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 III курсу
групи КВ-23
Перетятько Б.В.

Київ – 2024

Мета: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Виконання роботи

Опис предметної області

Предметна область – це система обліку тренувань і виконання вправ користувачами. Ця база даних охоплює дані про користувачів, їхні тренування та вправи, що виконуються під час тренувань. Вона дозволяє вести облік фізичних характеристик користувачів, таких як зріст та вага, а також зберігати інформацію про кількість виконаних повторень і підходів під час тренувань. Ця система допомагає відстежувати прогрес кожного користувача і аналізувати результати його тренувань.

Опис сутностей

Для побудови бази даних обраної області, були виділені такі сутності:

1. **Користувач (User)**

   Атрибути:

   a. `user_id`: Ідентифікатор користувача.
   b. `user_firstname`: Ім'я користувача.
   c. `user_lastname`: Прізвище користувача.
   d. `user_weight`: Вага користувача.
   e. `user_height`: Зріст користувача.

   Призначення: Збереження інформації про користувачів системи (спортсменів).

2. **Тренування (Training)**

   Атрибути:

   a. `training_id`: Ідентифікатор тренування.
   b. `user_id`: Ідентифікатор користувача, який проходить тренування.
   c. `start_date_time`: Дата і час початку тренування.
   d. `end_date_time`: Дата і час завершення тренування.

   Призначення: Збереження даних про тренування користувачів, включаючи їх тривалість і користувача, який тренується.

3. **Вправа (Exercise)**

   Атрибути:

a. `exercise_id`: Ідентифікатор вправи.
b. `exercise_name`: Назва вправи.
c. `difficulty`: Рівень складності вправи.
d. `description`: Опис вправи.

Призначення: Збереження інформації про різні вправи, які можуть виконуватися під час тренування.

4. **Тренування-Вправа (Workout)**

Атрибути:

a. `training_id`: Ідентифікатор тренування.
b. `exercise_id`: Ідентифікатор вправи.
c. `number_of_sets`: Кількість підходів.
d. `number_of_repetitions`: Кількість повторень у кожному підході.

Призначення: Збереження інформації про вправи, які виконувалися під час конкретного тренування, включаючи кількість підходів та повторень.

Опис зв’язків між сутностями

Кожен користувач може брати участь у кількох тренуваннях 1:N. Тренування прив'язане до одного користувача через поле user_id.

Під час кожного тренування користувач може виконувати кілька різних вправ 1:N. Ці вправи фіксуються у таблиці workout, яка зберігає інформацію про те, скільки підходів і повторень було виконано для кожної вправи.

Таблиця exercises містить інформацію про всі можливі вправи, а таблиця workout пов'язує їх з конкретними тренуваннями через training_id і exercise_id.

