

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-01

Роговий Д.С.

Перевірив:

Павловський В. І.

Київ – 2022

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

Деталізоване завдання:

Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати вилучення рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні внесення нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими не мовою програмування, а відповідним SQL-запитом!

Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

Програмний код організувати згідно шаблону Model-View-Controller(MVC). Приклад організації коду згідно шаблону доступний за даним посиланням. При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати лише мову SQL (без ORM).

**Логічна модель бази даних**

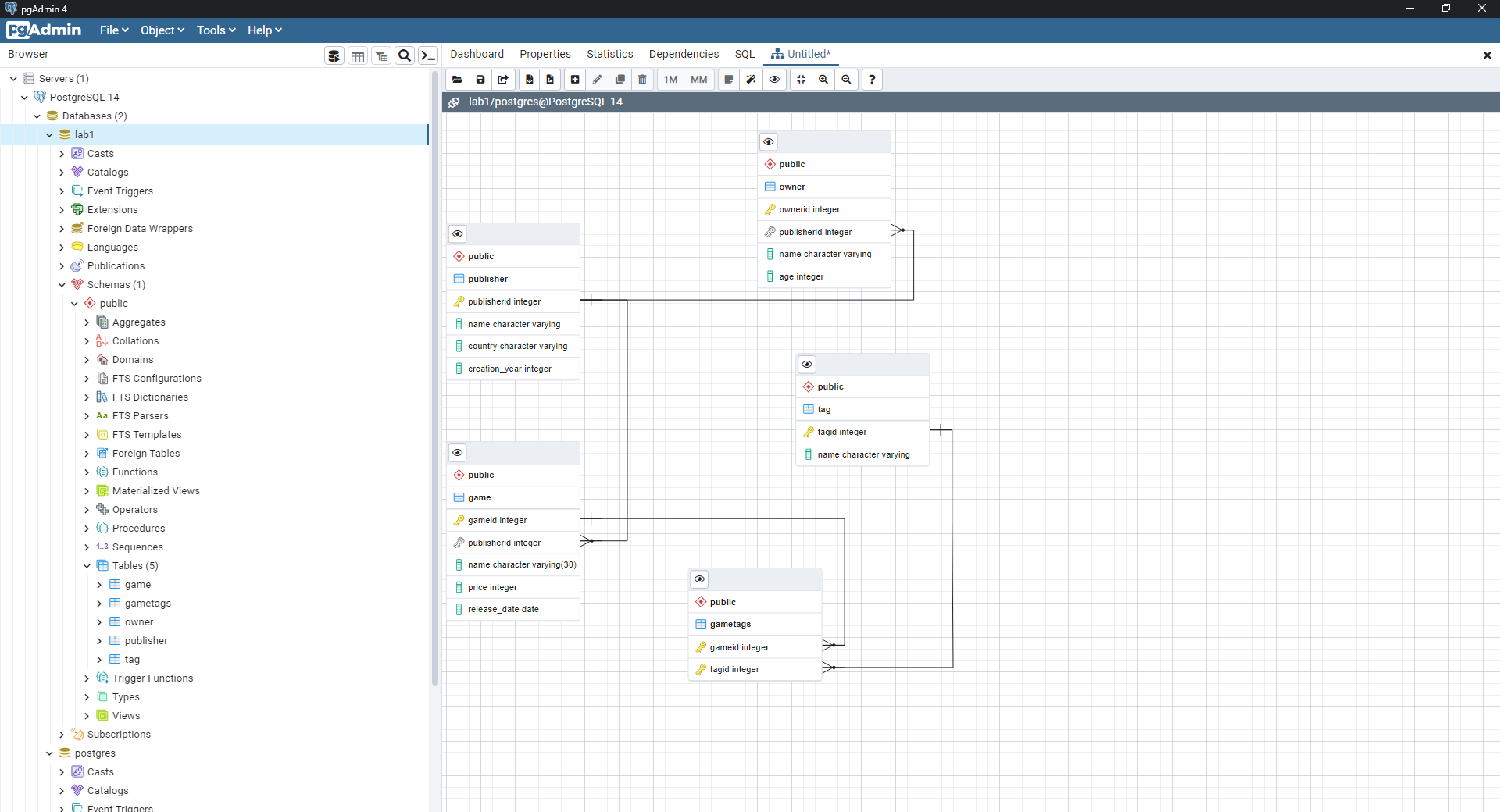


Рисунок 1 – Логічна модель бази даних

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувалась мова програмування python та середовище розробки Visual Studio 2022.

Для підключення до серверу бази даних PostgreSQL використано бібліотеку psycopg2.

**Шаблон проектування програмного додатку**

MVC – шаблон проектування, що використовується у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Це файл Model.py.

View – відповідає за ввід/вивід інформації. Це файл View.py.

Controller – Представляє клас, що обробляє ввід користувача, подає відповідну команду до Model.py і очікує результат, який потім виводить користувачу. Це файл Controller.py.

