

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №3**

з дисципліни

**«Бази даних і засоби управління»**

Виконав: студент ІII курсу

ФПМ групи КВ-72

Бербега В.О.

Перевірив(ла):

Київ – 2019

**Засоби оптимізації роботи СУБД PostgreSQL**

**Завдання ​ роботи полягає у наступному:**

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Вимоги до інструментарію :**

1. Бібліотека для реалізації ORM - SQLAlchemy для Python або інша з подібною функціональністю.
2. Середовище для відлагодження SQL-запитів до бази даних – pgAdmin 4.
3. СУБД - PostgreSQL 10 або 11.

**Вимоги до оформлення лабораторної роботи у електронному вигляді**

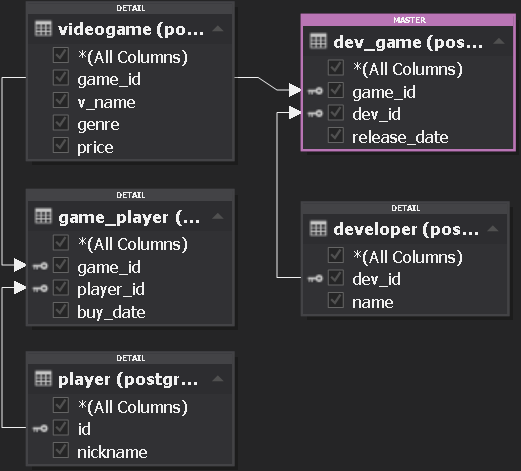
Опис та вміст репозиторію лабораторної роботи у ​репозиторії​ ​GitHub включає назву лабораторної роботи, варіант студента, а також:

* для завдання №1: оновлений програмний код у файлах репозиторію;
* для завдання №2: команди створення індексів, тексти і час виконання запитів SQL;
* для завдання №3: команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних;
* для завдання №4: скріншоти з ходом виконання запитів та їх результатів у обох транзакціях по кожному рівню ізоляції.

**Варіант 1**

|  |  |  |
| --- | --- | --- |
| № варіанта | Види індексів | Умови для тригера |
| 1 | Hash, Btree | before delete, insert |

**Модель бази даних**



**Опис програми**

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 3 модулів:

1. db.py - точка входу до програми.
2. controller.py – містить засоби обробки виключень та повідомлення помилок, викликає функцію головного меню, а також функції, що дозволяють виконувати операції з даними.
3. model.py – містить функції для підключення до бази даних, безпосередньої роботи з БД, а також клас, що представляє базу даних;
4. view.py – містить функції для виводу результатів роботи програми на екран, а також функції для діалогу з користувачем.
5. tables.py – містить класи, що представляють таблиці бази даних та зв’язки між ними.

Код програми

**controller.py**

import os

import re

tables = {'1':'Developer',

'2':'Player',

'3':'Videogame',

'4':'Dev\_Game',

'5':'Game\_Player',

}

columns = {'Developer':['dev\_id', 'name'],

'Player':['id', 'nickname'],

'Videogame':['game\_id', 'v\_name', 'genre', 'price'],

'Dev\_Game':['game\_id', 'dev\_id', 'release\_date'],

'Game\_Player':['game\_id', 'player\_id', 'buy\_date']

}

columnins = {'Developer':['name'],

'Player':['nickname'],

'Videogame':['v\_name', 'genre', 'price'],

'Dev\_Game':['game\_id','dev\_id', 'release\_date'],

'Game\_Player':['game\_id','player\_id', 'buy\_date']

}

fts\_columns = {'Developer':['name'],

'Player':['nickname'],

'Videogame':['v\_name', 'genre'],

}

import model as model

import view as view

DB = model.Database()

def game\_price\_range():

print("Input Range")

print("price more than: ")

lw= input()

print("price less than: ")

hg = input()

where = ("WHERE ((price >= "+lw+")AND(price <= "+hg+")) ORDER BY game\_id")

return(model.select1("videogame", where))

def game\_of\_developers(v\_game):

for temp in v\_game:

print("\n",temp.strip())

try:

yield model.select1("videogame, dev\_game, developer",

"WHERE dev\_game.dev\_id = developer.dev\_id \

AND dev\_game.game\_id = videogame.game\_id \

AND (developer.name ILIKE '{}' \

OR developer.name ILIKE '{}')".format(temp.strip(),temp.strip()), "videogame.game\_id, v\_name, genre, price ")

except (Exception, psycopg2.Error) as error:

print(error)

return

def insert\_into\_table(table\_num):

table = tables[table\_num]

table\_columns = columns[table]

values = []

try:

for i in range(len(table\_columns)):

answer = input(table\_columns[i] + ' = ')

values.append(answer)

except Exception as error:

print(error)

return

print("INSERT INTO " + table + " VALUES(" + ', '.join(values) + ')')

res = DB.insert(table, table\_columns, values)

return res

def delete\_from\_table(table\_num):

table = tables[table\_num]

table\_columns = columns[table]

while True:

print("Choose column to delete by:")

view.table\_columns\_names(table\_num)

chosen\_column\_num = input()

if re.match(r'^[1-{}]{}$'.format(len(table\_columns), "{1}"),

chosen\_column\_num):

chosen\_column = table\_columns[int(chosen\_column\_num)-1]

print("Input value: ")

print("DELETE FROM {} WHERE {} = ...".format(table, chosen\_column))

value = input()

print("DELETE FROM {} WHERE {} = {}".format(table,

chosen\_column, value))

where = [chosen\_column, value]

res = DB.delete(table, where)

return res

elif chosen\_column\_num == '0':

return

else:

print("No such option. Check your input")

def update\_table(table\_num):

table = tables[table\_num]

table\_columns = columns[table]

while True:

print("Choose column to update:")

view.table\_columns\_names(table\_num)

chosen\_column\_num = input()

if re.match(r'^[1-{}]{}$'.format(len(table\_columns), "{1}"),

chosen\_column\_num):

set\_column = table\_columns[int(chosen\_column\_num)-1]

print("Input value: ")

print("UPDATE {} SET {} = ...".format(table, set\_column))

set\_value = input()

print("Choose column to update by:")

view.table\_columns\_names(table\_num)

chosen\_column\_num = input()

if re.match(r'^[1-{}]{}$'.format(len(table\_columns), "{1}"),

chosen\_column\_num):

where\_column = table\_columns[int(chosen\_column\_num)-1]

print("Input value: ")

print("UPDATE {} SET {} = {} WHERE {} = ...".format(table,

set\_column, set\_value, where\_column))

where\_value = input()

print("UPDATE {} SET {} = {} WHERE {} = {}".format(table,

set\_column, set\_value, where\_column, where\_value))