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

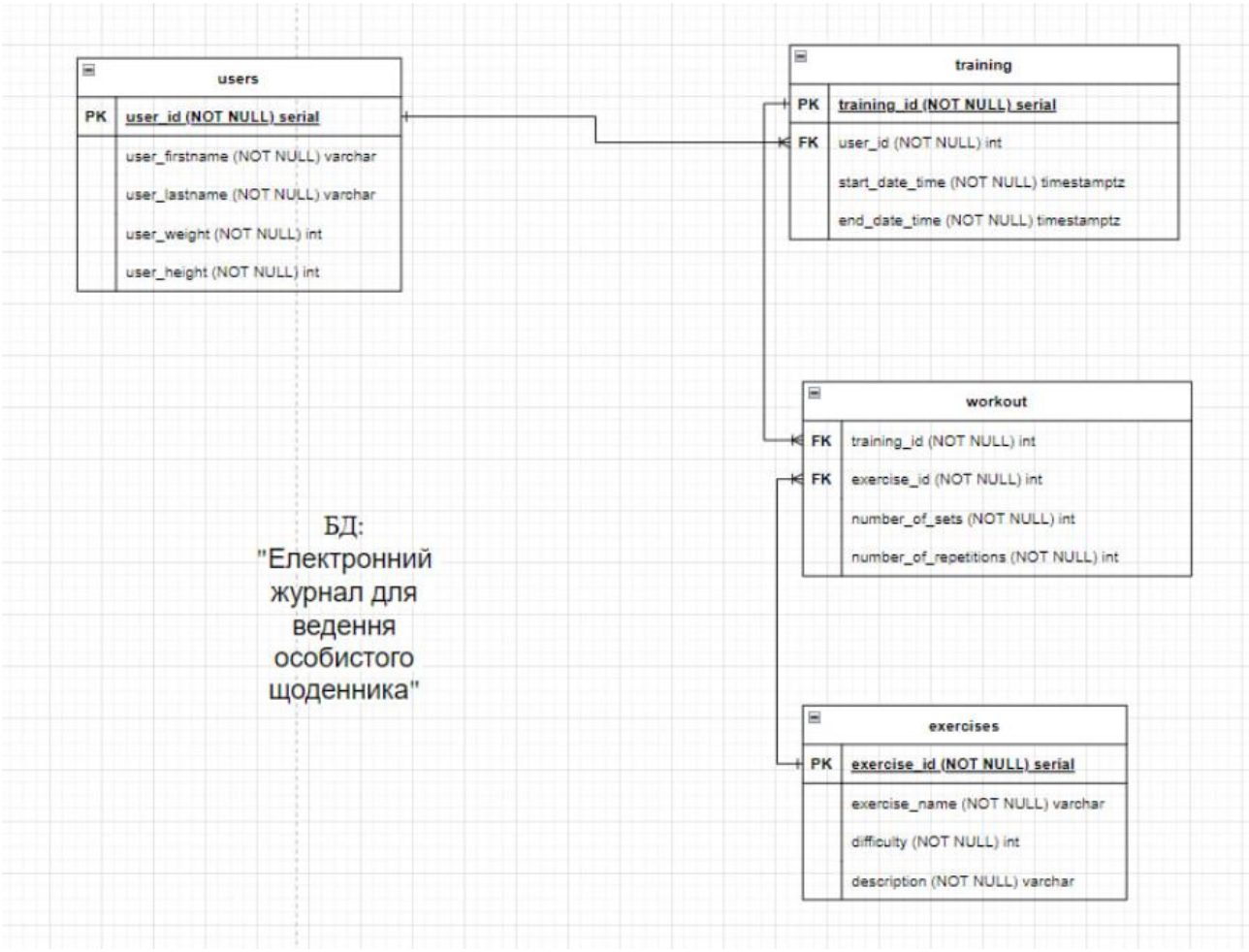

Рисунок 1 – Логічна модель

Середовище та компоненти розробки

У процесі розробки була використана мова програмування Python, інтегроване середовище розробки PC CE, а також була використана бібліотека psycopg, для взаємодії з базою даних PostgreSQL.

Шаблон проектування

Модель-представлення-контролер (MVC) – це шаблон проектування, що використовується у програмі. Кожен компонент відповідає за певну функціональну частину:

1. Модель (Model) – це клас, що відображає логіку роботи з даними, обробляє всі операції з даними, такі як додавання, оновлення, вилучення.
2. Представлення (View) – це клас, через який користувач взаємодіє з програмою. У даному випадку, консольний інтерфейс, який відображає дані для користувача та зчитує їх з екрану.
3. Контролер (Controller) – це клас, який відповідає за зв’язок між користувачем і системою. Він приймає введені користувачем дані та обробляє

їх. В залежності від результатів, викликає відповідні дії з Model або View.

Даний підхід дозволяє розділити логіку програми на логічні компоненти, що полегшує розробку, тестування і підтримку продукту.

