

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

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

Виконав:

студент ІII курсу

групи КВ-93

Осика І.О.

Перевірив:

Київ – 2021

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.

2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.

3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.

4. Програмний код виконати згідно шаблону MVC (модель-подання- контролер)

**Інформація про програму**

Використана мова програмування: Python 3.9

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), tabulate, sqlalchemy.

**Модель «сутність-зв’язок» кіберспортивного турніру**

Згідно цієї області для побудови бази даних було виділено наступні сутності:

1. Гравець, з атрибутами: ім’я, вік, нік, національність. Учасник турніру, який змагається у певній кібер-спортивній дисципліні.
2. Тренер, з атрибутами: ім’я, вік, нік, національність. Учасник турніру, який підтримує та може обговорити подальші дії під час тактичних пауз.
3. Команда, з атрибутами: країна\регіон, назва. Група в яку входить певна кількість гравців та 1-2 тренера.
4. Матч, з атрибутами: дата\час, ID матчу, рахунок. Один з матчів кібер-спортивного турніру, під час якого змагаються дві команди.

**Опис зв’язків**

Один або декілька тренерів можуть тренувати багатьох гравців тому між ними зв’язок M:N. Тобто, один тренер може тренувати багатьох гравців, і один гравець може тренуватися у багатьох тренерів

І тренер(и) і гравці є частиною однієї команди, тому зв’язок 1:N присутній між командою-тренером і командою-гравцем.

І так як це турнір, тому багато команд грають якусь кількість матчів одна проти одної, тож до сутності «матч» від сутності «команда» йдуть два зв’язки 1:N