set = [set\_column, set\_value]

where = [where\_column, where\_value]

res = DB.update(table, set, where)

return res

elif chosen\_column\_num == '0':

return

else:

print("No such option. Check your input")

elif chosen\_column\_num == '0':

return

else:

print("No such option. Check your input")

def fts\_table(text, mode, table\_num):

table = tables[table\_num]

to\_tsvector = fts\_columns[table]

where = ' || '.join("to\_tsvector(coalesce({}, ''))".format(w)

for w in to\_tsvector)

where += " @@ plainto\_tsquery('{}')".format(text)

return(model.full\_text\_search(table, where, mode))

def main\_menu():

while True:

view.show\_main\_menu()

option = input()

if re.match(r'^[1-5]{1}$', option):

while True:

view.tables\_names()

chosen\_table = input()

if re.match(r'^[1-6]{1}$', chosen\_table):

table = tables[chosen\_table]

if option == '1':

notes = DB.select(table)

view.print\_orm\_table(notes)

elif option == '2':

res = insert\_into\_table(chosen\_table)

if not res:

print("Data wasn't inserted")

else:

print("Successfully inserted")

elif option == '3':

res = update\_table(chosen\_table)

if not res:

print("Data wasn't updated")

else:

print("Operation successfull")

elif option == '4':

res = delete\_from\_table(chosen\_table)

if not res:

print("Data wasn't deleted")

else:

print("Operation successfull")

elif option == '5':

text = input("Input text to search: ")

view.fts\_mode()

mode = input()

if re.match(r'^[1,2]{1}$', mode):

notes = fts\_table(text, mode, chosen\_table)

view.print\_table(chosen\_table, notes)

elif continue\_or\_back == '0':

break

else:

print("No such option. Check your input")

elif chosen\_table == '0':

break

else:

print("No such option. Check your input")

view.back\_to\_menu()

back\_to\_menu = input()

if back\_to\_menu == '0':

continue

elif back\_to\_menu == '1':

break

else:

print("No such option. Check your input")

elif option == '6':

view.print\_table('3', game\_price\_range())

elif option == '7':

dev = input("Input developers names separated with comma: ")

dev = dev.split(', ')

for dev\_list in game\_of\_developers(dev):

view.print\_table('3', dev\_list)

elif option == '8':

num\_of\_rand = input("Number of random players\n")

res = model.random\_author(num\_of\_rand)

if not res:

print("Data wasn't updated")

else:

print("Successfully updated")

elif option == '9':

os.system("cls")

elif option == '0':

exit()

else:

print("No such option. Check your input")

if \_\_name\_\_ == "\_\_main\_\_":

main\_menu()

**view.py**

import os

from controller import tables, columns

def show\_main\_menu():

print('''\n+=+=+=+=+=+=+=+Main menu:=+=+=+=+=+=+=+

1. Select

2. Insert

3. Update

4. Delete

5. Full text search

6. Games in price range

7. Game of developer

8. Random player

9. Clear screen

0. Exit\t\n+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+''')

def fts\_mode():

print('''\n+=+=+=+=+=+=+Search mode:=+=+=+=+=+=+=+

1. Required word entry

2. Full phrase

0. Back to menu\n+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+''')

def back\_to\_menu():

print('''\n+=+=+=+=+=Back to main menu:+=+=+=+=+=+

1. Yes

0. No\n+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+''')

def table\_columns\_names(table):

table\_columns = columns[tables[table]]

print('+='\*20)

for i in range(len(table\_columns)):

print(str(i+1) + ". " + table\_columns[i])

print("0. Back to menu")

print('+='\*20)

def tables\_names():

print('+='\*20)

print("Choose table:")

for k, v in tables.items():

print("{}. {}".format(k, v))

print("0. Back to menu")

print('+='\*20)

def print\_table(table, notes):

if not notes:

print('+='\*20)

print('No data')

print('+='\*20)

return

print()

table\_columns = columns[tables[table]]

print('+='\*20)

for note in notes:

for i in range(len(note)):

print(table\_columns[i] + ' -', note[i])

print('+='\*20)

def print\_orm\_table(notes):

if not notes:

print('\nNo data')

return

for note in notes:

print(note)

print('-'\*40)

**model.py**

import psycopg2

import random, string

import datetime

import sqlalchemy as db

from sqlalchemy.orm import sessionmaker

from sqlalchemy.exc import SQLAlchemyError

from sqlalchemy.ext.declarative import declarative\_base

import tables

class Database():

engine = None

Base = tables.Base

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

try:

url = "postgresql://postgres:password@localhost:5432/postgres"

self.engine = db.create\_engine(url)

self.connection = self.engine.connect()

#self.table\_names = self.Base.metadata.tables.keys()

except (Exception, psycopg2.Error, SQLAlchemyError) as error:

print(error)

exit()

def connect\_db(self):

try:

Session = sessionmaker(bind=self.engine)

session = Session()

res = session

except (Exception, psycopg2.Error) as error:

res = False

print(error)

return res

def select(self, table):

session = self.connect\_db()

if not session: return False

my\_table = getattr(tables, table)

try:

res = session.query(my\_table).all()

except (Exception, psycopg2.Error, SQLAlchemyError) as error:

res = False

print(error)

session.commit()

return res

def insert(self, table, table\_columns, values):

session = self.connect\_db()

if not session: return False

my\_table = getattr(tables, table)

new\_row = my\_table(\*values)

try:

session.add(new\_row)

res = True

except (Exception, psycopg2.Error, SQLAlchemyError) as error:

res = False

print(error)

session.commit()

return res

def delete(self, table, where = ""):

session = self.connect\_db()

if not session: return False

my\_table = getattr(tables, table)

filter = {f"{where[0]}" : f"{where[1]}"}

try:

session.query(my\_table).filter\_by(\*\*filter).delete()

res = True

except (Exception, psycopg2.Error, SQLAlchemyError) as error:

res = False

print(error)

session.commit()

return res

def update(self, table, set = "", where = ""):

session = self.connect\_db()

if not session: return False

my\_table = getattr(tables, table)

filter = {f"{where[0]}" : f"{where[1]}"}

values = {f"{set[0]}" : f"{set[1]}"}

print(filter)

print(values)

try:

session.query(my\_table).filter\_by(\*\*filter).update(values)

res = True

except (Exception, psycopg2.Error, SQLAlchemyError) as error:

res = False

print(error)

session.commit()

return res

def connect\_db():

try:

res = psycopg2.connect(host="localhost", port="5432",

database="postgres", user="postgres",

password='password')

except (Exception, psycopg2.Error) as error:

res = False

print(error)

return res

def select(table, id = "", fields = "\*", where = ""):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("SELECT " + fields + " FROM " + table + ' ' + where + "ORDER BY " + id +" ASC")