Структура програми та її опис

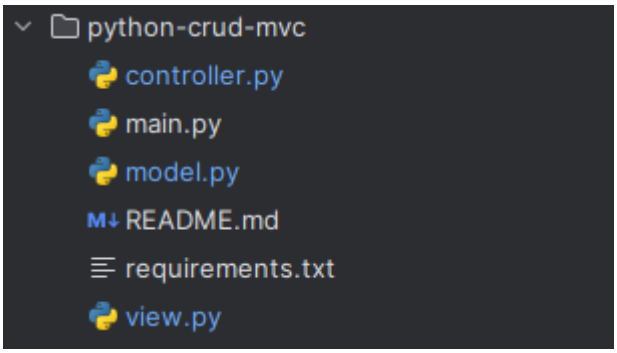

Рисунок 2 – Структура програми

З файлу main.py відбувається виклик контролера та передача йому управління.

У файлі model.py описаний клас моделі, який відповідає за управління підключенням до бази даних і виконанням низькорівневих запитів до неї.

У файлі controller.py реалізовано інтерфейс взаємодії з користувачем, включаючи обробку запитів користувача, а також інші дії, необхідні для взаємодії з моделлю та представленням.

У файлі view.py описаний клас, який відображає результати виконання різних дій користувача на екрані консолі. Цей компонент відповідає за представлення даних користувачу в зручному для сприйняття вигляді.

Отже, структура програми відповідає патерну MVC.

Структура меню програми

На рисунку 3 зображено меню користувача, яке складається з семи пунктів.

Рисунок 3 – Структура меню користувача

Фрагмент коду (файл controller.py), в якому наведено головний цикл роботи програми

```
def run(self):
    while True:
        choice = self.view.show_menu()
        if choice == '1':
            self.view_tables()
        elif choice == '2':
            self.view_columns()
        elif choice == '3':
            self.add_data()
        elif choice == '4':
            self.update_data()
        elif choice == '5':
            self.delete_data()
        elif choice == '6':
            self.generate_data()
        elif choice == '7':
            self.read_data()
        elif choice == '8':
            self.find_training_first()
        elif choice == '9':
            self.find_exercise_name()
        elif choice == '10':
            self.find_avg_exercises()
        elif choice == '11':
            break
```

Фрагмент коду (файл model.py), в якому наведено функції внесення, редагування, вилучення та генерації даних у базі даних

Функція внесення даних:

```
def add_data(self, table, columns, val):
    c = self.conn.cursor()
    columns_str = ', '.join(columns)
    placeholders = ', '.join(['%s'] * len(val))
    try:
        c.execute(f'INSERT INTO "public"."{table}" ({columns_str}) VALUES
({placeholders})', val)
        self.conn.commit()
        return "all done"
    except Exception as e:
        return e
```

Дана функція вставляє нове значення до відповідної таблиці в БД, генеруючи для неї відповідний запрос. Для відслітковування помилок використовується try-except.

Функція оновлення даних:

```
def update_data(self, table, columns, id, new_values):
    if len(columns) > 1:
        columns_str = '=%s, '.join(columns).strip() + "=%s"
    else:
        columns_str = columns[0] + "=%s"
    table_temp = table
    if table == 'users' or table == 'exercises':
        table_temp = table_temp[:-1]
    try:
        c = self.conn.cursor()
        c.execute(f'UPDATE {table} SET {columns_str} WHERE {table_temp}_id=%s',
(*new_values, id,))
        self.conn.commit()
        return "all done"
    except Exception as e:
        return e
```

Дана функція знаходить назву колонки де знаходиться pk таблиці для генерування строки запросу до відповідної таблиці в БД, після чого приводить отриманні дані від користувача і змінює відповідні дані.Для відслітковування помилок використовується try-except.

Функція видалення значень:

```
def delete_data(self, table, id):
    table_temp = table
    if table == 'users' or table == 'exercises':
        table_temp = table_temp[:-1]
    try:
        c = self.conn.cursor()
        c.execute(f'DELETE FROM {table} WHERE {table_temp}_id = %s', (id,))
        self.conn.commit()
        return "all done"
    except Exception as e:
        return e
```

Дана функція видаляє відповідний запис з обраної таблиці за pk, також у БД налаштоване CASCADE видалення, за рахунок цього будуть видалені відповідні записи в дочірніх таблицях.Для відслітковування помилок використовується try-except.