**Структура програми та її зміст**

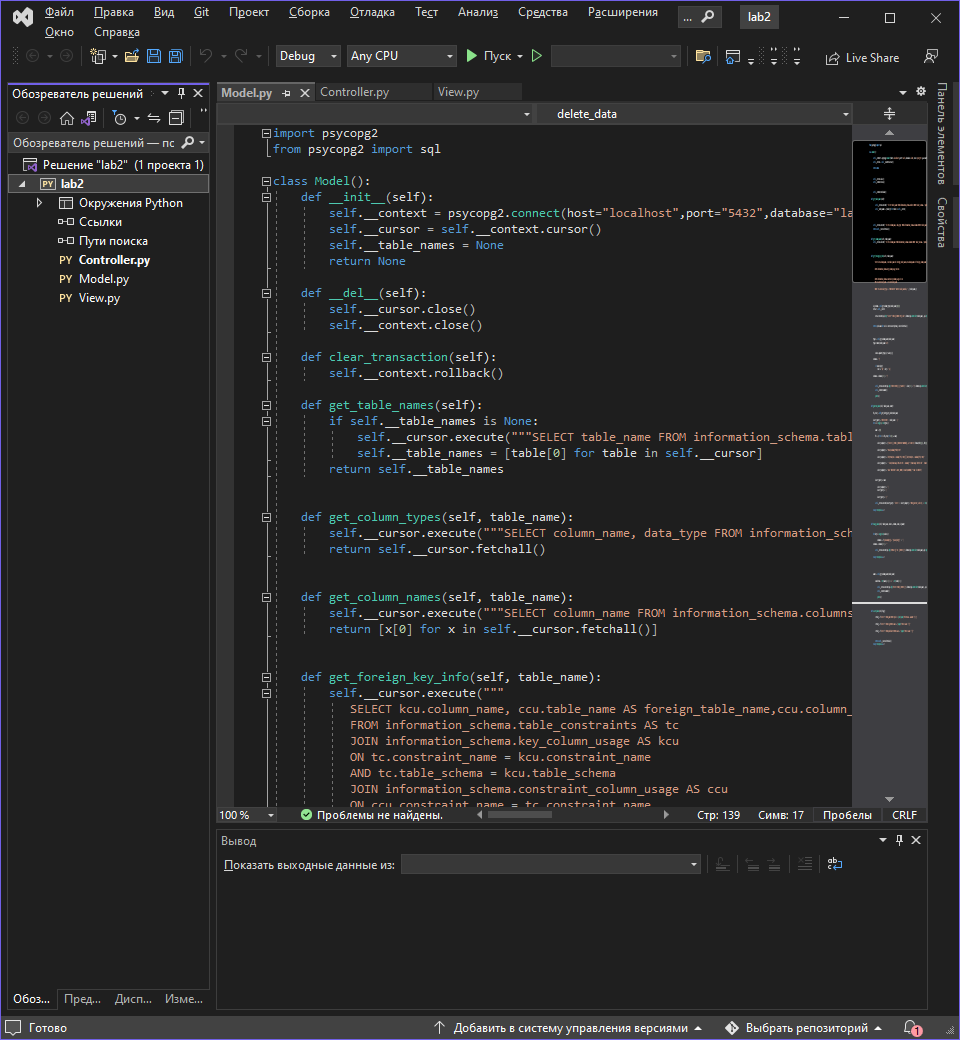


Рисунок 2 – Структура програми

1. View.py– файл, що відповідає за відображення програми у вікні, та приймає ввід/вивід користувача.
2. Controller.py – обробляє ввід користувача, подає відповідну команду до Model.py і очікує результат, який потім виводить користувачу.
3. Model.py – виконує складні операції з БД з конкретними таблицями БД.

**Структура меню програми**

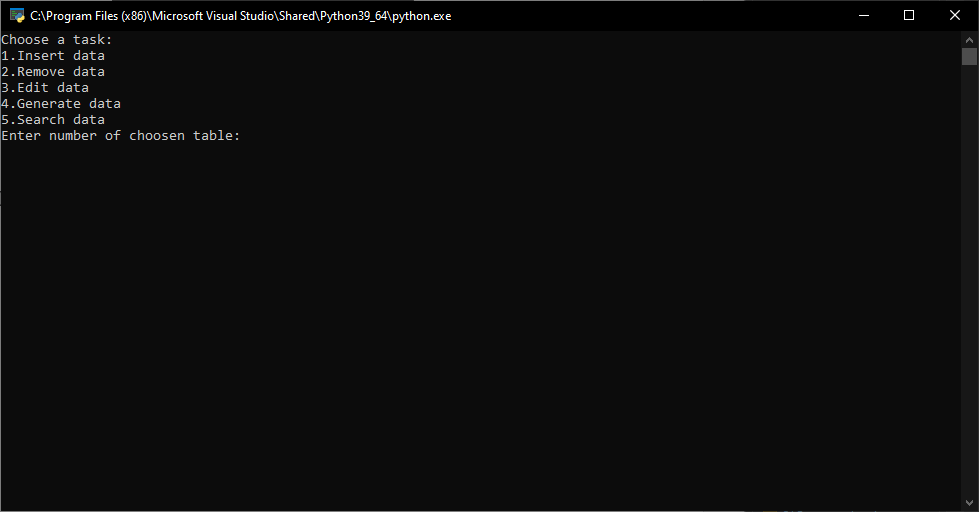


Рисунок 3 – структура меню вибору завдання

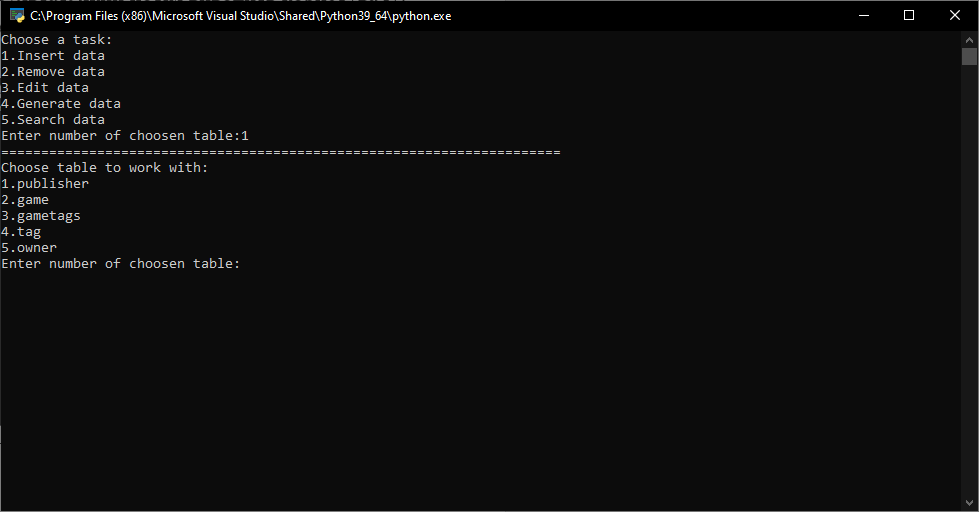


Рисунок 4 – структура меню вибору таблиці

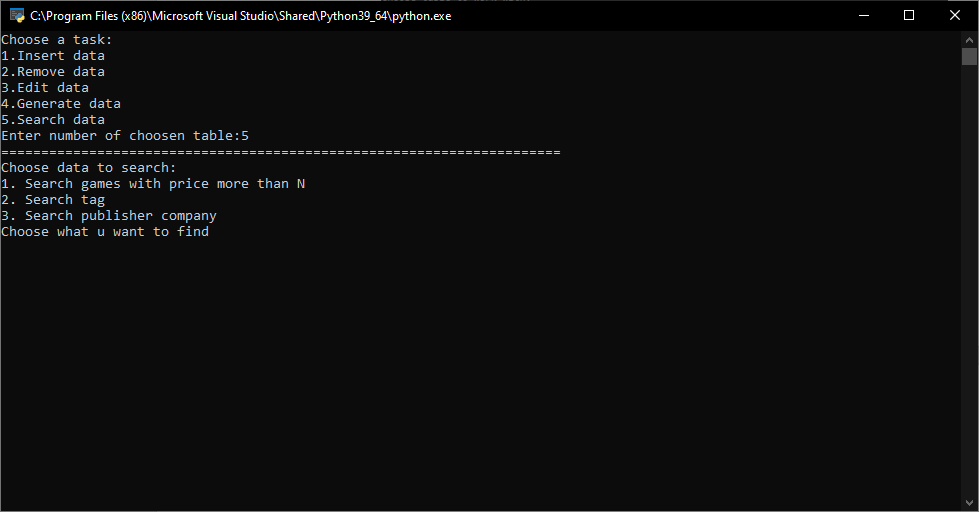


Рисунок 5 – структура меню пошуку

**Лістинги фрагментів програми з запитом пошуку**

В данній функції реалізовано три варіанти параметричного пошуку(див рисунок 5). Параметр flag функція отримує з методу search\_menu() класу Controler. Данні для пошуку функція отримує з input(). Обробка помилок відбувається за допомогою перехоплення try … except.