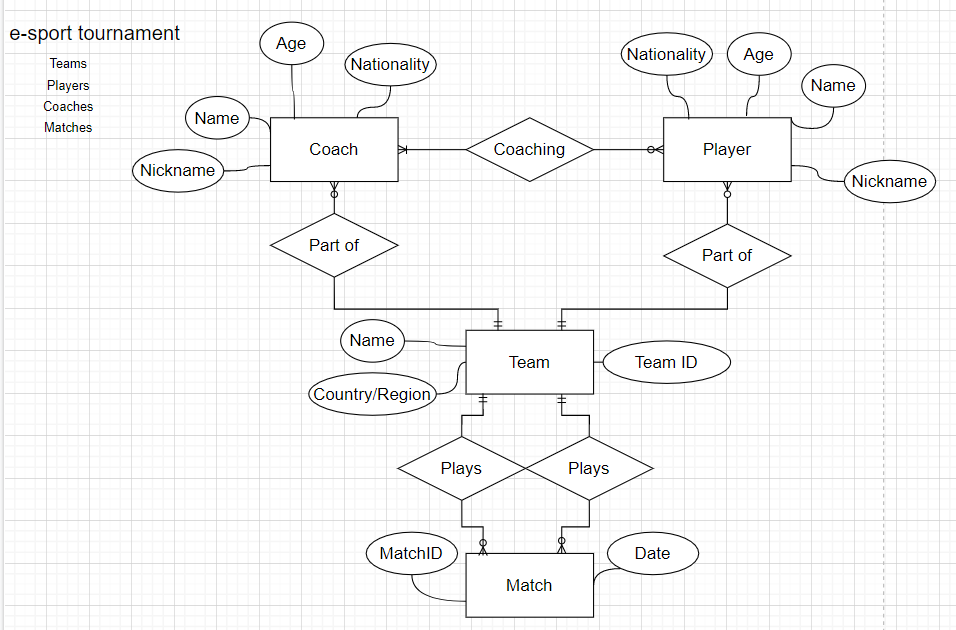


Рисунок 1 - ER-діаграма, побудована за нотацією Чена

**Перетворення моделі у схему бази даних**

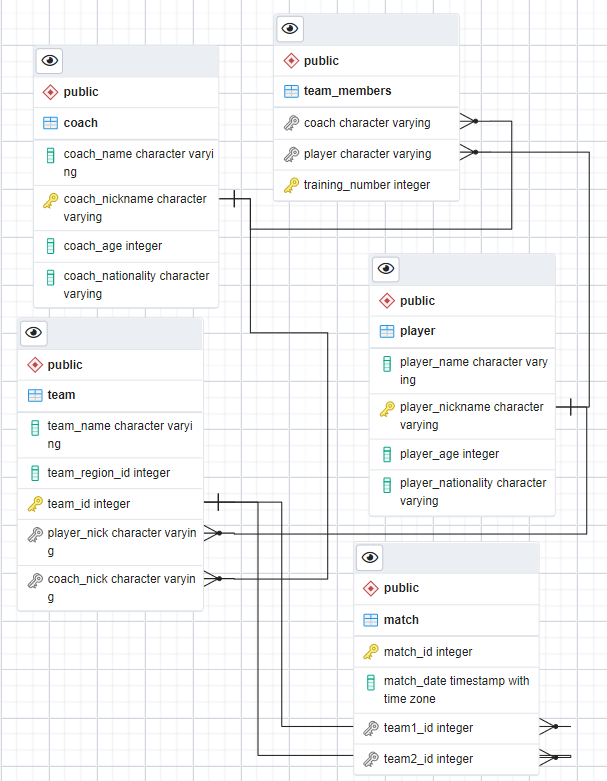


Рисунок 2 - Схема бази даних у pgAdmin 4

Опис структури БД

|  |  |  |
| --- | --- | --- |
| Сутність | Атрибут | Тип атрибуту |
| **player** - *містить дані про гравця* | **Player\_name** *– ПІБ гравця*  **Player\_nickname** *– нік гравця*  **Player\_age** *– вік гравця*  **Player\_nationality** *– його національність* | **character varying** *(рядок)*  **character varying** *(рядок)*  **integer** *(числовий)*  **character varying** *(рядок)* |
| **coach** - *містить дані про тренера* | **coach\_name** *– ПІБ тренера*  **coach\_nickname** *– нік тренера*  **coach\_age** *– вік тренера*  **coach\_nationality** *– його національність* | **character varying** *(рядок)*  **character varying** *(рядок)*  **integer** *(числовий)*  **character varying** *(рядок)* |
| **team** – *дані про команду* | **Team\_name** *– назва команди*  **Team\_region\_id** *– ідентифікатор регіону команди*  **Team\_id** *– унікальний ідентифікатор команди* | **character varying** *(рядок)*  **integer** *(числовий)*  **integer** *(числовий)* |
| **match** – *інформація про матч* | **Match\_id** *- унікальний ідентифікатор матчу*  **Match\_date** *– дата-час матчу* | **integer** *(числовий)*  **timestamp** *(мітка часу)* |

# Схема меню користувача

# Изображение выглядит как текст Автоматически созданное описание

На знімку екрану термінала продемонстровано початкове меню, де можна побачити функції які можна виконати з БД. Кожна команда запускає відповідну функцію з файлу controller.py, яка в подальшому передає аргументи у функцію файлу model.py, яка в свою чергу формує і здійснює запит до бази даних.

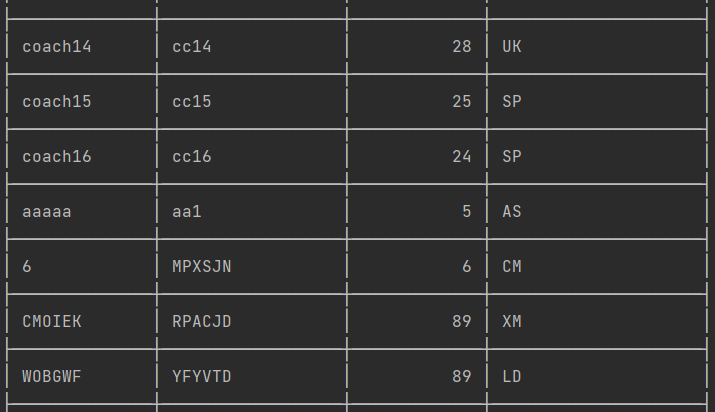
**Завдання 1**

***Запит на видалення***

def delete(self, table, cond):  
 try:  
 self.attr = cond.split(':')[0]  
 self.value = cond.split(':')[1]  
 self.connection.commit()  
 self.cursor.execute(f'DELETE FROM public."{table}" WHERE {self.attr} = \'{self.value}\' ')  
 self.connection.commit()  
 except Exception as e:  
 print(e)  
 return False  
 return True

Для перевірки розглянемо видалення даних з таблиці coach.

