

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: “Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-91

Неживий Б. М.

Перевірив:

Павловський В. І.

Київ – 2021

# Постановка задачі

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

*Загальне 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**

Посилання на репозиторій у GitHub з вихідним кодом програми та прикладеним звітом:  [https://github.com/n0tark/lab-pp](https://github.com/vladsel/database)

Використана мова програмування: Python 3.10.

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), time (для виміру часу запиту пошуку, що у 3 завданні)

Використаний шаблон проектування: MVC.

**Відомості про предметну галузь з лабораторної роботи №1**

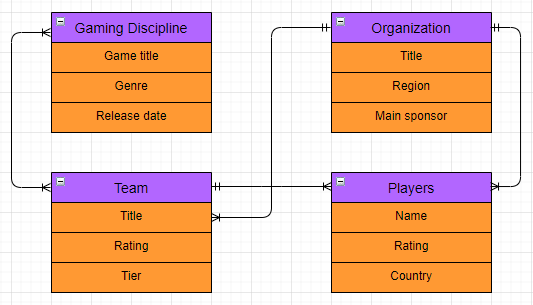


Рисунок 1 - ER-діаграма побудована за нотацією “Пташиної лапки (Crow’s foot)”, задана ER-діаграма була побудована у додатку [draw.io](https://app.diagrams.net/)

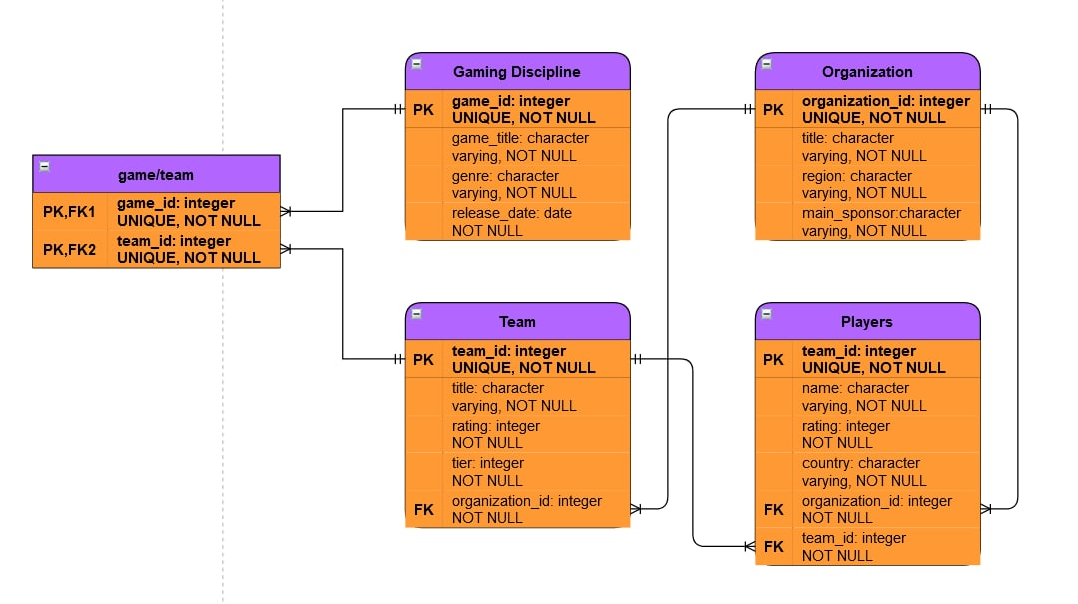


Рисунок 2 - Схема бази даних, побудовано у додатку [draw.io](https://app.diagrams.net/)

**Таблиця 1 - Опис структури БД.**

|  |  |  |
| --- | --- | --- |
| Відношення | Атрибут | Тип атрибуту |
| **gaming discipline –** містить дані про ігрову дисципліну. | **game\_id –** унікальний ідентифікатор  **title –** назва гри  **genre –** жанр  **release\_date –** офіційна дата виходу | **integer** (числовий)  **character varying** (рядок)  **character varying** (рядок)  **date** (дата) |
| **organization –** містить дані про кіберспортивну організацію. | **organization\_id** – унікальний  ідентифікатор  **title –** назва  **region –** основний регіон організації  **main\_sponsor –** основний спонсор | **integer** (числовий)    **character varying** (рядок)  **character varying** (рядок)  **character varying** (рядок) |
| **team –** містить дані про професійні команди | **team\_id** – унікальний ідентифікатор  **title** – назва команди  **rating –** місце в рейтингу команд  **tier –** рівень команди (1 – найвищий) | **integer** (числовий)  **character varying** (рядок)  **integer** (числовий)  **integer** (числовий) |
| **players** – містить дані про гравців | **player\_id -** унікальний ідентифікатор  **name –** ім’я гравця  **rating –** місце в рейтингу гравців  **country** – рідна країна гравця | **integer** (числовий)  **character varying** (рядок)  **integer** (числовий)  **character varying** (рядок) |
| **game/team** –  відношення дисциплін до команд | **game\_id–** ідентифікатор дисципліни  **team\_id –** ідентифікатор команди | **integer** (числовий)  **integer** (числовий) |