if flag == 1:

string = f"SELECT \* FROM game WHERE price < {int(input('Write max. amount: '))}"

elif flag == 2:

string = f"SELECT \* FROM tag WHERE name = '{input('Write name: ')}'"

elif flag == 3:

string = f"SELECT \* FROM publisher WHERE name = '{input('Write name: ')}'"

try:

self.\_\_cursor.execute(string)

return self.\_\_cursor.fetchall()

except Exception as e:

print(e)

**Лістинги фрагментів програми з запитом на створення**

Функція виконує створення нового об’єкту в заданій параметром таблиці, заповнюючи його значеннями з value. Спочатку ми прибираємо стовпчик з первинним ключем,оскільки він генеруєтся автоматично, потім отримуємо назви усіх колонок таблиці, і перетворюємо list value на словник values, кожен елемент якого має вигляд {‘назва колонки’:вміст}. Потім проходимося по цьому словнику, формуючи запит SQL. Перевірка виконуєтся за допомогою перехоплення помилок.

def insert\_data(self, table\_name, value):

types = self.get\_column\_names(table\_name)

types.remove(table\_name+'id')

values={}

for i in range(len(types)):

values.update({types[i]:value[i]})

line = ''

columns = '('

for key in values:

if values[key]:

line += '%(' + key + ')s,'

columns += key + ','

columns = columns[:-1] + ')'

try:

self.\_\_cursor.execute(sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')').format(sql.Identifier(table\_name), sql.SQL(columns)),values)

self.\_\_context.commit()

except Exception as e:

print(e)

**Лістинг фрагментів програми з запитом на вилучення**

Функція виконує видалення елемента таблиці. Назва таблиці і вміст обраного для видалення елемента передаєтся в якості параметрів. Отримуємо назву усіх колонок таблиці, потім формуємо рядок condition виду «назва\_колонки = вміст колонки для обраного об’єкту», проходячи циклом по усім колонкам. Далі через sql запит видаляємо обраний об’єкт. Перевірка виконуєтся за допомогою перехоплення помилок.

def delete\_data(self, table\_name, cond):

names = self.get\_column\_names(table\_name)

for i in range(len(names)):

condition = str(names[i-1]) + "= '" + str(cond[i-1]) + "'"

try:

self.\_\_cursor.execute(sql.SQL('DELETE FROM {} WHERE {}').format(sql.Identifier(table\_name), sql.SQL(condition)))

self.\_\_context.commit()

except Exception as e:

print(e)

**Лістинг фрагментів програми з запитом редагування**

Функція змінює вміст вже існуючого об’єкту. У функцію в якості параметрів передаются назва таблиці, данні для зміни, колонки у яких будуть змінюватися данні, первинний ключ обраного об’єкту та назву колонки, в якій зберігаєтся первинний ключ заданної колонки. Формуємо строки columns та string для параметричного sql запиту. Перевірка виконуєтся за допомогою перехоплення помилок.

def change\_data(self, table\_name, values, column, cond, id\_name):

string = f"{id\_name} = '{cond}'"

columns = ''

for key in range(len(values)):

if values[key]:

columns += f"{column[key]} = '{values[key]}'" + ','

columns = columns[:-1] + ''

try:

self.\_\_cursor.execute(sql.SQL('UPDATE {} SET {} WHERE {}').format(sql.Identifier(table\_name),sql.SQL(columns), sql.SQL(string)))

self.\_\_context.commit()

except Exception as e:

print(e)

**Лістинги фрагментів програм генерування випадкових даних в таблицях БД**

Функція генерує задану кількість випадкових значень у заданій таблиці. Спочатку ми отримуємо усі типи колонок таблиці, потім, за допомогою циклу формуємо великий sql запит. Перша його частина – insert\_query, де ми встановлюэмо в яку таблицю будуть генеруватися значення. Другий етап – самі параметри генерації – select\_subquery. Проходячи циклом по списку типів ми додаємо до рядку select\_subquery запит на генерацію, в залеженості від типу данних певного рядка, присутня окрема генерація для зовнішніх ключів. Потім, за допомогою двох вищезгаданих рядків формуєтся і виконуєтся sql запит. Перевірка виконуєтся за допомогою перехоплення помилок.

def generate\_data(self, table\_name, count):

types = self.get\_column\_types(table\_name)

fk\_array = self.get\_foreign\_key\_info(table\_name)

select\_subquery = ""

insert\_query = "INSERT INTO " + table\_name + " ("

for i in range(1, len(types)):

t = types[i]

name = t[0]

type = t[1]

fk = [x for x in fk\_array if x[0] == name]

if fk:

select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))

elif type == 'integer':

select\_subquery += 'trunc(random()\*100)::INT'

elif type == 'character varying':

select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)'

elif type == 'date':

select\_subquery += """ date(timestamp '2014-01-10' + random() \* (timestamp '2020-01-20' - timestamp '2014-01-10'))"""

elif type == 'time without time zone':

select\_subquery += "time '00:00:00' + DATE\_TRUNC('second',RANDOM() \* time '24:00:00')"

else:

continue

insert\_query += name

if i != len(types) - 1:

select\_subquery += ','

insert\_query += ','

else:

insert\_query += ') '

try:

self.\_\_cursor.execute(insert\_query + "SELECT " + select\_subquery + "FROM generate\_series(1," + str(count) + ") AS ser")

self.\_\_context.commit()

except Exception as e:

print(e)

**Результати роботи програми**

Далі будуть наведені приклади виконная операцій додавання, видалення, модифікації, генерування і пошуку елементів в таблиці Game. На рисунку 6 показано стан таблиці до виконання вищевказаних операцій