Стан таблиці до видалення:



Виклик відповідної функції:

Изображение выглядит как текст

Автоматически созданное описание

Стан таблиці після видалення:

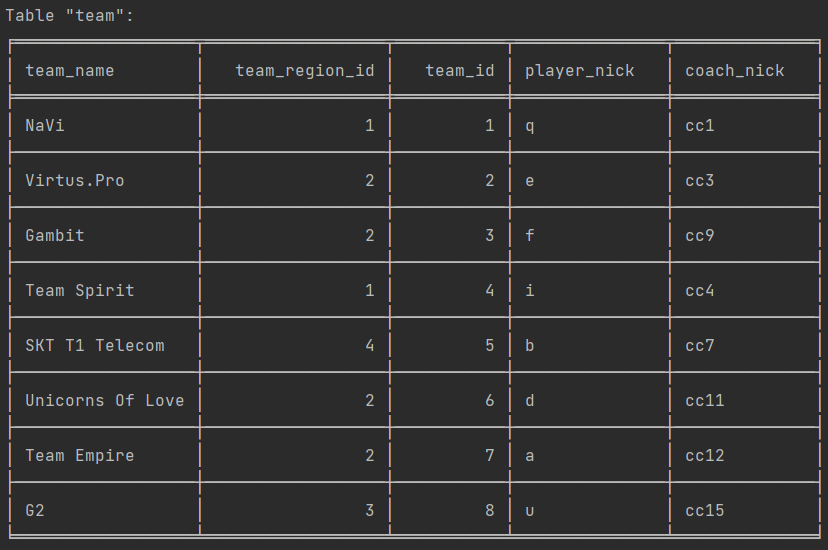
Изображение выглядит как текст, мебель, экран, шкафчик

Автоматически созданное описание

***Запит на вставку поля***

def insert(self, table, values):  
 try:  
 self.val = tuple(values)  
 self.connection.commit()  
 self.cursor.execute(f"INSERT INTO public.\"{table}\" VALUES {self.val}")  
 self.connection.commit()  
 except Exception as e:  
 print(e)  
 return False  
 return True

Вставимо дані в таблицю team

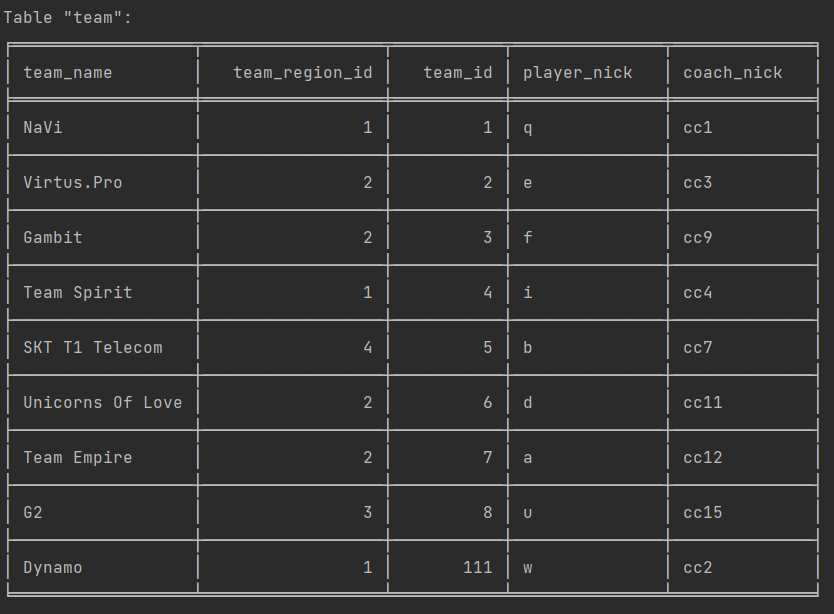
Стан таблиці до: 

Виклик відповідної функції:

Изображение выглядит как текст

Автоматически созданное описание

Стан таблиці після:

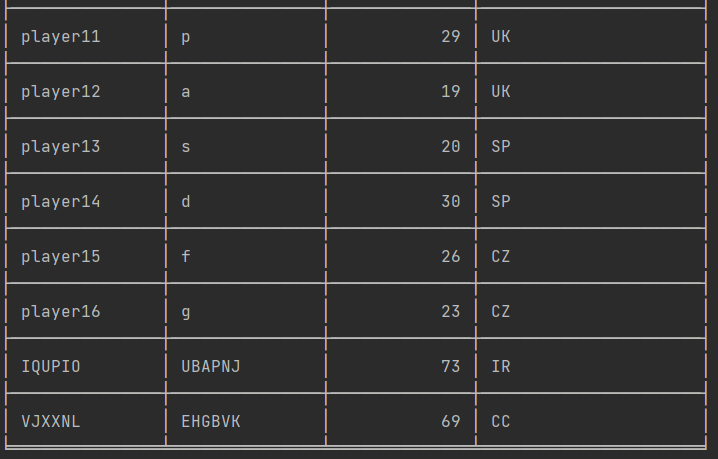


***Запит на редагування рядків***

def update(self, table, attr, update\_attr, new\_value):  
 try:  
 self.attr = attr.split(':')[0]  
 self.value = attr.split(':')[1]  
 self.connection.commit()  
 self.cursor.execute(  
 f"UPDATE public.\"{table}\" SET {update\_attr} = '{new\_value}' WHERE {self.attr} = '{self.value}'")  
 self.connection.commit()  
 except Exception as e:  
 print(a)  
 return False  
 return True