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def select1(table, where = "", fields = "\*"):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("SELECT " + fields + " FROM " + table + ' ' + where)

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def random\_author(num):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

cursor.execute("insert into player (nickname) select \* FROM rand\_player({})".format(num))

res = True

except (Exception, psycopg2.Error) as error:

res = False

print(error)

conn.commit()

cursor.close()

conn.close()

return res

def full\_text\_search(table, where, mode):

conn = connect\_db()

if not conn: return False

cursor = conn.cursor()

try:

if mode == '1':

cursor.execute("select \* from {} where {}".format(table, where))

res = cursor.fetchall()

elif mode == '2':

cursor.execute("select \* from {} where {}".format(table, where))

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = None

print(error)

conn.commit()

cursor.close()

conn.close()

return res

**tables.py**

import sqlalchemy as db

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

class Developer(Base):

\_\_tablename\_\_ = 'developer'

dev\_id = db.Column(db.Integer, db.Sequence('developer\_dev\_id\_seq'), primary\_key=True)

name = db.Column(db.String, primary\_key=True)

game\_of\_dev = db.orm.relationship("Videogame", secondary="dev\_game")

def \_\_init\_\_(self, dev\_id, name):

self.dev\_id = dev\_id

self.name = name

def \_\_repr\_\_(self):

return (f'dev\_id - {self.dev\_id}\n'

f'name - {self.name}')

class Videogame(Base):

\_\_tablename\_\_ = 'videogame'

game\_id = db.Column(db.Integer, db.Sequence('videogame\_game\_id\_seq'), primary\_key=True)

v\_name = db.Column(db.String)

genre = db.Column(db.String)

price = db.Column(db.Numeric)

dev\_of\_game = db.orm.relationship("Videogame", secondary="dev\_game")

player\_of\_game = db.orm.relationship("Player", secondary="game\_player")

def \_\_init\_\_(self, game\_id, v\_name, genre, price):

self.game\_id = game\_id

self.v\_name = v\_name

self.genre = genre

self.price = price

def \_\_repr\_\_(self):

return (f'game\_id - {self.game\_id}\n'

f'v\_name - {self.v\_name}\n'

f'genre - {self.genre}\n'

f'price - {self.price}')

class Player(Base):

\_\_tablename\_\_ = 'player'

id = db.Column(db.Integer, db.Sequence('player\_id\_seq'), primary\_key=True)

nickname = db.Column(db.String)

game\_of\_player = db.orm.relationship("Videogame", secondary="game\_player")

def \_\_init\_\_(self, id, nickname):

self.id = id

self.nickname = nickname

def \_\_repr\_\_(self):

return (f'id - {self.id}\n'

f'nickname - {self.nickname}')

class Dev\_Game(Base):

\_\_tablename\_\_ = 'dev\_game'

dev\_id = db.Column(db.String, db.ForeignKey('developer.dev\_id'))

game\_id = db.Column(db.String, db.ForeignKey('videogame.game\_id'))

release\_date = db.Column(db.DateTime)

dev\_of\_game = db.orm.relationship(Videogame, backref=db.orm.backref('dev\_game', cascade="all, delete-orphan"))

game\_of\_dev = db.orm.relationship(Developer, backref=db.orm.backref('dev\_game', cascade="all, delete-orphan"))

\_\_table\_args\_\_ = (db.PrimaryKeyConstraint('dev\_id', 'game\_id', name='dev\_game\_pkey'),)

def \_\_init\_\_(self, dev\_id, game\_id, release\_date):

self.dev\_id = dev\_id

self.game\_id = game\_id

self.release\_date = release\_date

def \_\_repr\_\_(self):

return (f'dev\_id - {self.dev\_id}\n'

f'game\_id - {self.game\_id}\n'

f'release\_date - {self.release\_date}')

class Game\_Player(Base):

\_\_tablename\_\_ = 'game\_player'

game\_id = db.Column(db.String, db.ForeignKey('videogame.game\_id'))

player\_id = db.Column(db.String, db.ForeignKey('player.id'))

buy\_date = db.Column(db.DateTime)

game\_of\_player = db.orm.relationship(Player, backref=db.orm.backref('game\_player', cascade="all, delete-orphan"))

player\_of\_game = db.orm.relationship(Videogame, backref=db.orm.backref('game\_player', cascade="all, delete-orphan"))

\_\_table\_args\_\_ = (db.PrimaryKeyConstraint('game\_id', 'player\_id', name='game\_player\_pkey'),)

def \_\_init\_\_(self, game\_id, player\_id, buy\_date):