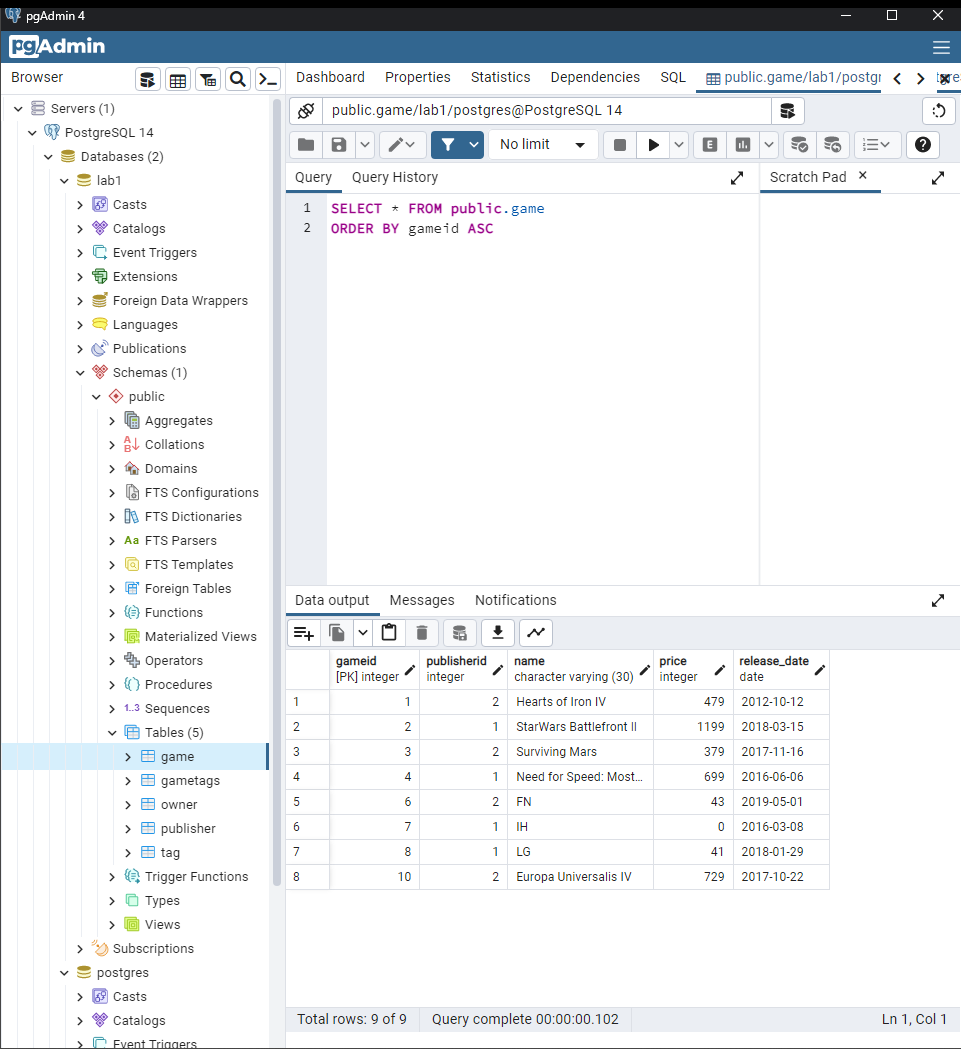


Рисунок 6 – таблиця до модифікацій

**Скріншоти результатів виконання операції додавання нового елементу до таблиці**

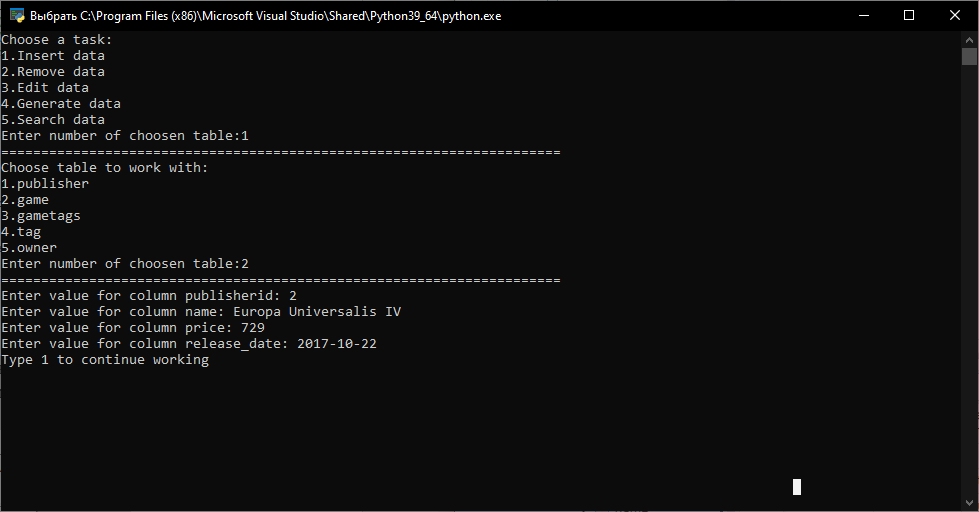


Рисунок 7 – параметри новоствореного елемента

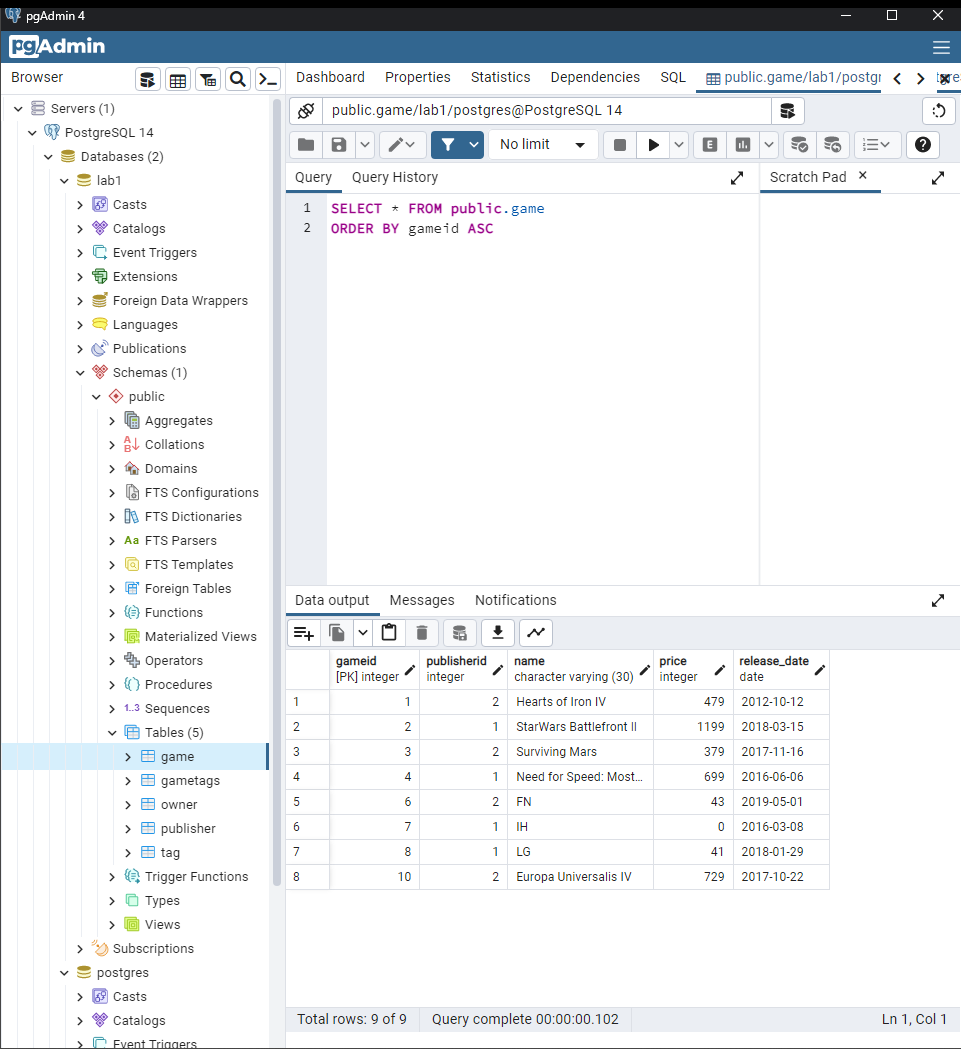


Рисунок 8 – Таблиця з доданим елементом

**Скріншот результатів виконання операції видалення елементу з таблиці**

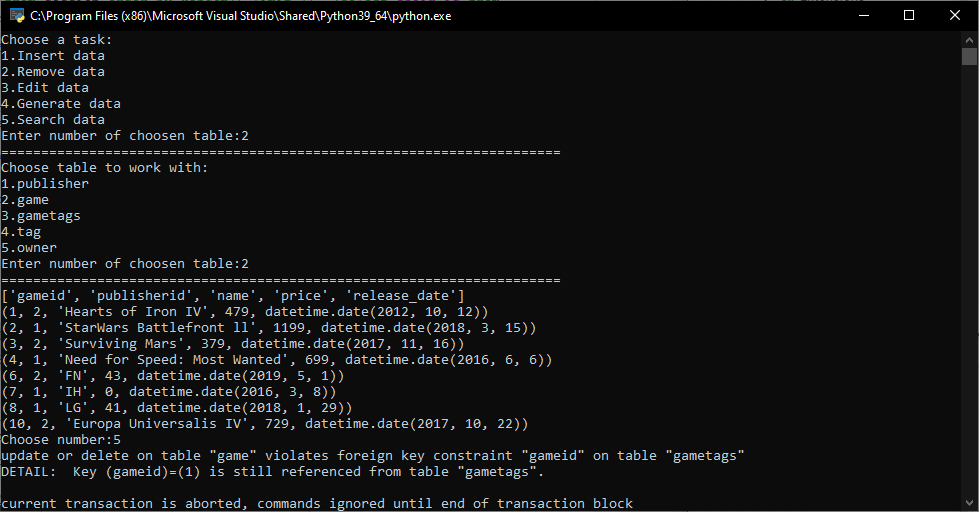


Рисунок 9 – вибір елементу для видалення

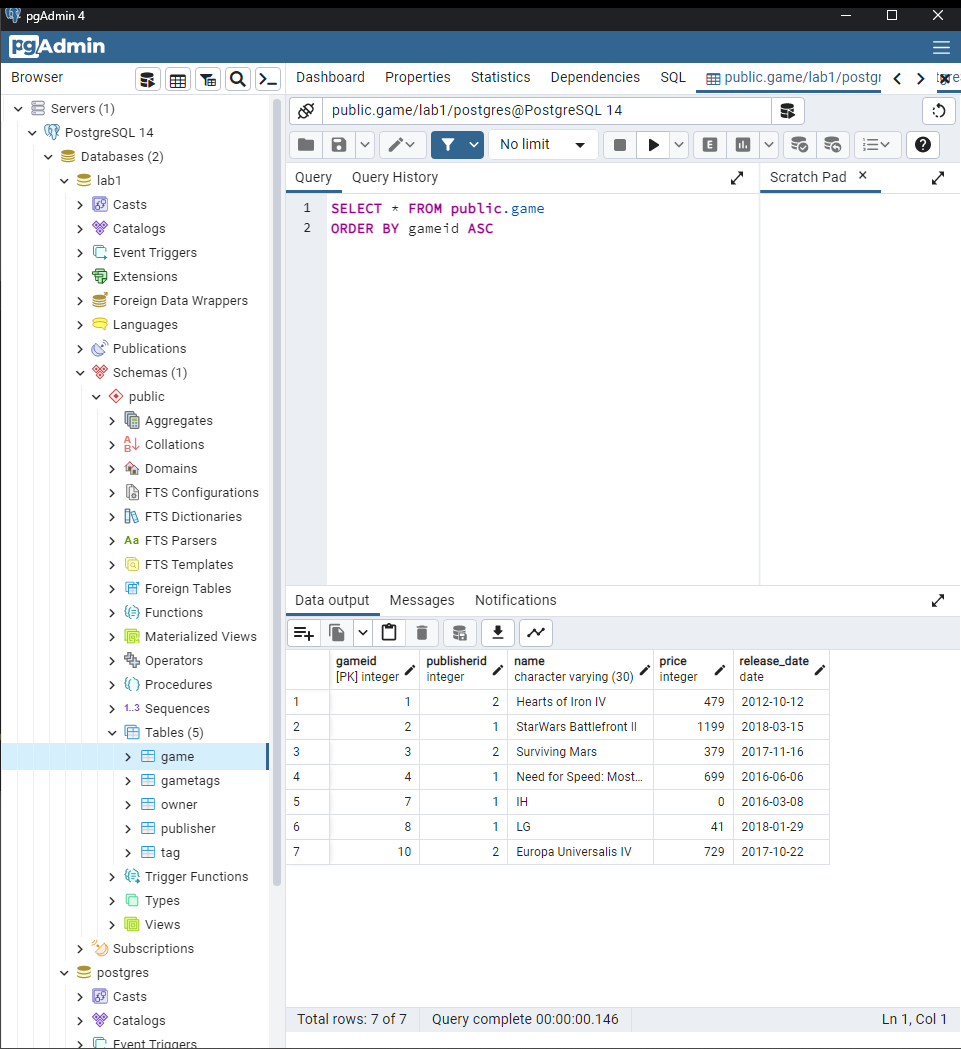


Рисунок 10 – Таблиця після видалення елементу

**Скріншот результатів модифікації елементу в таблиці**

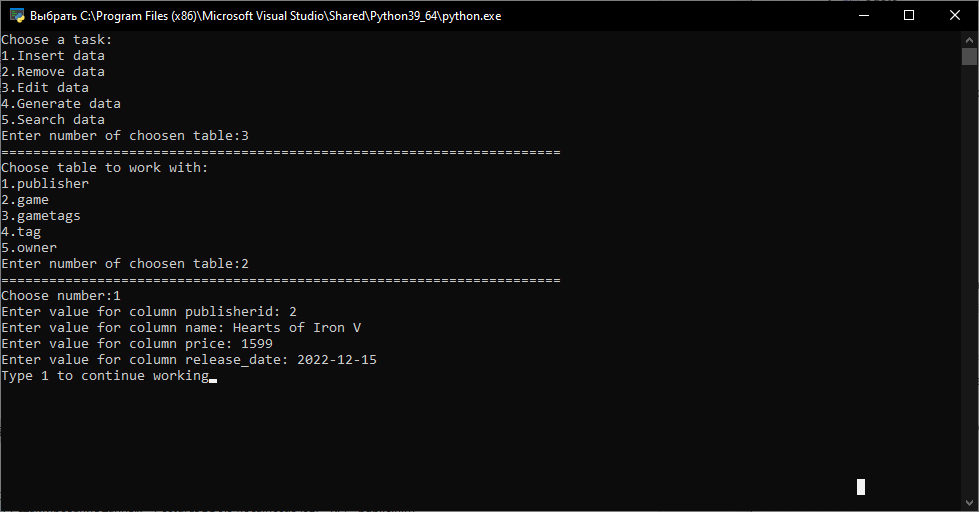


Рисунок 11 – Зміна параметрів елементу

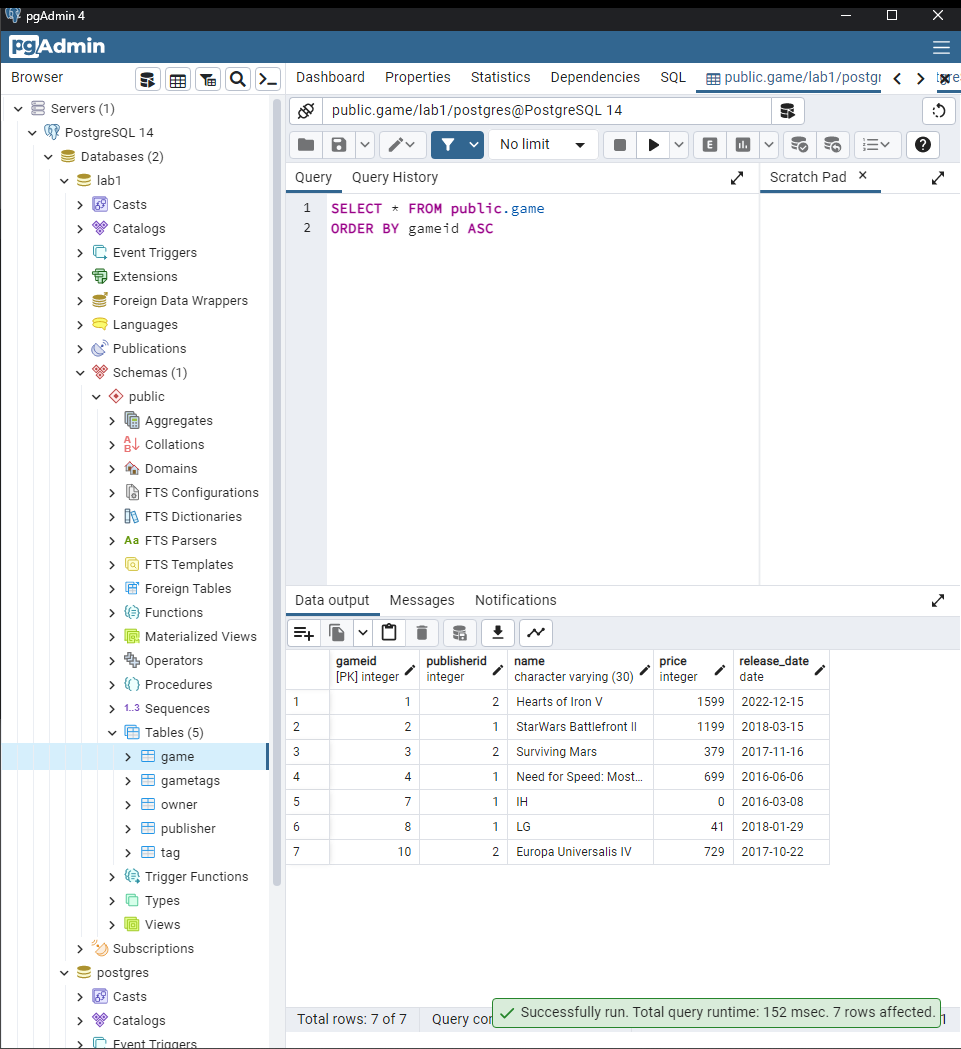


Рисунок 12 – таблиця зі зміненим елементом

**Скріншот результату генерації елементів в таблиці**

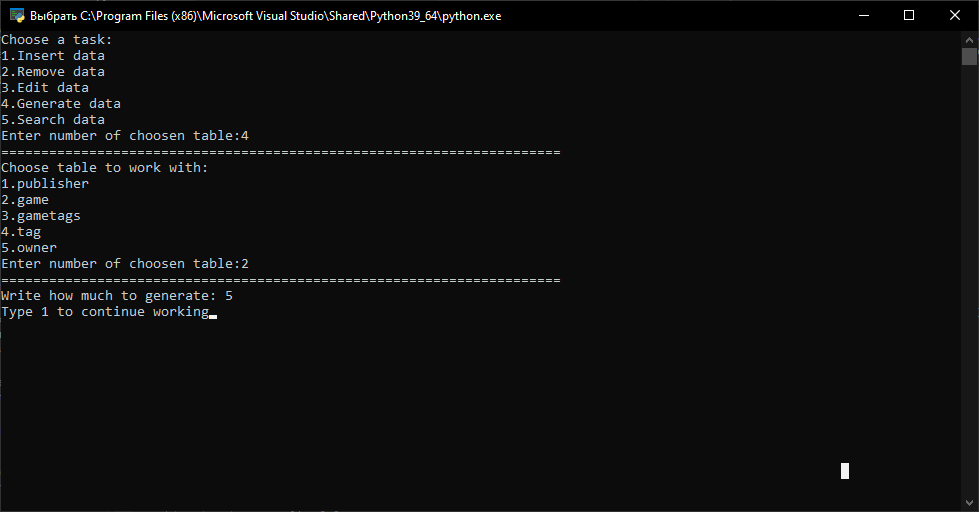


Рисунок 13 – параметри генерації елементів

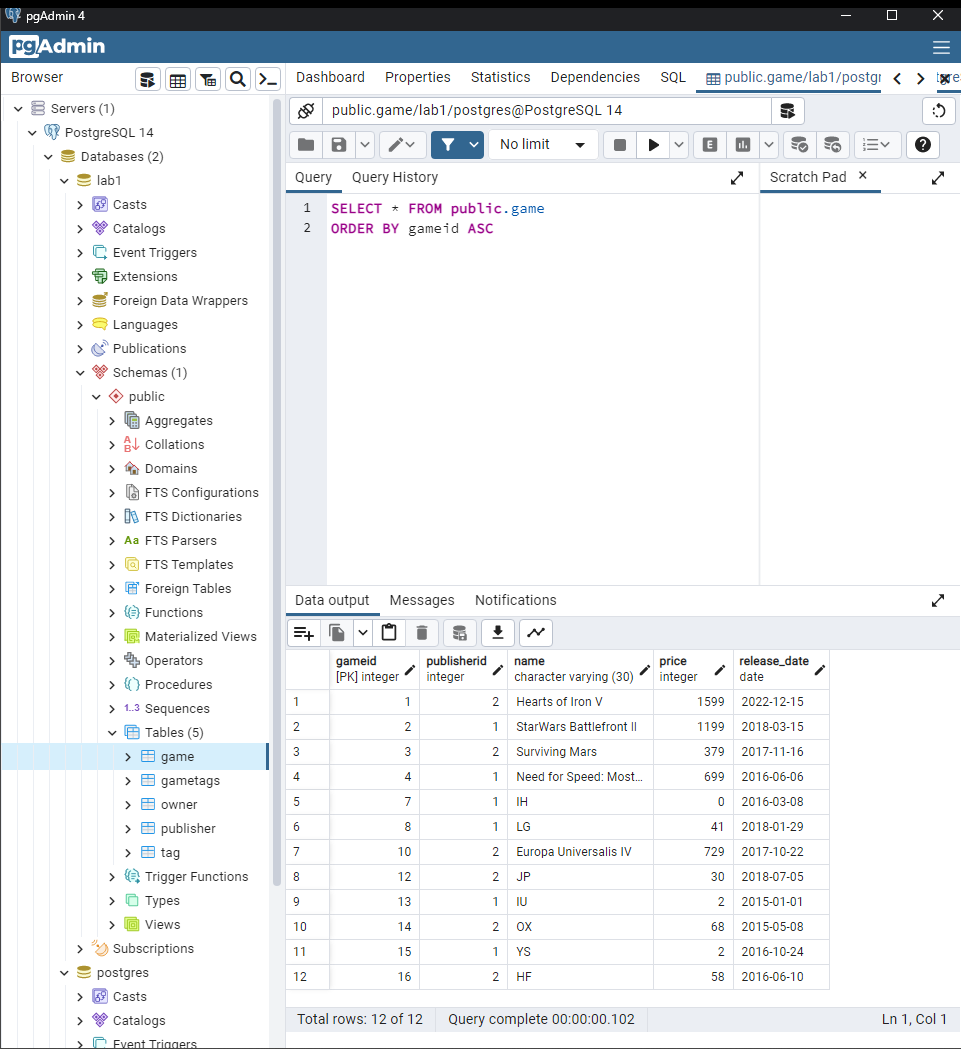


Рисунок 14 – таблиця зі згенерованими елементами

**Скріншот результатів пошуку елементів в таблиці**

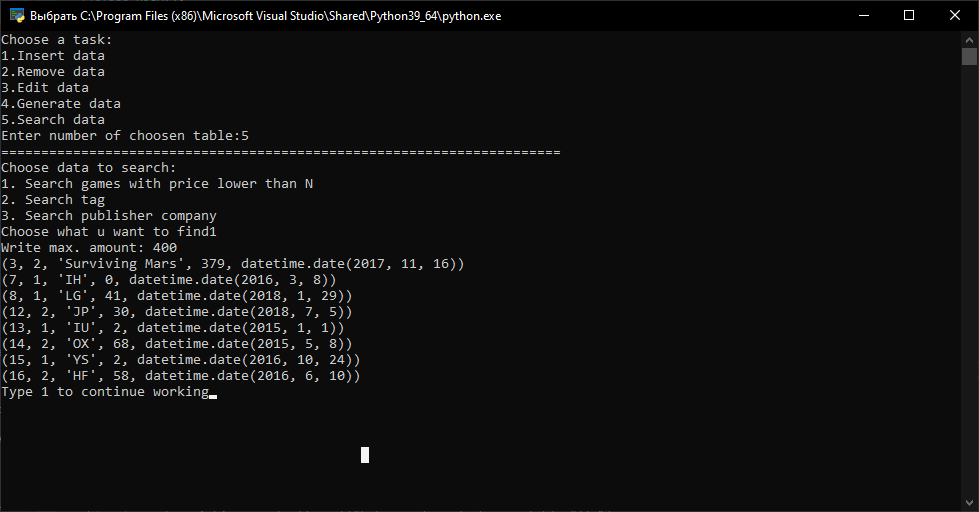


Рисунок 14 – результат пошуку

**Текст програми**

**View.py**

**----------------------------------------------------------------------------------------------------**

from model import Model

class View():