Функція генерування даних:

```
def generate_data(self, table, rows_count):
    try:
        c = self.conn.cursor()
        columns = self.get_all_columns(table)
        if table != "workout":
            del columns[0]
        columns_str = ', '.join(columns)
        column_types = self.get_all_column_types(table, columns)

        foreign_keys = self.get_foreign_keys(table)

        values_list = []
        for col in columns:
            if col in foreign_keys:
                ref_table = foreign_keys[col]
                table_temp = ref_table
                if ref_table == 'users' or ref_table == 'exercises':
                    table_temp = table_temp[:-1]
                c.execute(f"(SELECT {table_temp}_id FROM {ref_table} ORDER BY
random() LIMIT 1)")
                ref_value = c.fetchone()[0]  # Отримуємо перше значення з
вибірки
                values_list.append(str(ref_value))
            else:
                if column_types[col] == 'integer':
                    values_list.append("(random() * 100)::INT")
                elif column_types[col] == 'text':
                    values_list.append("array_to_string(array(select chr(65 +
trunc(random() * 26)::int) from generate_series(1, 5)), '')")
                elif column_types[col] == 'timestamp with time zone':
                    values_list.append("date_trunc('seconds', now() + (random()
* INTERVAL '365 days' - INTERVAL '182 days'))")
                else:
                    values_list.append("md5(random()::text)")

        values = ', '.join(values_list)

        sql = f"""
            DO $$
            DECLARE
```

```
                record_count INT := {rows_count};
            BEGIN
                FOR i IN 1..record_count LOOP
                    INSERT INTO {table} ({columns_str})
                    VALUES (
                        {values}
                    );
                END LOOP;
            END $$;
            """
        c.execute(sql)
        self.conn.commit()
        return "all done"
    except Exception as e:
        return e
```

Дана функція генерує рекорди у відповідну таблицю також налаштовано reference в таблицях. Для відслітковування помилок використовується try-except.

Приклади виконання вище наведених функцій

```
Зробіть вибір: 7
Введіть назву таблиці: users
user_id: 1 user_firstname: John user_lastname: Doe user_weight: 75 user_height: 180
user_id: 4 user_firstname: bohdan user_lastname: peretiatko user_weight: 13 user_height: 14
user_id: 6 user_firstname: ikki user_lastname: dsa user_weight: 12 user_height: 13
user_id: 2 user_firstname: misha user_lastname: marinchenko user_weight: 12 user_height: 13
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 5
Введіть назву таблиці: users
Введіть ID рядка, який потрібно видалити: 2
all done
```

```
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 7
Введіть назву таблиці: users
user_id: 1 user_firstname: John user_lastname: Doe user_weight: 75 user_height: 180
user_id: 4 user_firstname: bohdan user_lastname: peretiatko user_weight: 13 user_height: 14
user_id: 6 user_firstname: ikki user_lastname: dsa user_weight: 12 user_height: 13
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 7
Введіть назву таблиці: training
training_id: 1 user_id: 1 start_date_time: 2024-09-19 08:00:00+03:00 end_date_time: 2024-09-19 09:00:00+03:00
```

Зверху демонстрація як працює налаштоване каскадне видалення. Як можемо бачити залежні рекорди в таблиці training видалилися.

```
Введіть назву таблиці: training
Введіть назви колонок (через пробіл): user_id start_date_time end_date_time
Введіть відповідні значення (через пробіл): 4 2024-09-19 2024-09-19
all done
Меню:
```

```
all done
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 3
Введіть назву таблиці: training
Введіть назви колонок (через пробіл): user_id start_date_time end_date_time
Введіть відповідні значення (через пробіл): 8 2024-09-19 2024-09-19
insert або update в таблиці "training" порушує обмеження зовнішнього ключа "user_id"
DETAIL:  Ключ (user_id)=(8) не присутній в таблиці "users".
```