В вибраній базі даних «Кіберспорт» можна виділити наступні таблиці: загальні відомості про ігрову дисципліну (gaming discipline), конкретну організацію (organization), команди, що в них входять (team), та члени цих команд, тобто гравці (player), відношення команди до гри (game/team).

Стовпці заданих таблиць:

* + - 1. discipline: game\_id, title, genre, release\_date.
      2. organization: organization\_id, title, region, main\_sponsor.
      3. team: team\_id, title, rating, tier.
      4. player: player\_id, name, rating, country.

**Структура програми**

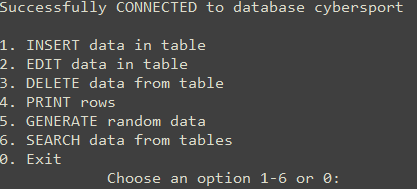
За шаблоном проектування MVC, додаток складається з таких модулей:

*model.py* — підключається до БД та виконує операції – SELECT, INSERT, DELETE, UPDATE та, більш складні операції з БД.

*controller.py* — головне та допоміжні меню, для зручного керування БД.

*mainl.py* — точка входу в програму, та підключення до БД.

# Схема меню користувача



На зображенні терміналу показане початкове меню, в якому ми бачимо функції для управління БД. Кожна команда відповідає за конкретну 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.

***Методи реалізовані до пункту 1 завдання лабораторної роботи:***

1. INSERT data in table — відповідає за виклик функції вставки даних в таблицю бази;
2. EDIT data in table — викликає функцію редагування даних у таблиці бази даних;
3. DELETE data from table — викликає функцію видалення даних у таблиці бази даних;
4. PRINT rows — викликає функцію виводу даних з таблиці бази даних.

***Метод реалізований до пункту 2 завдання лабораторної роботи:***

1. GENERATE random data — дозволяє користувачеві заповнити таблицю, або всю базу даних випадково згенерованими даними.

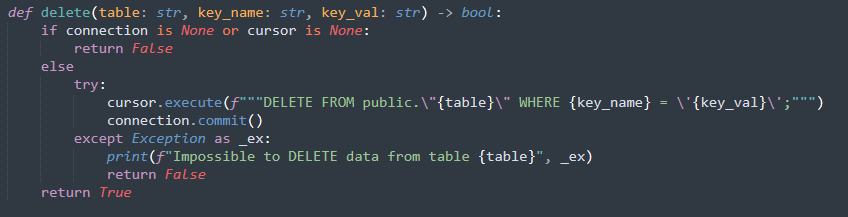
Кожна з описаних вище функцій викликає допоміжну функцію select\_table(), даний скрипт дозволяє користувачеві обирати таблицю

***Метод реалізований до пункту 3 завдання лабораторної роботи:***

1. SEARCH data from tables — викликає функцію пошуку даних у таблицях за атрибутами та поєднання таблиць за ключем.

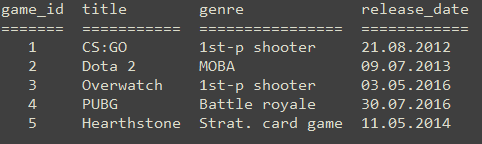
**Завдання 1**

***Запит на видалення***

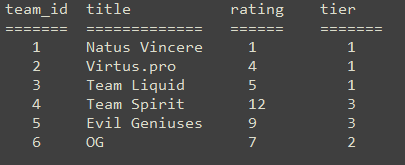
******

Для перевірки роботи розглянемо запити на видалення даних з дочірньої таблиці **team** та батьківської таблиці **discipline**.