self.game\_id = game\_id

self.player\_id = player\_id

self.buy\_date = buy\_date

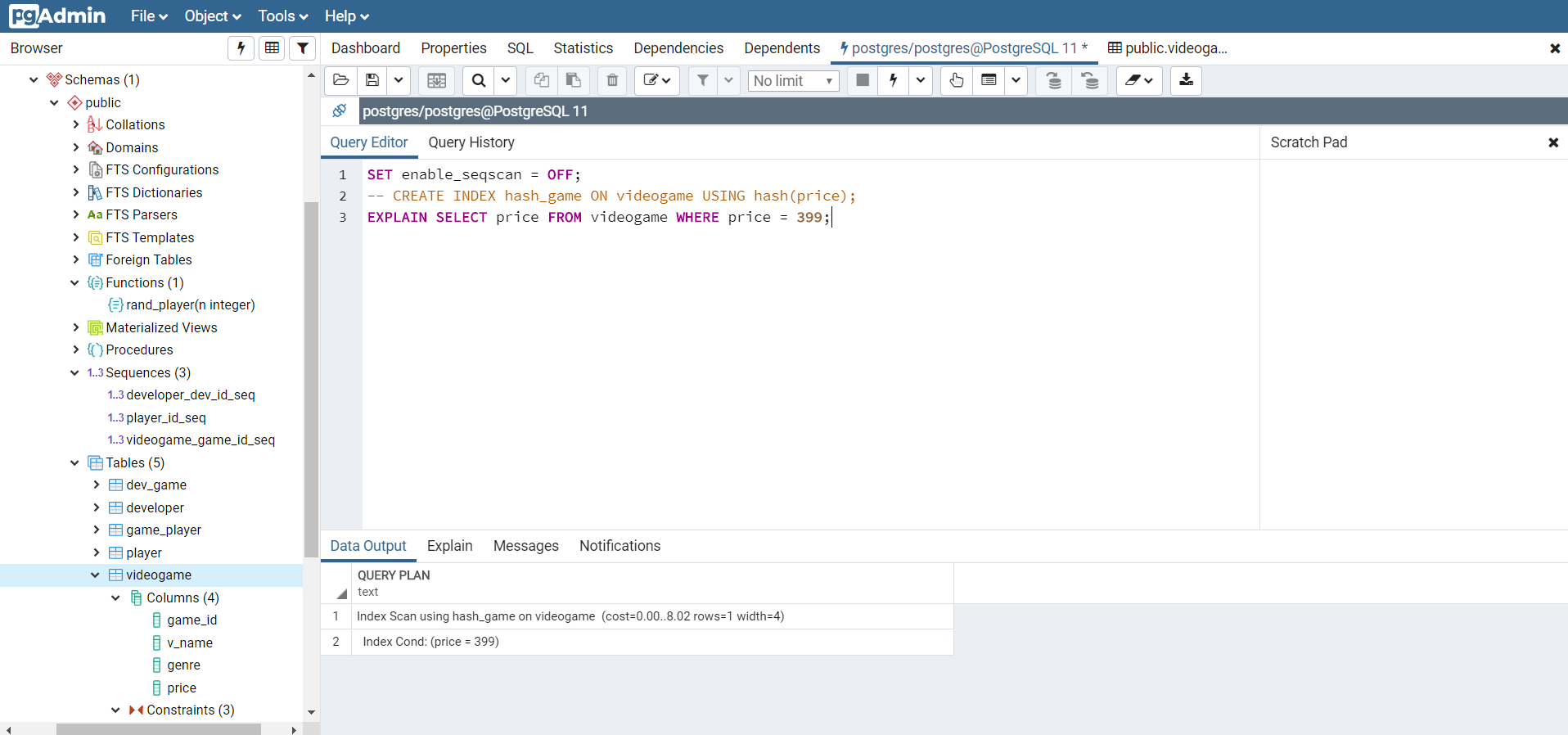
def \_\_repr\_\_(self):

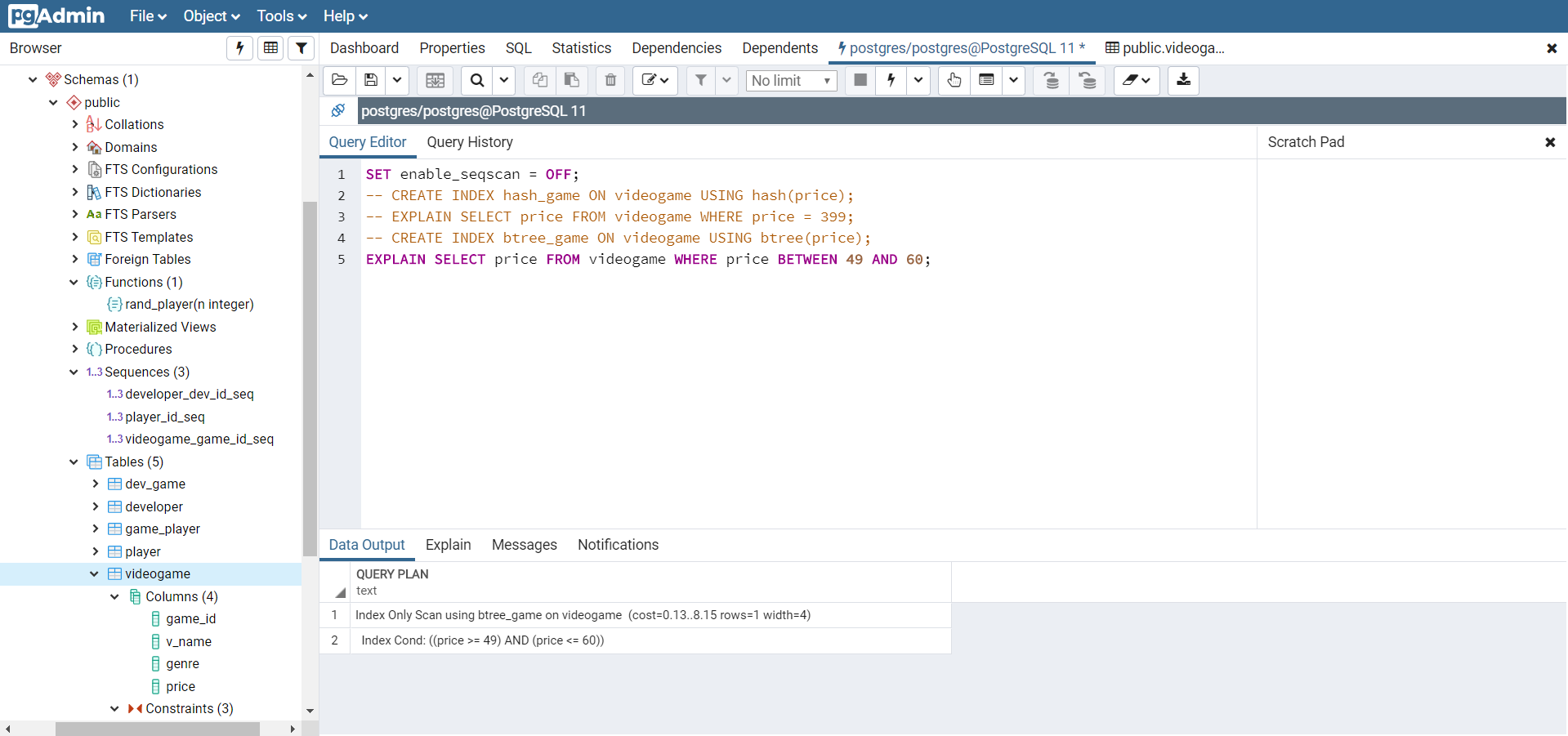
return (f'game\_id - {self.game\_id}\n'

f'player\_id - {self.player\_id}\n'