Оновимо дані в таблиці company

Стан таблиці до редагування:

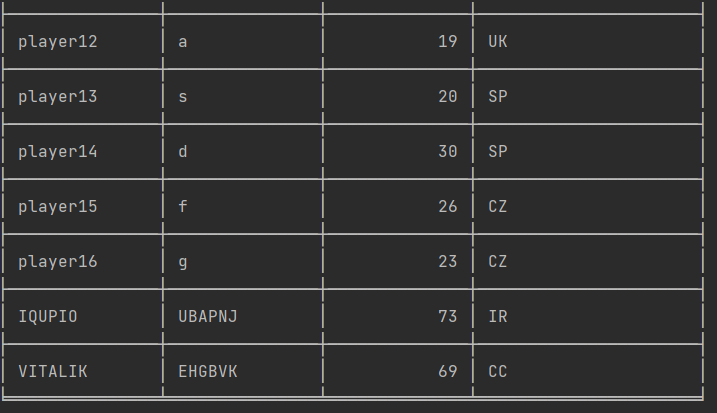


Виклик відповідної функції:

Изображение выглядит как текст

Автоматически созданное описание

Стан таблиці після редагування:



**Завдання 2**

def generate(self, table, count1, count2):  
 try:  
 count1 = str(int(count1) + 1)  
 count2 = str(int(count2)+int(count1))  
  
 if table == "coach":  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.coach  
 SELECT chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),   
 floor(65 + random()\*25)::int,  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int)  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 elif table == 'team':  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.team  
 SELECT generate\_series,  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),   
 generate\_series,  
 generate\_series  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 elif table == 'player':  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.player  
 SELECT chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),   
 floor(65 + random()\*25)::int,  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int)  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 elif table == 'match':  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.match  
 SELECT generate\_series,  
 DateTime.toZone('UTC'),  
 floor(65 + random()\*25)::int,  
 floor(65 + random()\*25)::int  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 else:  
 print("Such table doesn't exist")  
 except Exception as e:  
 print(e)  
 return False  
 return True

Вставимо 5 псевдорандомізованих записів в одну з таблиць.

Записи таблиць перед вставкою даних та після:

 Изображение выглядит как стол

Автоматически созданное описание

**Завдання 3**

def search\_by\_attr(self, table1, table2, attr, value):  
 try:  
 self.connection.commit()  
 self.cursor.execute(f"""SELECT \* FROM public.{table1} as first INNER JOIN public.{table2} as second on first.{attr} = second.{attr} WHERE {value}""")  
  
 self.rows = self.cursor.fetchall()  
 self.columns = []  
 for i in self.cursor.description:  
 self.columns.append(list(i)[0])  
 self.data = []  
 self.data.append(self.rows)  
 self.data.append(self.columns)  
 return self.data  
 except Exception as e:  
 print(e)  
 return None

Виконаємо об’єднання двох таблиць та пошук у них за двома атрибутами:

Изображение выглядит как текст, монитор, снимок экрана

Автоматически созданное описание

**Завдання 4**

Код програмного модулю “model.py”

import psycopg2  
  
from sqlalchemy import create\_engine  
  
engine = create\_engine('postgresql://postgres:12346789@localhost:5432/public',  
 echo=True)  
  
  
class Model:  
 def \_\_init\_\_(self, host, port, database, user, password):  
 self.host = host  
 self.port = port  
 self.database = database  
 self.user = user  
 self.\_\_password = password  
 self.f = 0  
 try:  
 self.connection = psycopg2.connect(host=host, port=port, database=database, user=user, password=password)  
 self.cursor = self.connection.cursor()  
 self.f = 1  
 except Exception as e:  
 print(e)  
  