Зверху я представив приклад додавання данних в дочірню таблицю training. На першому скріншоті видно що я додав усе правильно по існуючому FK що не викликає помилок. А далі я спробував записати неіснуючий FK тому це і викликало помилку і написало в чому саме проблема.

```
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 6
Введіть назву таблиці: users
Введіть кількість рядків для генерації: 5
all done
```

```
2  select * from users
3
```

Data Output | Messages | Notifications

	user_id [PK] integer	user_firstname text	user_lastname text	user_weight integer	user_height integer
1	1	John	Doe	75	180
2	4	bohdan	peretiatko	13	14
3	6	ikki	dsa	12	13
4	16	XLASX	QMMBS	61	47
5	17	SCOGQ	MAAVC	98	15
6	18	IMYRC	JVHBL	41	9
7	19	TXXWA	NSCYI	71	54
8	20	AKTZV	AQTGP	34	63
9	21	WWGQC	FFXHT	2	50

Зверху продемонстрував роботу функції генерації данних в відповідну таблицю.

```
DO $$
DECLARE
    record_count INT := 1;
BEGIN
    FOR i IN 1..record_count LOOP
        INSERT INTO users (user_firstname, user_lastname, user_weight, user_height)
        VALUES (
            array_to_string(array(select chr(65 + trunc(random() * 26)::int) from generate_series(1,
             5)), ''), array_to_string(array(select chr(65 + trunc(random() * 26)::int) from
             generate_series(1, 5)), ''), (random() * 100)::INT, (random() * 100)::INT
        );
    END LOOP;
END $$;
```

Зверху через debugger в PyCharm подивився як виглядає кінцевий SQL запит.

```
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 8
Введіть вагу користувача: 75
Введіть назву вправи: Push Ups
training_id: 1 start_date_time: 2024-09-19 08:00:00+03:00 end_date_time: 2024-09-19 09:00:00+03:00
Час виконання запиту: 0.0080 секунд
Меню:
```

```
Меню:
1. Вивід назв таблиць
2. Вивід імен стовпчиків таблиці
3. Додавання даних в таблицю
4. Оновлення даних в таблиці
5. Видалення даних в таблиці
6. Генерування даних в таблицю
7. Перегляд данних в таблиці
8. Знайти тренування по вазі користувача і назви вправи
9. Знайти вправу по кількості підходів і складністю
10. Знайти середні показники для тренувань по початку тренування
11. Вихід
Зробіть вибір: 9
Введіть точну кількість підходів: 3
Введіть точний рівень складності вправи: 2
exercise_name: Push Ups difficulty: 2 number_of_sets: 3
Час виконання запиту: 0.0010 секунд
Меню:
```

Зверху навів приклади уведення пошукового запиту та результатів виконання запитів

```
SELECT DISTINCT t.training_id, t.start_date_time, t.end_date_time
FROM training t
JOIN users u ON t.user_id = u.user_id
JOIN workout w ON t.training_id = w.training_id
JOIN exercises e ON w.exercise_id = e.exercise_id
WHERE u.user_weight = 75
AND e.exercise_name = 'Push Ups';
```