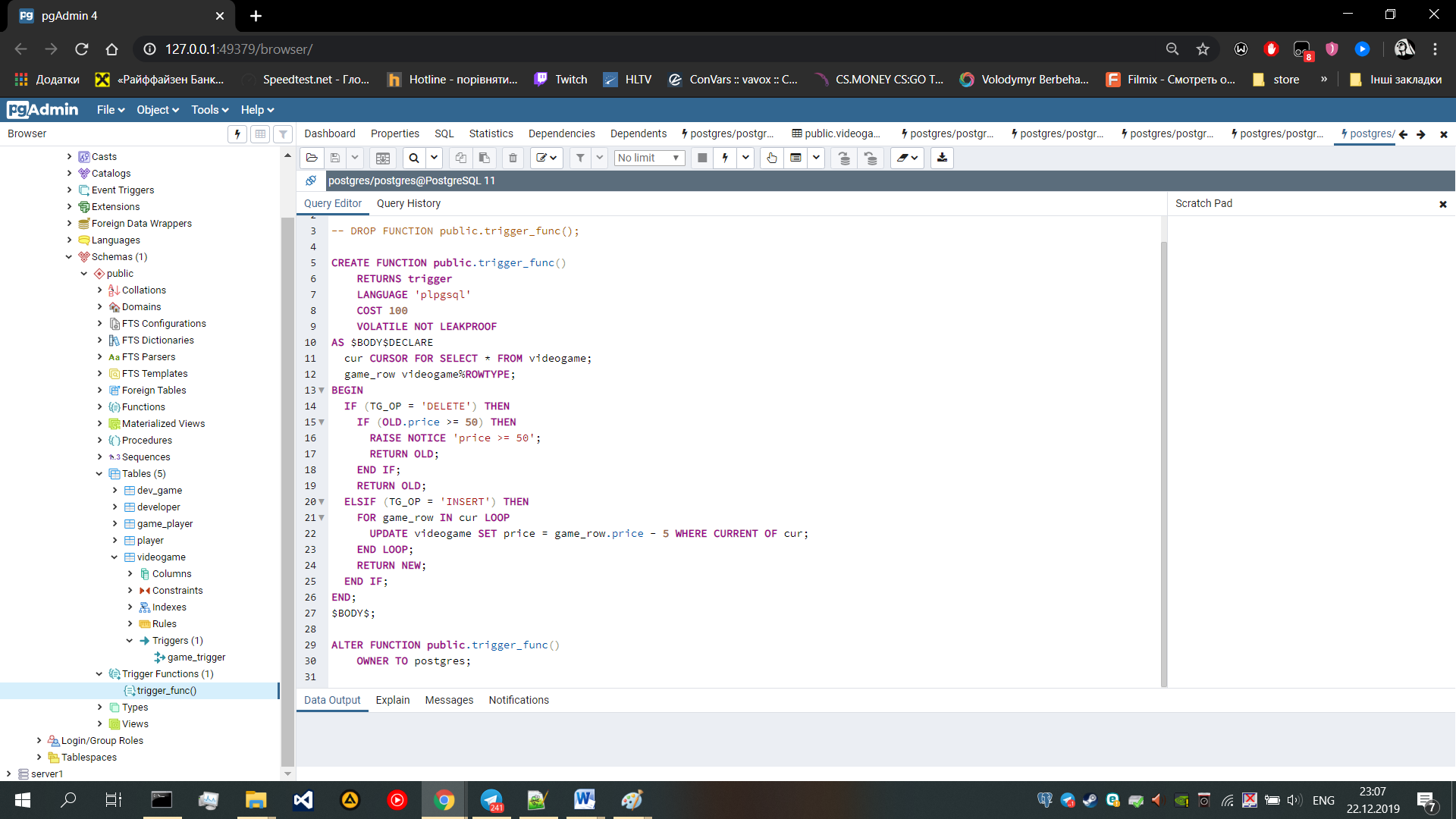
f'buy\_date - {self.buy\_date}')