 @property  
 def getter(self):  
 return self.\_\_password  
  
 def disconnect(self):  
 self.connection.close()  
 self.cursor.close()  
  
 def read\_table(self, table):  
 try:  
 self.connection.commit()  
 self.cursor.execute(f'SELECT \* FROM public.{table}')  
 self.rows = self.cursor.fetchall()  
 self.columns = []  
 for i in self.cursor.description:  
 self.columns.append(list(i)[0])  
 self.data = []  
 self.data.append(self.rows)  
 self.data.append(self.columns)  
 # print(self.data)  
 # return self.data  
 except Exception as e:  
 print(e)  
 self.data = None  
 return self.data  
 return self.data  
  
 def delete(self, table, cond):  
 try:  
 self.attr = cond.split(':')[0]  
 self.value = cond.split(':')[1]  
 self.connection.commit()  
 self.cursor.execute(f'DELETE FROM public."{table}" WHERE {self.attr} = \'{self.value}\' ')  
 self.connection.commit()  
 except Exception as e:  
 print(e)  
 return False  
 return True  
  
 def insert(self, table, values):  
 try:  
 self.val = tuple(values)  
 self.connection.commit()  
 self.cursor.execute(f"INSERT INTO public.\"{table}\" VALUES {self.val}")  
 self.connection.commit()  
 except Exception as e:  
 print(e)  
 return False  
 return True  
  
 def get\_columns(self, table):  
 try:  
 self.connection.commit()  
 self.cursor.execute(f'SELECT \* FROM public."{table}"')  
 self.rows = self.cursor.fetchall()  
 self.columns = []  
 for i in self.cursor.description:  
 self.columns.append(list(i)[0])  
 return self.columns  
 except Exception as e:  
 print(e)  
 return False  
  
 def update(self, table, attr, update\_attr, new\_value):  
 try:  
 self.attr = attr.split(':')[0]  
 self.value = attr.split(':')[1]  
 self.connection.commit()  
 self.cursor.execute(  
 f"UPDATE public.\"{table}\" SET {update\_attr} = '{new\_value}' WHERE {self.attr} = '{self.value}'")  
 self.connection.commit()  
 except Exception as e:  
 print(e)  
 return False  
 return True  
  
 def delete\_table(self, table):  
 try:  
 self.connection.commit()  
 self.cursor.execute(f'DELETE FROM public."{table}"')  
 self.connection.commit()  
 except Exception as e:  
 print(e)  
 return False  
 return True  
  
 def generate(self, table, count1, count2):  
 try:  
 count1 = str(int(count1) + 1)  
 count2 = str(int(count2)+int(count1))  
  
 if table == "coach":  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.coach  
 SELECT chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),   
 floor(65 + random()\*25)::int,  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int)  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 elif table == 'team':  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.team  
 SELECT generate\_series,  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),   
 generate\_series,  
 generate\_series  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 elif table == 'player':  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.player  
 SELECT chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int) || chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int),   
 floor(65 + random()\*25)::int,  
 chr(floor(65 + random()\*25)::int) ||   
 chr(floor(65 + random()\*25)::int)  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 elif table == 'match':  
 self.connection.commit()  
 self.cursor.execute(f"""INSERT INTO public.match  
 SELECT generate\_series,  
 DateTime.toZone('UTC'),  
 floor(65 + random()\*25)::int,  
 floor(65 + random()\*25)::int  
 FROM generate\_series({count1},{count2}) """)  
 self.connection.commit()  
 else:  
 print("Such table doesn't exist")  
 except Exception as e:  
 print(e)  
 return False  
 return True  
  
 def search\_by\_attr(self, table1, table2, attr, value):  
 try:  
 self.connection.commit()  
 self.cursor.execute(f"""SELECT \* FROM public.{table1} as first INNER JOIN public.{table2} as second on first.{attr} = second.{attr} WHERE {value}""")  
  
 self.rows = self.cursor.fetchall()  
 self.columns = []  
 for i in self.cursor.description:  
 self.columns.append(list(i)[0])  
 self.data = []  
 self.data.append(self.rows)  
 self.data.append(self.columns)  
 return self.data  
 except Exception as e:  
 print(e)  
 return None  
  
 def text\_search(self, mode, word, table):  
 self.connection.commit()  
 try:  
 if mode == '1':  
 self.words = word.split()  
 self.words = '|'.join(self.words)  
 self.cursor.execute(  
 f'select \* from public."{table}" where not (to\_tsvector(coach\_name) @@ to\_tsquery(\'{self.words}\'))')  
 elif mode == '2':  
 self.words = word.split()  
 self.words = '&'.join(self.words)  
 self.cursor.execute(  
 f'select \* from public."{table}" where (to\_tsvector(coach\_name) @@ to\_tsquery(\'{self.words}\'))')  
 else:  
 return None  
 self.rows = self.cursor.fetchall()  
 self.columns = []  
 for i in self.cursor.description:  
 self.columns.append(list(i)[0])  
 self.data = []  
 self.data.append(self.rows)  
 self.data.append(self.columns)  
 except Exception as e:  
 print(e)  
 self.data = None  
 return self.data  
 return self.data

Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити. Для цього в ньому використовується бібліотека – psycopg2.