Зверху кінцевий SQL запит для першого прикладу

ПОВНИЙ КОД В model.py:

```
import psycopg
import time
```

```
class Model:
    def __init__(self):
        self.conn = psycopg.connect(
            dbname='lab1',
            user='postgres',
            password='Vfdgjxrf1!',
            host='localhost',
            port=5432
        )

    def get_all_tables(self): # повертає усі таблиці в базі данних
        c = self.conn.cursor()
        c.execute("SELECT table_name FROM information_schema.tables WHERE
table_schema = 'public'")
        tables = [table[0] for table in c.fetchall()]
        return tables

    def get_all_columns(self, table_name): # повертає усі колонки у відповідній
таблиці
        c = self.conn.cursor()
        c.execute("SELECT column_name FROM information_schema.columns WHERE
table_name = %s ORDER BY ordinal_position", (table_name,))
        columns = [row[0] for row in c.fetchall()]
        return columns

    def get_all_column_types(self, table_name, columns) -> dict: # повертає усі
колонки і типи колонок у вигляді словника
        column_types = {}
        with self.conn.cursor() as cursor:
            cursor.execute("""
                SELECT column_name, data_type
                FROM information_schema.columns
                WHERE table_name = %s AND column_name = ANY(%s);
            """, (table_name, columns))

            for column_name, data_type in cursor.fetchall():
                column_types[column_name] = data_type

        return column_types

    def get_foreign_keys(self, table): # повертає усі зовнішні ключі для
відповідної таблиці
        query = f"""
            SELECT
                kcu.column_name,
                ccu.table_name AS referenced_table
            FROM
                information_schema.table_constraints AS tc
                JOIN information_schema.key_column_usage AS kcu
                    ON tc.constraint_name = kcu.constraint_name
                JOIN information_schema.constraint_column_usage AS ccu
                    ON ccu.constraint_name = tc.constraint_name
            WHERE tc.table_name = '{table}' AND tc.constraint_type = 'FOREIGN
KEY';
        """
        c = self.conn.cursor()
        c.execute(query)
        foreign_keys = {row[0]: row[1] for row in c.fetchall()}
        return foreign_keys

    def add_data(self, table, columns, val): # додає дані до таблиці
        c = self.conn.cursor()
        columns_str = ', '.join(columns)
        placeholders = ', '.join(['%s'] * len(val))
```

```
        try:
            c.execute(f'INSERT INTO "public"."{table}" ({columns_str}) VALUES
({placeholders})', val)
            self.conn.commit()
            return "all done"
        except Exception as e:
            return e

    def read_data(self, table): # повертає дані з таблиці
        table_temp = table
        if table == 'users' or table == 'exercises':
            table_temp = table_temp[:-1]
        try:
            c = self.conn.cursor()
            c.execute(f'SELECT * FROM {table}')
            return c.fetchall()
        except Exception as e:
            print(e)

    def delete_data(self, table, id): # видаляє дані з таблиці
        table_temp = table
        if table == 'users' or table == 'exercises':
            table_temp = table_temp[:-1]
        try:
            c = self.conn.cursor()
            c.execute(f'DELETE FROM {table} WHERE {table_temp}_id = %s', (id,))
            self.conn.commit()
            return "all done"
        except Exception as e:
            return e

    def update_data(self, table, columns, id, new_values): # оновлює дані в
таблиці
        if len(columns) > 1:
            columns_str = '=%s, '.join(columns).strip() + "=%s"
        else:
            columns_str = columns[0] + "=%s"
        table_temp = table
        if table == 'users' or table == 'exercises':
            table_temp = table_temp[:-1]
        try:
            c = self.conn.cursor()
            c.execute(f'UPDATE {table} SET {columns_str} WHERE
{table_temp}_id=%s', (*new_values, id,))
            self.conn.commit()
            return "all done"
        except Exception as e:
            return e

    def generate_data(self, table, rows_count): # генерує і додає дані в таблицю
        try:
            c = self.conn.cursor()
            columns = self.get_all_columns(table)
            if table != "workout":
                del columns[0]
            columns_str = ', '.join(columns)
            column_types = self.get_all_column_types(table, columns)

            foreign_keys = self.get_foreign_keys(table)

            values_list = []
            for col in columns:
                if col in foreign_keys:
                    ref_table = foreign_keys[col]
```

```
                    table_temp = ref_table