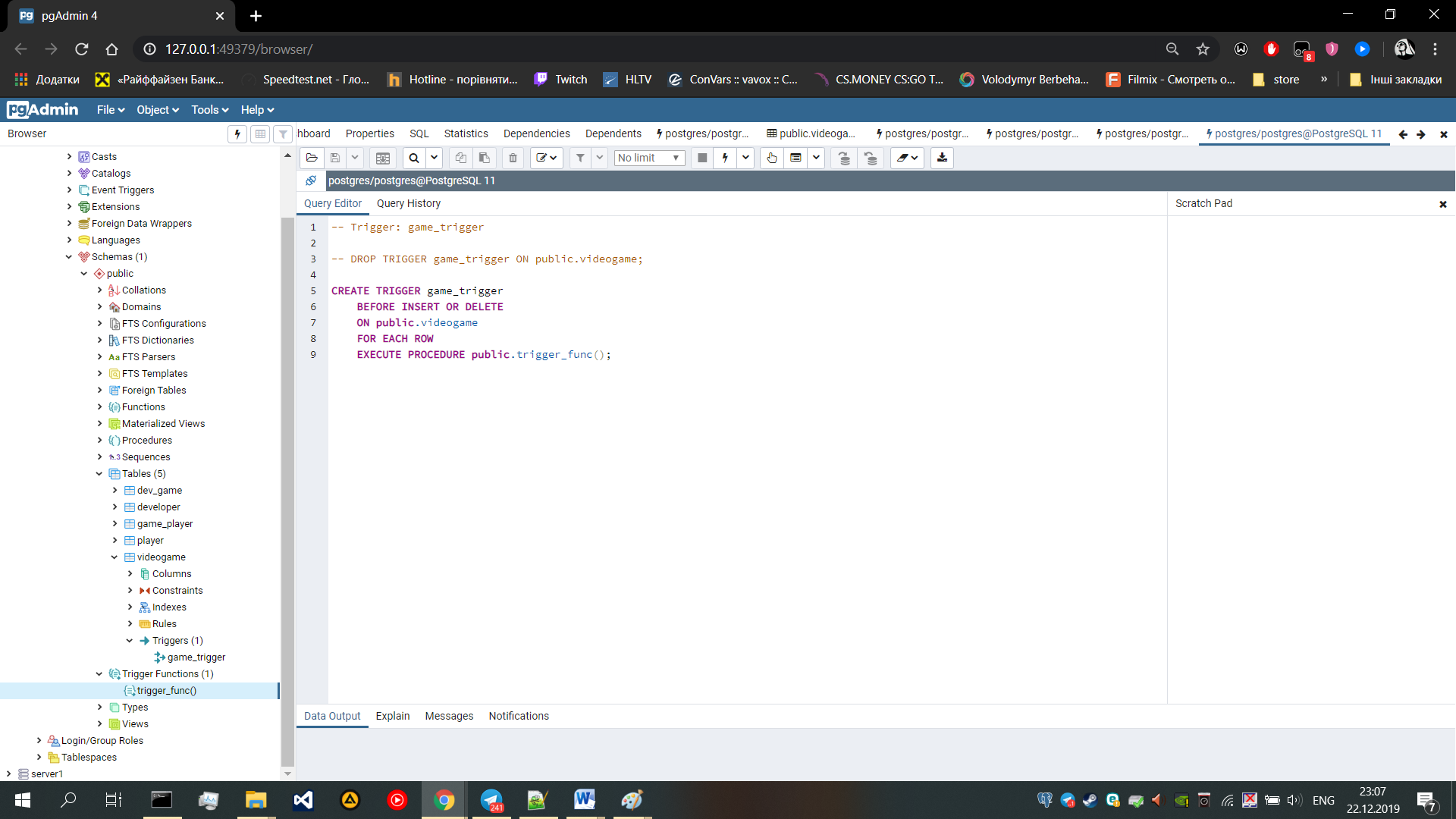
**Hash index screenshot**

****

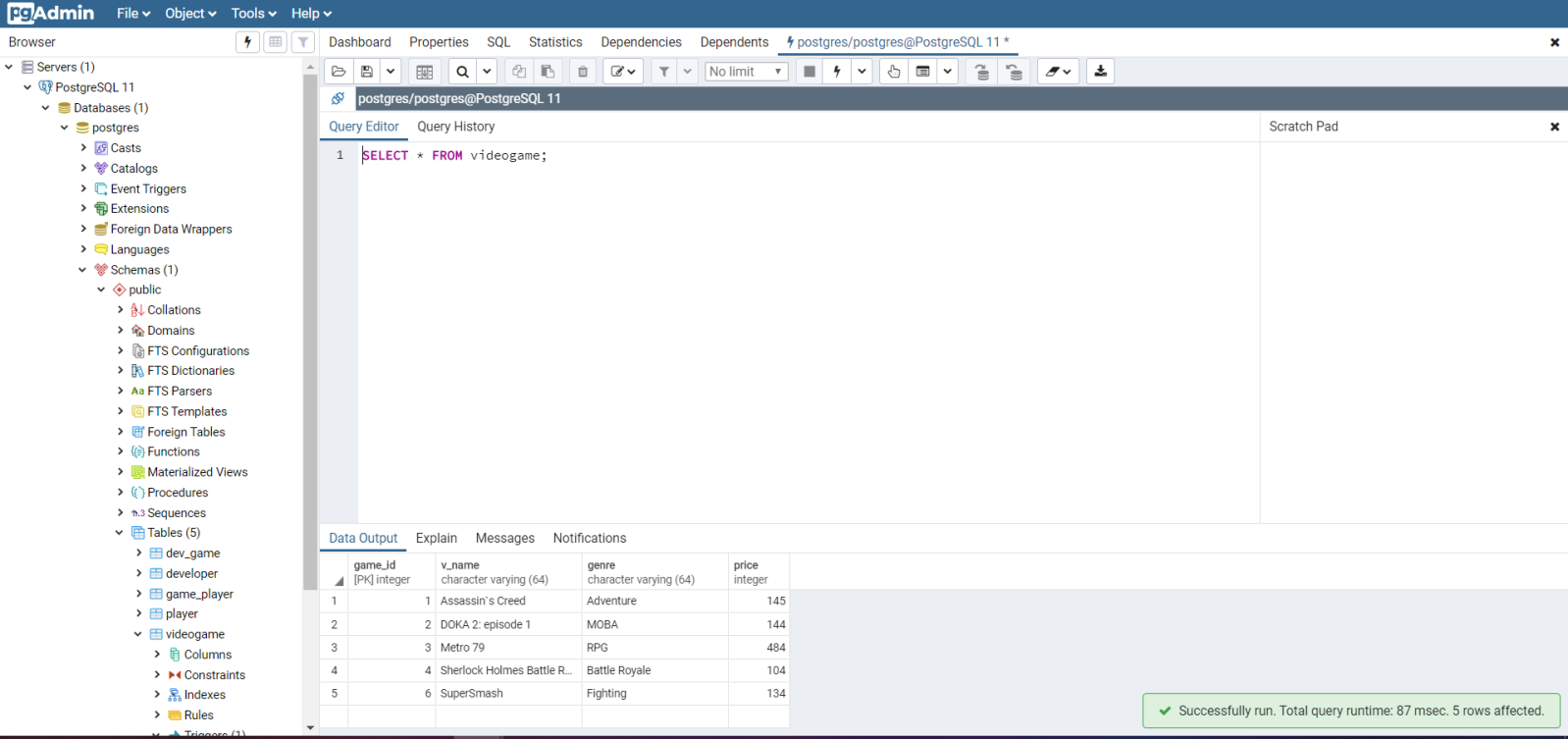
**Btree index screenshot**