def \_\_init\_\_(self):

self.table\_names=''

def main\_menu(self):

print("Choose a task:")

print("1.Insert data")

print("2.Remove data")

print("3.Edit data")

print("4.Generate data")

print("5.Search data")

task\_number=int(input("Enter number of choosen table:"))

if 0 < task\_number < 6:

return task\_number

else:

return -1

def search\_menu(self):

print("Choose data to search:")

print("1. Search games with price more than N")

print("2. Search tag")

print("3. Search publisher company")

def output\_table\_names(self):

model = Model()

table\_names = model.get\_table\_names()

print("Choose table to work with:")

i=0

while i<len(table\_names):

print(str(i+1)+'.'+ table\_names[i])

i+=1

return table\_names[int(input("Enter number of choosen table:"))-1]

def output\_table\_data(self,values):

if type(values) is list:

for i in values:

print(i)

else:

print(values[0])

for i in range(len(values[1])):

print(values[1][i])

**----------------------------------------------------------------------------------------------------**

**Controler.py**

**----------------------------------------------------------------------------------------------------**

from model import Model

from View import View

from array import \*

import time

import os

class Controler():

def \_\_init\_\_(self):

self.model=Model()

self.view=View()

def insert\_menu(self):

table = self.view.output\_table\_names()

print('='\*70)

table\_data = self.model.get\_table\_data(table)

table\_data=list(table\_data)

tid = str(table+'id')

if tid in table\_data[0]:

table\_data[0].remove(tid)

values = []

for i in range(len(table\_data[0])):

values.append(input(f"Enter value for column {table\_data[0][i]}: "))

self.model.insert\_data(table, values)

if int(input("Type 1 to continue working")) == 1:

print('|\n'\*20)

self.main\_control()

def delete\_menu(self):

table = self.view.output\_table\_names()

print('='\*70)

table\_data = self.model.get\_table\_data(table)

self.view.output\_table\_data(table\_data)

try:

self.model.delete\_data(table, table\_data[1][int(input('Choose number:')) - 1])

except Exception as e:

print(e)

if int(input("Type 1 to continue working")) == 1:

print('|\n'\*20)

self.main\_control()

def change\_menu(self):

table = self.view.output\_table\_names()

print('='\*70)

table\_data = self.model.get\_table\_data(table)

id\_name = table\_data[0][0]

try:

num = table\_data[1][int(input('Choose number:')) - 1][0]

except Exception as e:

print('\*'\*70)

print(e)

return 0

tid = str(table+'id')

if tid in table\_data[0]:

table\_data[0].remove(tid)

values = []

for i in range(len(table\_data[0])):