                    if ref_table == 'users' or ref_table == 'exercises':
                        table_temp = table_temp[:-1]
                    c.execute(f"(SELECT {table_temp}_id FROM {ref_table} ORDER
BY random() LIMIT 1)")
                    ref_value = c.fetchone()[0]  # Отримуємо перше значення з
вибірки
                    values_list.append(str(ref_value))
                else:
                    if column_types[col] == 'integer':
                        values_list.append("(random() * 100)::INT")
                    elif column_types[col] == 'text':
                        values_list.append("array_to_string(array(select chr(65
+ trunc(random() * 26)::int) from generate_series(1, 5)), '')")
                    elif column_types[col] == 'timestamp with time zone':
                        values_list.append("date_trunc('seconds', now() +
(random() * INTERVAL '365 days' - INTERVAL '182 days'))")
                    else:
                        values_list.append("md5(random()::text)")

            values = ', '.join(values_list)

            sql = f"""
                DO $$
                DECLARE
                    record_count INT := {rows_count};
                BEGIN
                    FOR i IN 1..record_count LOOP
                    INSERT INTO {table} ({columns_str})
                    VALUES (
                        {values}
                    );
                    END LOOP;
                END $$;
                """
            c.execute(sql)
            self.conn.commit()
            return "all done"
        except Exception as e:
            return e

    def find_training_first(self, user_weight, exercise_name): # пошук
тренування
        try:
            c = self.conn.cursor()
            sql = f"""
                SELECT DISTINCT t.training_id, t.start_date_time,
t.end_date_time
                FROM training t
                JOIN users u ON t.user_id = u.user_id
                JOIN workout w ON t.training_id = w.training_id
                JOIN exercises e ON w.exercise_id = e.exercise_id
                WHERE u.user_weight = {user_weight}
                AND e.exercise_name = '{exercise_name}';
                """
            start_time = time.time()
            c.execute(sql)
            elapsed_time = time.time() - start_time
            res_time_string = f"Час виконання запиту: {elapsed_time:.4f} секунд"
            columns = []
            columns.append("training_id")
            columns.append("start_date_time")
            columns.append("end_date_time")
            return c.fetchall(), columns, res_time_string
```

```
        except Exception as e:
            print(e)
            return [], []

    def find_exercise_name(self, number_of_sets, difficulty): # пошук вправи
        try:
            c = self.conn.cursor()
            sql = f"""
                SELECT DISTINCT e.exercise_name, e.difficulty, w.number_of_sets
FROM exercises e
                JOIN workout w ON e.exercise_id = w.exercise_id
                WHERE w.number_of_sets = %s AND e.difficulty = %s;
                """

            columns = []
            columns.append("exercise_name")
            columns.append("difficulty")
            columns.append("number_of_sets")
            start_time = time.time()
            c.execute(sql, (number_of_sets, difficulty))
            elapsed_time = time.time() - start_time
            res_time_string = f"Час виконання запиту: {elapsed_time:.4f} секунд"
            return c.fetchall(), columns, res_time_string
        except Exception as e:
            print(e)
            return [], []

    def find_avg_exercises(self, date): # пошук середних показників для
тренувань
        try:
            c = self.conn.cursor()
            sql = f"""
                SELECT
                    t.start_date_time,
                    AVG(w.number_of_sets) AS avg_sets,
                    AVG(w.number_of_repetitions) AS avg_reps
                FROM
                    training t
                JOIN
                    workout w ON t.training_id = w.training_id
                WHERE
                    t.start_date_time >= '{date}'
                GROUP BY
                    t.start_date_time
                ORDER BY
                    t.start_date_time DESC;
                """

            columns = []
            columns.append("exercise_name")
            columns.append("difficulty")
            columns.append("number_of_sets")
            start_time = time.time()
            c.execute(sql)
            elapsed_time = time.time() - start_time
            res_time_string = f"Час виконання запиту: {elapsed_time:.4f} секунд"
            return c.fetchall(), columns, res_time_string
        except Exception as e:
            print(e)
            return [], []
```