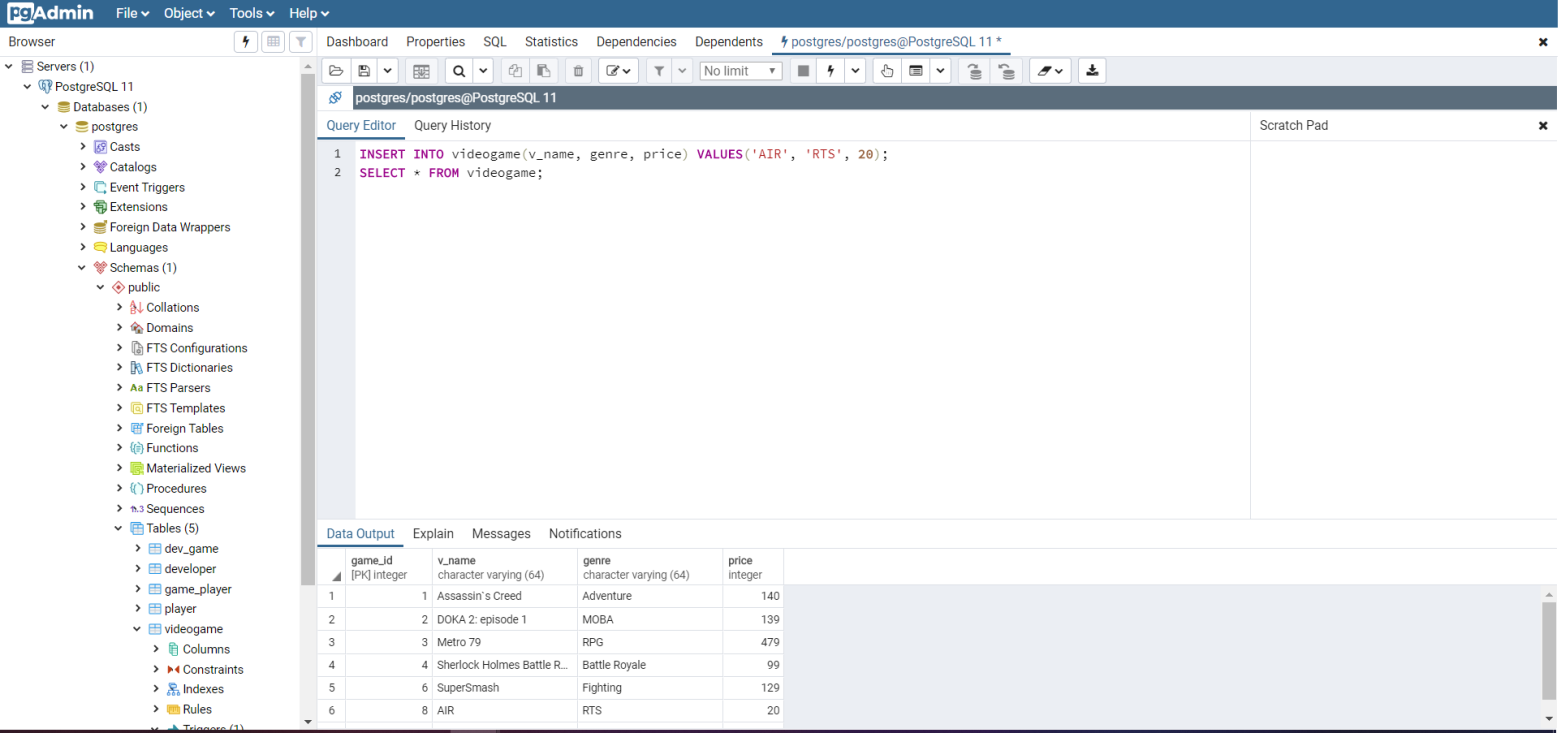
**Тригер**

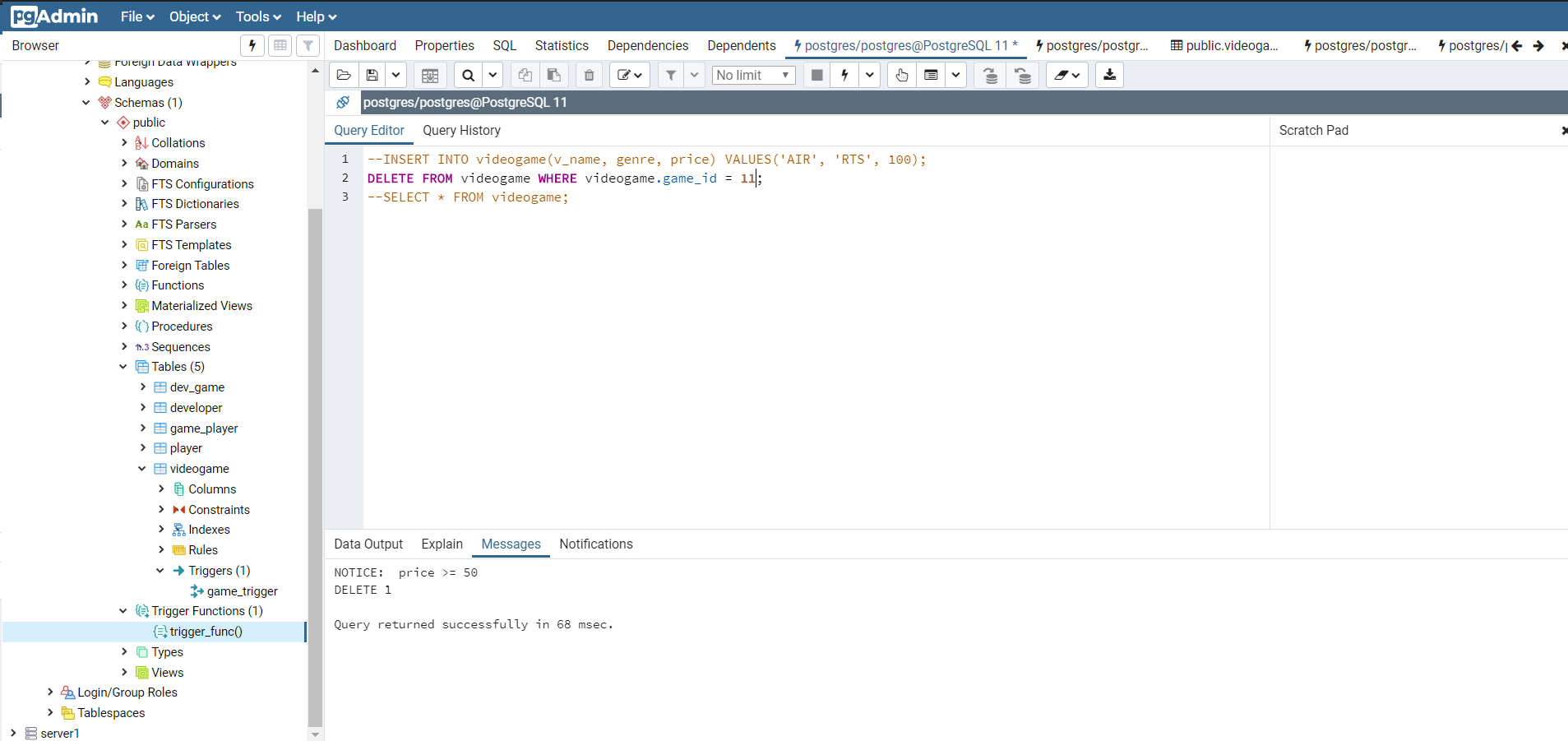




Скріншот таблиці даних videogame до INSERT (віднімає 5 від ціни гри)



