
            raise Exception(f"Помилка отримання стовпчиків: {e}")

    def view_table_data(self, table_name):
        try:
            identifier_column = f'{table_name.lower()}_id'
            self.cursor.execute(f"SELECT * FROM {table_name} ORDER BY
{identifier_column}")
            return self.cursor.fetchall()
        except Exception as e:
            self.connection.rollback()
            raise Exception(f"Помилка перегляду даних таблиці: {e}")

    def insert_data(self, table_name, columns, values):
        try:
            if len(columns) != len(values):
                raise ValueError("Кількість стовпців не відповідає кількості
значень.")

            for column, value in zip(columns, values):
                if column == f'{table_name.lower()}_id':
                    try:
                        value = int(value)
                    except ValueError:
                        raise ValueError(f"Значення ідентифікатора повинно
бути цілим числом для стовпця {column}.")

            self.cursor.execute(f'SELECT {column} FROM {table_name}')
            existing_identifiers = [item for sublist in
self.cursor.fetchall() for item in sublist]
            if value in existing_identifiers:

```

```

        raise ValueError("Ідентифікатор вже існує.")
        elif column.endswith('_id') and column !=
f'{table_name.lower()}_id':
        try:
            value = int(value)
        except ValueError:
            raise ValueError(f"Значення для {column} повинно
бути цілим числом.")

        referenced_table = column[:-3]
        self.cursor.execute(f'SELECT {referenced_table}_id FROM
{referenced_table}')
        referenced_values = [item for sublist in
self.cursor.fetchall() for item in sublist]
        if value not in referenced_values:
            raise ValueError(f"Значення зовнішнього ключа для
{column} не існує.")

        columns_str = ', '.join(columns)
        placeholders = ', '.join(['%s'] * len(values))
        query = f"INSERT INTO {table_name} ({columns_str}) VALUES
({placeholders})"
        self.cursor.execute(query, values)
        self.connection.commit()
    except IntegrityError as e:
        self.connection.rollback()
        raise ValueError(f"Помилка вставки даних (можливо, порушення
обмежень цілісності): {e}")
    except Exception as e:
        self.connection.rollback()
        raise e

def update_data(self, table_name, column, row_id, new_value):
    identifier_column = f'{table_name.lower()}_id'
    is_unique_identifier = identifier_column == column

    if is_unique_identifier:
        try:
            val_id = int(new_value)
        except ValueError:
            raise ValueError("Значення ідентифікатора повинно бути цілим
числом.")

        self.cursor.execute(f'SELECT {identifier_column} FROM
{table_name}')

```

```

        existing_identifiers = [item for sublist in
self.cursor.fetchall() for item in sublist]
        if val_id in existing_identifiers:
            raise ValueError("Ідентифікатор вже існує.")
        elif column.endswith('_id') and column != f'{table_name.lower()}_id':
            try:
                val_id = int(new_value)
            except ValueError:
                raise ValueError(f"Значення для {column} повинно бути цілим
числом.")

        referenced_table = column[:-3]
        self.cursor.execute(f'SELECT {referenced_table}_id FROM
{referenced_table}')
        referenced_values = [item for sublist in self.cursor.fetchall()
for item in sublist]
        if val_id not in referenced_values:
            raise ValueError(f"Значення зовнішнього ключа для {column}
не існує.")

        try:
            query = f"UPDATE {table_name} SET {column} = %s WHERE
{identifier_column} = %s"
            self.cursor.execute(query, (new_value, row_id))
            if self.cursor.rowcount == 0:
                raise ValueError(f"Рядок з id {row_id} не знайдено в таблиці
{table_name}.")
            self.connection.commit()
        except IntegrityError as e:
            self.connection.rollback()
            raise ValueError(f"Помилка оновлення даних (можливо, порушення
обмежень цілісності): {e}")
        except Exception as e:
            self.connection.rollback()
            raise e

    def delete_data(self, table_name, row_id):
        try:
            identifier_column = f'{table_name.lower()}_id'
            row_id = int(row_id)
            query = f"DELETE FROM {table_name} WHERE {identifier_column} =
%s"

            self.cursor.execute(query, (row_id,))
            if self.cursor.rowcount == 0:

```