values.append(input(f"Enter value for column {table\_data[0][i]}: "))

self.model.change\_data(table, values, table\_data[0], num, id\_name)

if int(input("Type 1 to continue working")) == 1:

print('|\n'\*20)

self.main\_control()

def generate\_menu(self):

table = self.view.output\_table\_names()

print('='\*70)

self.model.generate\_data(table, int(input("Write how much to generate: ")))

if int(input("Type 1 to continue working")) == 1:

print('|\n'\*20)

self.main\_control()

def search\_menu(self):

self.view.search\_menu()

n = int(input("Choose what u want to find"))

data = self.model.search\_data(n) if 1 <= n <= 3 else print("Wrong parameter")

if len(data) == 0:

print("Nothing found")

else:

self.view.output\_table\_data(data)

if int(input("Type 1 to continue working")) == 1:

print('|\n'\*20)

self.main\_control()

def main\_control(self):

flag = self.view.main\_menu()

if flag == 1:

print('='\*70)

self.insert\_menu()

elif flag == 2:

print('='\*70)

self.delete\_menu()

elif flag == 3:

print('='\*70)

self.change\_menu()

elif flag == 4:

print('='\*70)

self.generate\_menu()

elif flag == 5:

print('='\*70)

self.search\_menu()

elif flag ==-1:

print('\*'\*70)

print("invalid input")

time.sleep(3)

print('\*'\*70)

self.main\_control()

if \_\_name\_\_ == '\_\_main\_\_':

model = Model()

control = Controler()

control.main\_control()