Таблиця **discipline** до видалення даних:

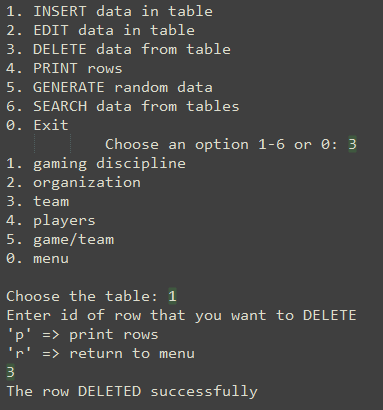


Таблиця **team** до видалення даних:

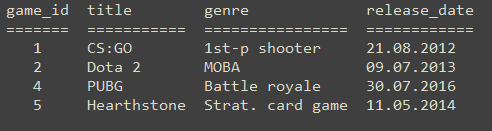


У даній програмній реалізації видалення запису з батьківської таблиці, який зв’язаний з дочірньою таблицею, буде видалено каскадно, тобто всі дані з цим зовнішнім ключем будуть видалені.

Видалимо з таблиці **discipline** рядок з game\_id = 3:



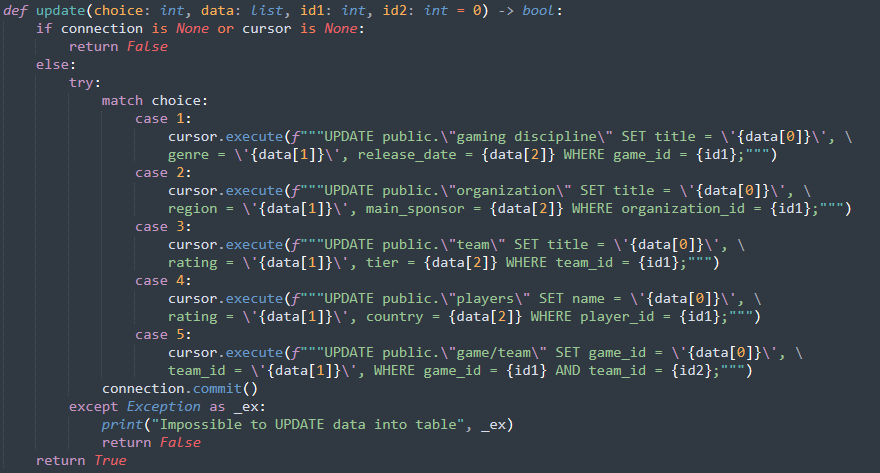
Таблиця **discipline** після видалення даних:



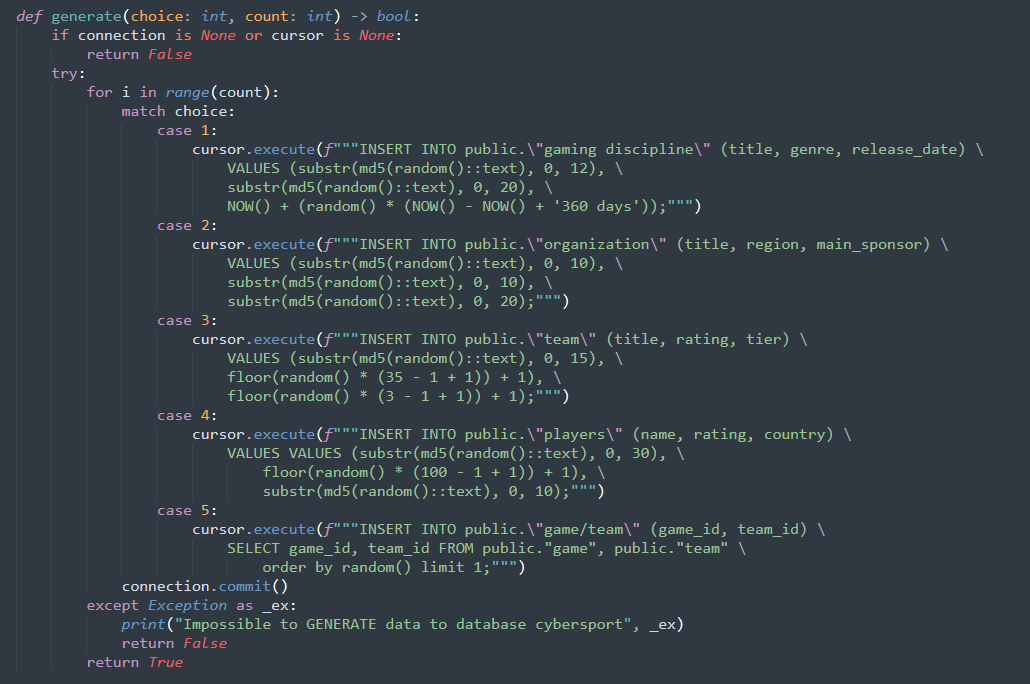
***Запит на вставку поля***



***Запит на редагування рядків***

**

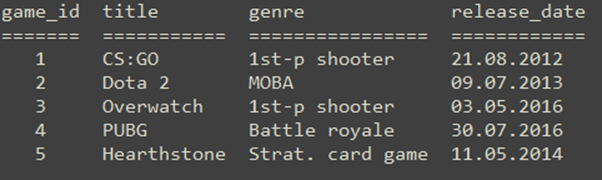
**Завдання 2**



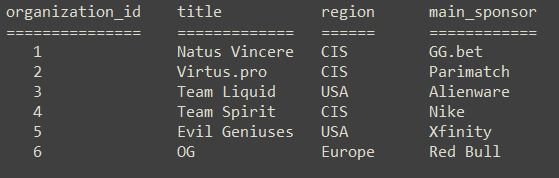
Вставимо по 5 псевдорандомізованих записів у кожну таблицю.

Записи таблиць перед вставкою даних:

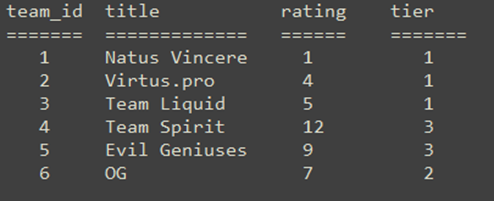
Таблиця **gaming discipline**:



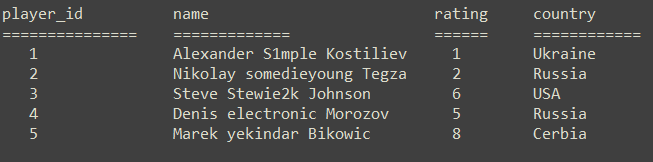
Таблиця **organization**:



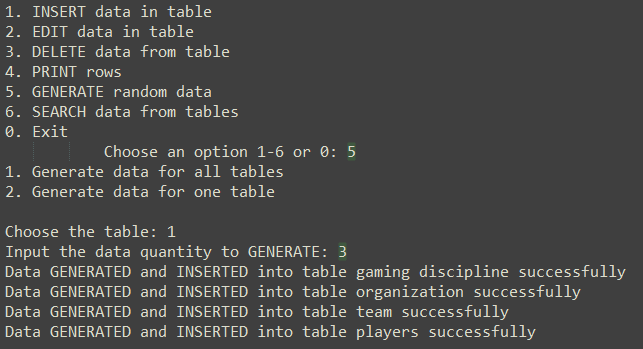
Таблиця **team**:



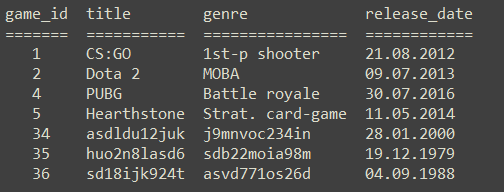
Таблиця **players**:



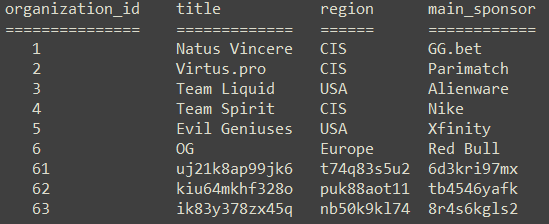
Таблиці після вставки згенерованих записів:



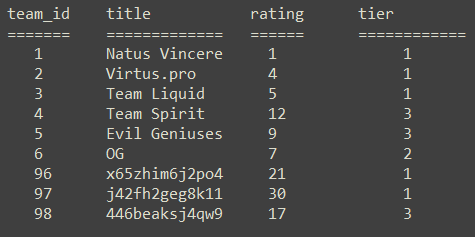
Таблиця **gaming discipline**:



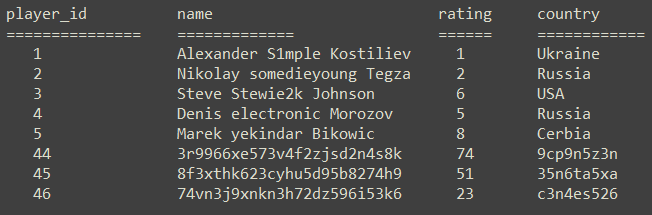
Таблиця **organization**:



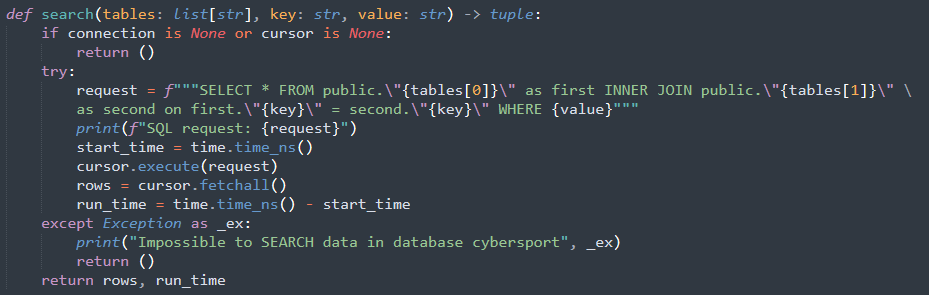
Таблиця **team**:



Таблиця **players**:



**Завдання 3**



**Завдання 4**

Код програмного модулю “model.py”

import psycopg2

import time

cursor = None

connection = None

def connect():

try:

global cursor, connection

connection = psycopg2.connect(

host="localhost",

user="postgres",

password="postgres",

database="cybersport\_db",

port="5432"

)

cursor = connection.cursor()

print("Successfully CONNECTED to database cybersport")

# cursor.execute("SELECT version();")

# print(f"Server version {cursor.fetchone()}")

except Exception as \_ex:

print("Failed CONNECTION to database cybersport", \_ex)

def disconnect():

try:

cursor.close()

connection.close()

print("Successfully DISCONNECTED from database cybersport")

except Exception as \_ex:

print("Impossible to DISCONNECT from database cybersport", \_ex)

def insert(choice: int, data: list) -> bool:

if connection is None or cursor is None:

return False

else:

try:

match choice:

case 1:

cursor.execute(f"""INSERT INTO public.\"gaming discipline\" (title, genre, release\_date) \

VALUES (\'{data[0]}\', \'{data[1]}\', {data[2]});""")

case 2:

cursor.execute(f"""INSERT INTO public.\"organization\" (title, region, main\_sponsor) \

VALUES (\'{data[0]}\', \'{data[1]}\', {data[2]});""")

case 3:

cursor.execute(f"""INSERT INTO public.\"team\" (title, rating, tier) \

VALUES (\'{data[0]}\', \'{data[1]}\', {data[2]});""")

case 4:

cursor.execute(f"""INSERT INTO public.\"players\" (name, rating, country) \

VALUES (\'{data[0]}\', \'{data[1]}\', {data[2]});""")

case 5:

cursor.execute(f"""INSERT INTO public.\"game/team\" (game\_id, team\_id) \

VALUES (\'{data[0]}\', {data[1]});""")

connection.commit()

except Exception as \_ex:

print("Impossible to INSERT data into table", \_ex)

return False

return True

def delete(table: str, key\_name: str, key\_val: str) -> bool:

if connection is None or cursor is None:

return False

else

try:

cursor.execute(f"""DELETE FROM public.\"{table}\" WHERE {key\_name} = \'{key\_val}\';""")

connection.commit()

except Exception as \_ex:

print(f"Impossible to DELETE data from table {table}", \_ex)

return False

return True

def select\_by\_key(table: str, key\_name: str, key\_val: str) -> list:

if connection is None or cursor is None:

return []

else:

try:

cursor.execute(f"""SELECT \* FROM public.\"{table}\" WHERE {key\_name} = \'{key\_val}\';""")

except Exception as \_ex:

print(f"Impossible to SELECT data from table {table} by key {key\_name}", \_ex)

return []

return cursor.fetchall()

def select\_by\_table(table: str, quantity: str = '100', offset: str = '0') -> list:

if connection is None or cursor is None:

return []

else:

try:

if table == 'game/team':

cursor.execute(f"""SELECT \* FROM public.\"{table}\" ORDER BY {"game\_id"} \

ASC limit {quantity} offset {offset};""")

else:

cursor.execute(f"""SELECT \* FROM public.\"{table}\" ORDER BY {table + "\_id"} \

ASC limit {quantity} offset {offset};""")

except Exception as \_ex:

print(f"Impossible to SELECT data from table {table}", \_ex)

return []

return cursor.fetchall()

def update(choice: int, data: list, id1: int, id2: int = 0) -> bool:

if connection is None or cursor is None:

return False

else:

try:

match choice:

case 1:

cursor.execute(f"""UPDATE public.\"gaming discipline\" SET title = \'{data[0]}\', \

genre = \'{data[1]}\', release\_date = {data[2]} WHERE game\_id = {id1};""")

case 2:

cursor.execute(f"""UPDATE public.\"organization\" SET title = \'{data[0]}\', \

region = \'{data[1]}\', main\_sponsor = {data[2]} WHERE organization\_id = {id1};""")

case 3:

cursor.execute(f"""UPDATE public.\"team\" SET title = \'{data[0]}\', \

rating = \'{data[1]}\', tier = {data[2]} WHERE team\_id = {id1};""")

case 4:

cursor.execute(f"""UPDATE public.\"players\" SET name = \'{data[0]}\', \

rating = \'{data[1]}\', country = {data[2]} WHERE player\_id = {id1};""")

case 5:

cursor.execute(f"""UPDATE public.\"game/team\" SET game\_id = \'{data[0]}\', \

team\_id = \'{data[1]}\', WHERE game\_id = {id1} AND team\_id = {id2};""")

connection.commit()

except Exception as \_ex:

print("Impossible to UPDATE data into table", \_ex)

return False

return True

def generate(choice: int, count: int) -> bool:

if connection is None or cursor is None:

return False

try:

for i in range(count):

match choice:

case 1:

cursor.execute(f"""INSERT INTO public.\"gaming discipline\" (title, genre, release\_date) \

VALUES (substr(md5(random()::text), 0, 12), \

substr(md5(random()::text), 0, 20), \

NOW() + (random() \* (NOW() - NOW() + '360 days'));""")

case 2:

cursor.execute(f"""INSERT INTO public.\"organization\" (title, region, main\_sponsor) \

VALUES (substr(md5(random()::text), 0, 10), \

substr(md5(random()::text), 0, 10), \

substr(md5(random()::text), 0, 20);""")

case 3:

cursor.execute(f"""INSERT INTO public.\"team\" (title, rating, tier) \

VALUES (substr(md5(random()::text), 0, 15), \

floor(random() \* (35 - 1 + 1)) + 1), \

floor(random() \* (3 - 1 + 1)) + 1);""")

case 4:

cursor.execute(f"""INSERT INTO public.\"players\" (name, rating, country) \

VALUES VALUES (substr(md5(random()::text), 0, 30), \

floor(random() \* (100 - 1 + 1)) + 1), \

substr(md5(random()::text), 0, 10);""")

case 5:

cursor.execute(f"""INSERT INTO public.\"game/team\" (game\_id, team\_id) \

SELECT game\_id, team\_id FROM public."game", public."team" \

order by random() limit 1;""")

connection.commit()

except Exception as \_ex:

print("Impossible to GENERATE data to database cybersport", \_ex)

return False

return True

def search(tables: list[str], key: str, value: str) -> tuple:

if connection is None or cursor is None:

return ()

try:

request = f"""SELECT \* FROM public.\"{tables[0]}\" as first INNER JOIN public.\"{tables[1]}\" \

as second on first.\"{key}\" = second.\"{key}\" WHERE {value}"""

print(f"SQL request: {request}")

start\_time = time.time\_ns()

cursor.execute(request)

rows = cursor.fetchall()

run\_time = time.time\_ns() - start\_time

except Exception as \_ex:

print("Impossible to SEARCH data in database cybersport", \_ex)

return ()

return rows, run\_time

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

Функція connect() – намагається підключитись до БД.

Функція disconnect() – намагається відключитись від БД.

Функція insert(choice, data) – намагається вставити в таблицю по номеру choice дані зі списку data.

Функція delete(table, key\_name, key\_val) – намагається видали з таблиці table рядок з ключем key\_name, який == key\_val.

Функція select\_by\_key(table, key\_name, key\_val) – намагається взяти дані з таблиці table рядок з ключем key\_name, який == key\_val.

Функція select\_by\_table(table, quantity, offset) – намагається взяти дані з таблиці table та відсортувати їх у порядку зростання по primary key.

Функція update(choice, data, id1, id2) – намагається змінити в таблиці choice, даними зі списку data, рядок з первинним ключем, який == id, якщо первинний ключ складений, тоді має дорівнювати id1 та id2.

Функція generate(choice, count) – генерує дані і намагається вставити в таблицю choice, count разів ці даніазрр.

Функція search(tables, key, value) ­– пробує об’єднати і взяти дані з таблиць.