```

        raise ValueError(f"Рядок з id {row_id} не знайдено в таблиці
{table_name}.".")
        self.connection.commit()
    except ValueError as ve:
        self.connection.rollback()
        raise ve
    except IntegrityError as e:
        self.connection.rollback()
        raise ValueError(f"Помилка видалення даних (можливо, є залежні
записи): {e}")
    except Exception as e:
        self.connection.rollback()
        raise e

def generate_data(self, table_name, count):
    try:
        self.cursor.execute(
            """
            SELECT column_name, data_type
            FROM information_schema.columns
            WHERE table_name = %s AND table_schema = 'public'
            """,
            (table_name.lower(),)
        )
        columns_info = self.cursor.fetchall()

        if not columns_info:
            raise ValueError(f"Таблиця {table_name} не має стовпців або
неправильна назва таблиці.")

        self.cursor.execute(
            """
            SELECT kcu.column_name
            FROM information_schema.table_constraints tc
            JOIN information_schema.key_column_usage kcu
            ON tc.constraint_name = kcu.constraint_name
            AND tc.table_schema = kcu.table_schema
            WHERE tc.constraint_type = 'PRIMARY KEY'
            AND tc.table_name = %s
            AND tc.table_schema = 'public'
            """,
            (table_name.lower(),)
        )
        pk_info = self.cursor.fetchone()
        if pk_info:

```



```

        id_column = pk_info[0]
    else:
        raise ValueError(f"Таблиця {table_name} не має первинного
ключа.")

    for _ in range(count):
        insert_query = f'INSERT INTO {table_name} ('
        select_subquery = ""

        for column_info in columns_info:
            column_name = column_info[0]
            column_type = column_info[1]

            if column_name == id_column:
                select_subquery += f"(SELECT
COALESCE(MAX({id_column}), 0) + 1 FROM {table_name}),"
            elif column_name.endswith('_id') and column_name !=
id_column:
                related_table_name = column_name[:-3]
                select_subquery += f"(SELECT {related_table_name}_id
FROM {related_table_name} ORDER BY RANDOM() LIMIT 1),"
            elif column_type == 'integer':
                if column_name.lower() == 'year':
                    select_subquery += '(2000 + FLOOR(RANDOM() *
100)), '
                else:
                    select_subquery += 'FLOOR(RANDOM() * 100 + 1), '
            elif column_type in ['character varying', 'varchar']:
                select_subquery += f"Random {column_name} ' ||
substr(md5(random())::text, 1, 5),"
            elif column_type == 'date':
                if column_name == 'end_date':
                    select_subquery += "current_date + (FLOOR(RANDOM()
* 365))::int,"
                else:
                    select_subquery += "current_date - (FLOOR(RANDOM()
* 365))::int,"
            elif column_type == 'timestamp with time zone':
                if column_name == 'end_date':
                    select_subquery += "NOW() + (FLOOR(RANDOM() *
365) || ' days')::interval,"
                else:
                    select_subquery += "NOW() - (FLOOR(RANDOM() *
365) || ' days')::interval,"
            else:

```

```

        select_subquery += 'NULL, '

        insert_query += f'{column_name}, '

        insert_query = insert_query.rstrip(',') + f') VALUES
({select_subquery.rstrip(",")})'
        self.cursor.execute(insert_query)

        self.connection.commit()
    except Exception as e:
        self.connection.rollback()
        raise e

def close_connection(self):
    try:
        self.cursor.close()
        self.connection.close()
    except Exception as e:
        raise Exception(f"Помилка закриття підключення: {e}")

```

#### Файл view.py:

```

class View:
    def display_menu(self):
        print("\nМеню:")
        print("1. Вивід назв таблиць")
        print("2. Вивід імен стовпчиків таблиці")
        print("3. Перегляд даних у таблиці")
        print("4. Додавання даних в таблицю")
        print("5. Оновлення даних в таблиці")
        print("6. Видалення даних в таблиці")
        print("7. Генерування даних в таблицю")
        print("8. Вихід")
        return input("Оберіть опцію: ")

    def prompt(self, message):
        return input(message)

    def display_result(self, result):
        if not result:
            print("Немає даних для відображення.")
        else:
            for row in result:
                print(row)

```

```
def display_message(self, message):  
    print(message)
```

Файл controller.py:

```
from model import Model  
from view import View  
  
class Controller:  
    def __init__(self):  
        self.model = Model(db_name="postgres", user="postgres",  
password="root")  
        self.view = View()  
  
    def run(self):  
        while True:  
            choice = self.view.display_menu()  
            if choice == '1':  
                self.view_tables()  
            elif choice == '2':  
                self.view_columns()  
            elif choice == '3':  
                self.view_table_data()  
            elif choice == '4':  
                self.add_data()  
            elif choice == '5':  
                self.update_data()  
            elif choice == '6':  
                self.delete_data()  
            elif choice == '7':  
                self.generate_data()  
            elif choice == '8':  
                self.model.close_connection()  
                self.view.display_message("Вихід з програми.")  
                break  
            else:  
                self.view.display_message("Невірний вибір. Спробуйте ще  
раз.")  
  
    def view_tables(self):  
        try:  
            tables = self.model.list_tables()  
            self.view.display_result(tables)
```

```

        except Exception as e:
            self.view.display_message(f"Помилка при отриманні списку таблиць:
{e}")

    def view_columns(self):
        table_name = self.view.prompt("Введіть назву таблиці: ")
        try:
            columns = self.model.list_columns(table_name)
            self.view.display_result(columns)
        except Exception as e:
            self.view.display_message(f"Помилка при отриманні стовпців
таблиці {table_name}: {e}")

    def view_table_data(self):
        table_name = self.view.prompt("Введіть назву таблиці: ")
        try:
            data = self.model.view_table_data(table_name)
            self.view.display_result(data)
        except Exception as e:
            self.view.display_message(f"Помилка при перегляді даних таблиці
{table_name}: {e}")

    def add_data(self):
        table_name = self.view.prompt("Введіть назву таблиці: ")
        columns_input = self.view.prompt("Введіть назви стовпців через кому: ")
        values_input = self.view.prompt("Введіть значення через кому: ")

        columns = [col.strip() for col in columns_input.split(',')]
        values = [val.strip() for val in values_input.split(',')]

        try:
            self.model.insert_data(table_name, columns, values)
            self.view.display_message("Дані додано успішно.")
        except Exception as e:
            self.view.display_message(f"Помилка додавання даних: {e}")

    def update_data(self):
        table_name = self.view.prompt("Введіть назву таблиці: ")
        column = self.view.prompt("Введіть назву стовпчика для оновлення: ")

        new_value = self.view.prompt("Введіть нове значення: ")
        row_id = self.view.prompt("Введіть номер рядка: ")

        try:

```

```

        self.model.update_data(table_name, column, row_id, new_value)
        self.view.display_message("Дані оновлено успішно.")
    except Exception as e:
        self.view.display_message(f"Помилка оновлення даних: {e}")

def delete_data(self):
    table_name = self.view.prompt("Введіть назву таблиці: ")
    row_id = self.view.prompt("Введіть номер рядка для видалення: ")

    try:
        self.model.delete_data(table_name, row_id)
        self.view.display_message("Дані видалено успішно.")
    except Exception as e:
        self.view.display_message(f"Помилка видалення даних: {e}")

def generate_data(self):
    table_name = self.view.prompt("Введіть назву таблиці для генерації
даних: ")
    count_input = self.view.prompt("Введіть кількість записів для
генерації: ")

    try:
        count = int(count_input)
    except ValueError:
        self.view.display_message("Неправильний формат числа. Будь ласка,
введіть ціле число.")
        return

    try:
        self.model.generate_data(table_name, count)
        self.view.display_message(f"Успішно згенеровано {count} записів
для таблиці {table_name}.")
    except Exception as e:
        self.view.display_message(f"Помилка генерації даних для таблиці
{table_name}: {e}")

```

## Результат виконання програми

Перегляд змісту таблиці:

```

Оберіть опцію: 3
Введіть назву таблиці: researcher
(1, 'Yangkang\n', 'Chen', 'yangkang.chen@utexas.edu', 'Seismology')
(2, 'Sergey', 'Fomel', 'sergey.fomel@utexas.edu', 'Geophysics')
(3, 'Alexandros', 'Savvaidis', 'alexandros.savvaidis@utexas.edu', 'Geophysics')
(4, 'Li ', 'Zhang', 'li.zhang@zju.edu.cn', 'Data Science')
(5, 'Emily', 'Roberts', 'emily.roberts@utexas.edu', 'Data Science')

```

### Додавання даних:

```

Оберіть опцію: 4
Введіть назву таблиці: researcher
Введіть назви стовпців через кому: researcher_id, first_name, last_name, email, specialization
Введіть значення через кому: 6, Terence, Tao, terencetao@ucla.edu, Mathematics
Дані додано успішно.

```

	researcher_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (50)	specialization character varying (50)
1	1	Yangkang	Chen	yangkang.chen@utexas.edu	Seismology
2	2	Sergey	Fomel	sergey.fomel@utexas.edu	Geophysics
3	3	Alexandros	Savvaidis	alexandros.savvaidis@utexas.edu	Geophysics
4	4	Li	Zhang	li.zhang@zju.edu.cn	Data Science
5	5	Emily	Roberts	emily.roberts@utexas.edu	Data Science
6	6	Terence	Tao	terencetao@ucla.edu	Mathematics

### Редагування даних:

	researcher_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (50)	specialization character varying (50)
1	1	Yangkang	Chen	yangkang.chen@utexas.edu	Seismology
2	2	Sergey	Fomel	sergey.fomel@utexas.edu	Geophysics
3	3	Alexandros	Savvaidis	alexandros.savvaidis@utexas.edu	Geophysics
4	4	Li	Zhang	li.zhang@zju.edu.cn	Data Science
5	5	Emily	Roberts	emily.roberts@utexas.edu	Data Science
6	6	Terence	Tao	terencetao@ucla.edu	Mathematics

```

Оберіть опцію: 5
Введіть назву таблиці: researcher
Введіть назву стовпчика для оновлення: specialization
Введіть нове значення: Harmonic Analysis
Введіть номер рядка: 6
Дані оновлено успішно.

```

	researcher_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (50)	specialization character varying (50)
1	1	Yangkang	Chen	yangkang.chen@utexas.edu	Seismology
2	2	Sergey	Fomel	sergey.fomel@utexas.edu	Geophysics
3	3	Alexandros	Savvaiddis	alexandros.savvaiddis@utexas.edu	Geophysics
4	4	Li	Zhang	li.zhang@zju.edu.cn	Data Science
5	5	Emily	Roberts	emily.roberts@utexas.edu	Data Science
6	6	Terence	Tao	terencetao@ucla.edu	Harmonic Analysis

Видалення даних:

	publication_id [PK] integer	title character varying (100)	year integer	journal character varying (100)	research_project_id integer
1	1	AI-Driven Earthquake Prediction	2023	Bulletin of the Seismological Society of America	1
2	2	Random title bd7ab	2039	Random journal 4660b	1
3	3	Random title 88624	2028	Random journal eaa60	1
4	4	Random title 02252	2008	Random journal c1bf2	1
5	5	Random title 369b8	2001	Random journal 82243	1

Оберіть опцію: 6  
Введіть назву таблиці: publication  
Введіть номер рядка для видалення: 3  
Дані видалено успішно.

	publication_id [PK] integer	title character varying (100)	year integer	journal character varying (100)	research_project_id integer
1	1	AI-Driven Earthquake Prediction	2023	Bulletin of the Seismological Society of America	1
2	2	Random title bd7ab	2039	Random journal 4660b	1
3	4	Random title 02252	2008	Random journal c1bf2	1
4	5	Random title 369b8	2001	Random journal 82243	1

Генерування даних:

	research_project_id [PK] integer	title character varying (100)	description character varying (500)
1	1	AI-Driven Earthquake Prediction	This project developed an artificial intelligence algorithm capable of predicting earthquakes with a 70% success rate during a seven-month trial in China. The AI used se

Оберіть опцію: 7  
Введіть назву таблиці для генерації даних: research\_project  
Введіть кількість записів для генерації: 3  
Успішно згенеровано 3 записів для таблиці research\_project.

	research_project_id [PK] integer	title character varying (100)	description character varying (500)
1	1	AI-Driven Earthquake Prediction	This project developed an artificial intelligence algorithm capable of predicting
2	2	Random title 7e5de	Random description 3b874
3	3	Random title 71a59	Random description 81556
4	4	Random title ea0b0	Random description 5c9ce

	id [PK] integer	researcher_id integer	research_project_id integer
1	1	1	1
2	2	2	1
3	3	3	1

Оберіть опцію: 7  
Введіть назву таблиці для генерації даних: researcher\_project  
Введіть кількість записів для генерації: 10  
Успішно згенеровано 10 записів для таблиці researcher\_project.

	id [PK] integer	researcher_id integer	research_project_id integer
1	1	1	1
2	2	2	1
3	3	3	1
4	4	5	2
5	5	1	2
6	6	2	1
7	7	6	2
8	8	5	1
9	9	6	3
10	10	1	1
11	11	6	4
12	12	3	1
13	13	5	1