**----------------------------------------------------------------------------------------------------**

**Model.py**

**----------------------------------------------------------------------------------------------------**

import psycopg2

from psycopg2 import sql

class Model():

def \_\_init\_\_(self):

self.\_\_context = psycopg2.connect(host="localhost",port="5432",database="lab1",user="postgres",password="101002")

self.\_\_cursor = self.\_\_context.cursor()

self.\_\_table\_names = None

return None

def \_\_del\_\_(self):

self.\_\_cursor.close()

self.\_\_context.close()

def clear\_transaction(self):

self.\_\_context.rollback()

def get\_table\_names(self):

if self.\_\_table\_names is None:

self.\_\_cursor.execute("""SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public'""")

self.\_\_table\_names = [table[0] for table in self.\_\_cursor]

return self.\_\_table\_names

def get\_column\_types(self, table\_name):

self.\_\_cursor.execute("""SELECT column\_name, data\_type FROM information\_schema.columns WHERE table\_name = %(TableName)s ORDER BY table\_schema, table\_name""", {'TableName':table\_name})

return self.\_\_cursor.fetchall()

def get\_column\_names(self, table\_name):

self.\_\_cursor.execute("""SELECT column\_name FROM information\_schema.columns WHERE table\_schema = 'public' AND table\_name = %s ORDER BY table\_schema, table\_name""", (table\_name,))

return [x[0] for x in self.\_\_cursor.fetchall()]

def get\_foreign\_key\_info(self, table\_name):

self.\_\_cursor.execute("""

SELECT kcu.column\_name, ccu.table\_name AS foreign\_table\_name,ccu.column\_name AS foreign\_column\_name

FROM information\_schema.table\_constraints AS tc

JOIN information\_schema.key\_column\_usage AS kcu

ON tc.constraint\_name = kcu.constraint\_name

AND tc.table\_schema = kcu.table\_schema

JOIN information\_schema.constraint\_column\_usage AS ccu

ON ccu.constraint\_name = tc.constraint\_name

AND ccu.table\_schema = tc.table\_schema

WHERE tc.constraint\_type = 'FOREIGN KEY' AND tc.table\_name=%s;""", (table\_name,))

return self.\_\_cursor.fetchall()

def get\_table\_data(self, table\_name):

id\_column = self.get\_column\_types(table\_name)[0][0]

cursor = self.\_\_cursor

try:

cursor.execute(sql.SQL('SELECT \* FROM {} ORDER BY {} ASC').format(sql.Identifier(table\_name), sql.SQL(id\_column)))

except Exception as e:

return str(e)

return [col.name for col in cursor.description], cursor.fetchall()

def insert\_data(self, table\_name, value):

types = self.get\_column\_names(table\_name)

types.remove(table\_name+'id')

values={}

for i in range(len(types)):

values.update({types[i]:value[i]})

line = ''

columns = '('

for key in values:

if values[key]:

line += '%(' + key + ')s,'

columns += key + ','

columns = columns[:-1] + ')'

try:

self.\_\_cursor.execute(sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')').format(sql.Identifier(table\_name), sql.SQL(columns)),values)

self.\_\_context.commit()

except Exception as e:

print(e)

def generate\_data(self, table\_name, count):

types = self.get\_column\_types(table\_name)

fk\_array = self.get\_foreign\_key\_info(table\_name)

select\_subquery = ""

insert\_query = "INSERT INTO " + table\_name + " ("

for i in range(1, len(types)):

t = types[i]

name = t[0]

type = t[1]

fk = [x for x in fk\_array if x[0] == name]

if fk:

select\_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT 1)'.format(fk[0][2], fk[0][1]))

elif type == 'integer':

select\_subquery += 'trunc(random()\*100)::INT'

elif type == 'character varying':

select\_subquery += 'chr(trunc(65 + random()\*25)::INT) || chr(trunc(65 + random()\*25)::INT)'

elif type == 'date':

select\_subquery += """ date(timestamp '2014-01-10' + random() \* (timestamp '2020-01-20' - timestamp '2014-01-10'))"""

elif type == 'time without time zone':

select\_subquery += "time '00:00:00' + DATE\_TRUNC('second',RANDOM() \* time '24:00:00')"

else:

continue

insert\_query += name

if i != len(types) - 1:

select\_subquery += ','

insert\_query += ','

else:

insert\_query += ') '

try:

self.\_\_cursor.execute(insert\_query + "SELECT " + select\_subquery + "FROM generate\_series(1," + str(count) + ") AS ser")

self.\_\_context.commit()

except Exception as e:

print(e)

def change\_data(self, table\_name, values, column, cond, id\_name):

string = f"{id\_name} = '{cond}'"

columns = ''

for key in range(len(values)):

if values[key]:

columns += f"{column[key]} = '{values[key]}'" + ','

columns = columns[:-1] + ''

try:

self.\_\_cursor.execute(sql.SQL('UPDATE {} SET {} WHERE {}').format(sql.Identifier(table\_name),sql.SQL(columns), sql.SQL(string)))

self.\_\_context.commit()

except Exception as e:

print(e)

def delete\_data(self, table\_name, cond):

names = self.get\_column\_names(table\_name)

for i in range(len(names)):

condition = str(names[i-1]) + "= '" + str(cond[i-1]) + "'"

try:

self.\_\_cursor.execute(sql.SQL('DELETE FROM {} WHERE {}').format(sql.Identifier(table\_name), sql.SQL(condition)))

self.\_\_context.commit()

except Exception as e:

print(e)

def search\_data(self,flag):

if flag == 1:

string = f"SELECT \* FROM game WHERE price < {int(input('Write max. amount: '))}"

elif flag == 2:

string = f"SELECT \* FROM tag WHERE name = '{input('Write name: ')}'"

elif flag == 3:

string = f"SELECT \* FROM publisher WHERE name = '{input('Write name: ')}'"

try:

self.\_\_cursor.execute(string)

return self.\_\_cursor.fetchall()

except Exception as e:

print(e)