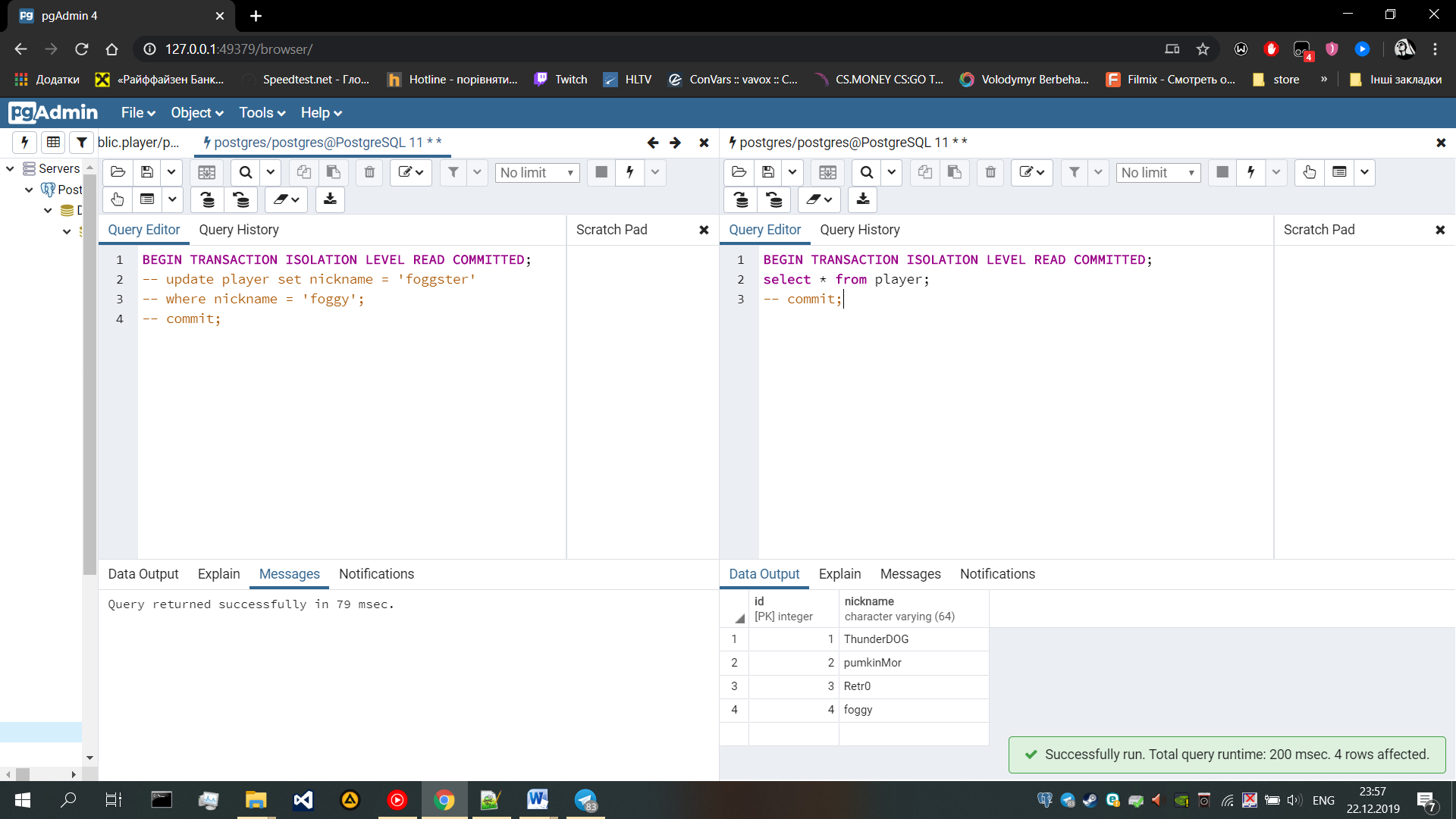
 INSERT (віднімає 5 від ціни гри)

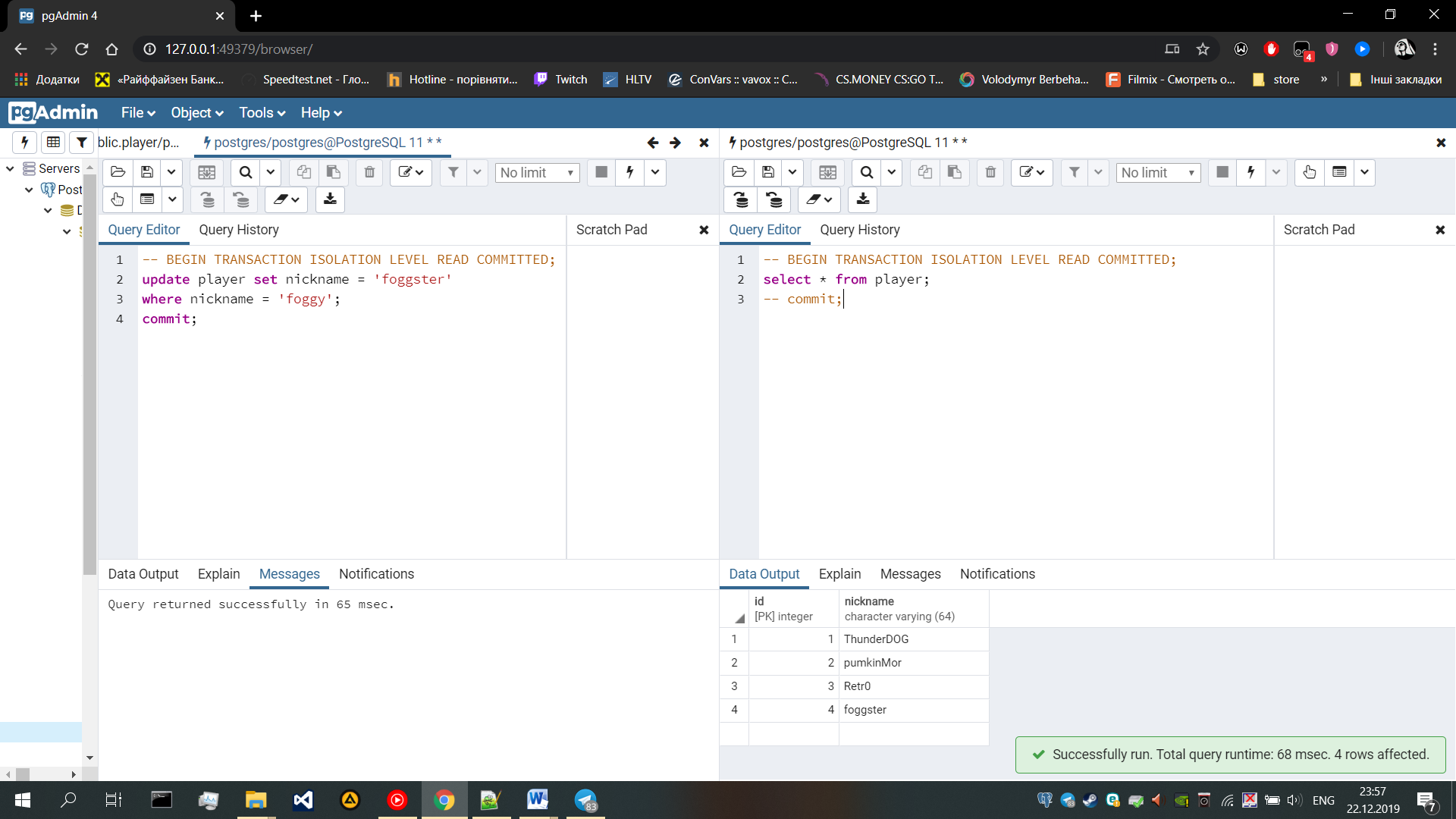
 DELETE

**Дослідження рівнів ізоляції**

**READ COMMITTED**

**Non-repeatable read**





**REPEATABLE READ**

