

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

Туркін М. П.

Перевірив:\_\_\_\_\_\_\_\_\_\_\_\_\_

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**

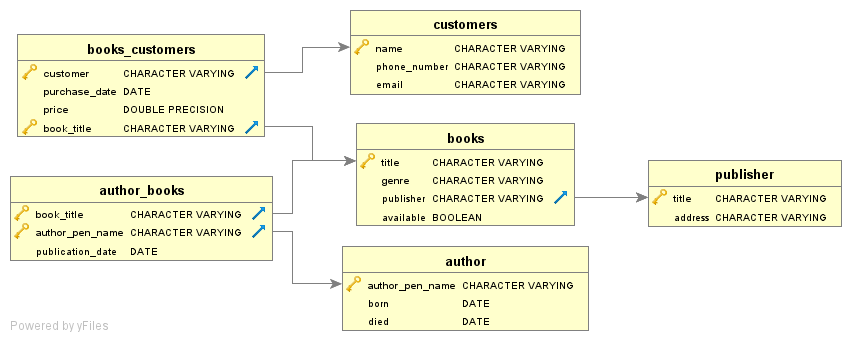
Опис та вміст репозиторію лабораторної роботи у ​репозиторії​ ​GitHub включає назву лабораторної роботи, варіант студента, а також:

* 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 17**

|  |  |  |
| --- | --- | --- |
| № варіанта | Види індексів | Умови для тригера |
| 17 | GIN, BRIN | before update, delete |

**Модель даних БД «Online book store»**

****

**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. Main.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.

**Посилання для навігації по тексту програми**

1. [main.py](#main)
2. Controller.py
   1. Функція [статичного](#available_books) пошуку всіх наявних книг
   2. Функція для [динамічного](#book_of_author) пошуку книг за автором
   3. Функція, що [дозволяє](#insert_into_table) сформувати дані для операції insert
   4. Функція, що [дозволяє](#delete_from_table) сформувати дані для операції delete
   5. Функція, що [дозволяє](#update_table) сформувати дані для операції update
   6. Функція для [повнотекстового](#fts_table) пошуку у певній таблиці бази даних
   7. Функція [головного](#main_menu) меню
3. [Model.py](#Model)
   1. [Клас бази даних](#class_Database)
      1. [Функція для підключення до бази даних](#connect_db)
      2. [Функція, що дозволяє виконати операцію select](#select)
      3. [Функція, що дозволяє виконати операцію delete](#delete)
      4. [Функція, що дозволяє виконати операцію insert](#insert)
      5. [Функція, що дозволяє виконати операцію update](#update)
   2. Функція connect для запитів пошуку
   3. Функція select для запитів пошуку
   4. [Функція для заповнення таблиці author рандомними даними](#random)
   5. [Функція, що дозволяє виконати операцію повнотекстового пошуку](#full_text_search)
4. [View.py](#View)
5. [Tables.py](#Tables)

**Лістинг модуля main.py**

import lab3\_controller as controller

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

controller.main\_menu()

**Лістинг модуля Controller.py**

import os

import re

tables = {'1':'author',

'2':'books',

'3':'customers',

'4':'publisher',

'5':'author\_books',

'6':'books\_customers'

}

columns = {'author':['author\_pen\_name', 'born', 'died'],

'books':['title', 'genre', 'publisher', 'available'],

'customers':['name', 'phone\_number', 'email'],

'publisher':['title', 'address'],

'author\_books':['book\_title', 'author\_pen\_name', 'publication\_date'],

'books\_customers':['customer', 'purchase\_date', 'price', 'book\_title']

}

fts\_columns = {'author':['author\_pen\_name'],

'books':['title', 'genre', 'publisher'],

'customers':['name', 'email'],

'publisher':['title', 'address'],

'author\_books':['book\_title', 'author\_pen\_name'],

'books\_customers':['customer', 'book\_title']

}

import lab3\_model as model

import lab3\_view as view

def available\_books():

return(model.select("books, author, author\_books", "author.author\_pen\_name, books.title",

"WHERE books.available = 'true' AND author\_books.book\_title = books.title \

AND author\_books.author\_pen\_name = author.author\_pen\_name"))

def book\_of\_author(author\_pen\_name, bool):

for name in author\_pen\_name:

try:

yield model.select("books, author, author\_books", "author.author\_pen\_name, books.title",

"WHERE (books.available = '1' OR books.available = '{}') \

AND author\_books.book\_title = books.title \

AND author\_books.author\_pen\_name = author.author\_pen\_name \

AND (author.author\_pen\_name ILIKE '{}' \

OR author.author\_pen\_name ILIKE '{}')".format(bool, name.strip() + " %", "% " + name.strip()))

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) + ')')

view.continue\_or\_back()

continue\_or\_back = input()

if continue\_or\_back == '1':

res = model.insert(table, values = '(' + ', '.join(values) + ')')

return res

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

return

else:

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

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))

view.continue\_or\_back()

continue\_or\_back = input()

if continue\_or\_back == '1':

where = " WHERE {} = {}".format(chosen\_column, value)

res = model.delete(table, where)

return res

elif continue\_or\_back == '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 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))

view.continue\_or\_back()

continue\_or\_back = input()

if continue\_or\_back == '1':

set = " SET {} = {}".format(set\_column, set\_value)

where = " WHERE {} = {}".format(where\_column, where\_value)

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

return res

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

return

else:

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

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 = view.tables[chosen\_table]

if option == '1':

notes = model.select(table)

view.print\_table(chosen\_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 = delete\_from\_table(chosen\_table)

if not res:

print("Data wasn't deleted")

else:

print("Operation successfull")

elif option == '4':

res = update\_table(chosen\_table)

if not res:

print("Data wasn't updated")

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.continue\_or\_back()

continue\_or\_back = input()

if continue\_or\_back == '1':

continue

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

break

else:

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

elif option == '6':

view.print\_table('2', available\_books())

elif option == '7':

author = input("Input authors names separated with comma: ")

author = author.split(',')

bool = input("Output only available books? (y/n): ")

for book\_list in book\_of\_author(author, bool):

view.print\_table('2', book\_list)

elif option == '8':

num\_of\_rand = input("How many random authors insert?\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()

**Лістинг модуля Model.py**

import psycopg2

import sqlalchemy as db

from sqlalchemy.orm import sessionmaker

from sqlalchemy.exc import SQLAlchemyError

from sqlalchemy.ext.declarative import declarative\_base

import tables

data\_types = {'Author':['str', 'date', 'date'],

'Books':['str', 'str', 'str', 'bool'],

'Customers':['str', 'str', 'str'],

'Publisher':['str', 'str'],

'Author\_Books':['str', 'str', 'date'],

'Books\_Customers':['str', 'date', 'money', 'str']

}

class Database():

engine = None

Base = tables.Base

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

try:

url = "postgresql://postgres:qwerty@localhost:5432/Online\_book\_store"

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)

dtype = data\_types[table]

for i in range(len(dtype)):

if dtype[i] == 'bool':

if values[i] in ['true', 'True', 'y', 'yes', '1']:

values[i] = True

else:

values[i] = False

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]}"}

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="Online\_book\_store", user="postgres",

password='qwerty')

except (Exception, psycopg2.Error) as error:

res = False

print(error)

return res

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

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 author (author\_pen\_name, born, died) select \* FROM rand\_author({})".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 not ({})".format(table, where))

res = cursor.fetchall()

except (Exception, psycopg2.Error) as error:

res = None

print(error)

conn.commit()

cursor.close()

conn.close()

return res

**Лістинг модуля View.py**

import os

from lab3\_controller import tables, columns

def show\_main\_menu():

print('''\nChoose what to do:

1. Select

2. Insert

3. Delete

4. Update

5. Full text search

6. Available books

7. All books of author

8. Random author

9. Clear screen

0. Exit''')

def fts\_mode():

print('''\nChose full text search mode:

1. Required word entry

2. Word is not included

0. Go back''')

def continue\_or\_back():

print('''\nDo you want to continue or go back:

1. Continue

0. Go back''')

def print\_or\_back():

print('''\nDo you want to output next results or go back:

1. Output next

0. Go back''')

def table\_columns\_names(table):

table\_columns = columns[tables[table]]

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

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

print("0. Go back")

def tables\_names():

print("\nChoose table:")

for k, v in tables.items():

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

print("0. Go back")

def print\_table(table, notes):

if not notes:

print('\nNo data')

return

print()

table\_columns = columns[tables[table]]

for note in notes:

for i in range(len(note)):

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

print('-'\*40)

def print\_orm\_table(notes):

if not notes:

print('\nNo data')

return

for note in notes:

print(note)

print('-'\*40)

**Лістинг модуля Tables.py**

import sqlalchemy as db

from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

class Author(Base):

\_\_tablename\_\_ = 'author'

author\_pen\_name = db.Column(db.String, primary\_key=True)

born = db.Column(db.DateTime)

died = db.Column(db.DateTime)

books\_of\_author = db.orm.relationship("Books", secondary="author\_books")

def \_\_init\_\_(self, author\_pen\_name, born, died):

self.author\_pen\_name = author\_pen\_name

self.born = born

self.died = died

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

return (f'author\_pen\_name - {self.author\_pen\_name}\n'

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

f'died - {self.died}')

class Books(Base):

\_\_tablename\_\_ = 'books'

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

genre = db.Column(db.String)

publisher = db.Column(db.String, db.ForeignKey('publisher.title'))

available = db.Column(db.Boolean)

author\_of\_book = db.orm.relationship("Author", secondary="author\_books")

customer\_of\_book = db.orm.relationship("Customers", secondary="books\_customers")

def \_\_init\_\_(self, title, genre, publisher, available):

self.title = title

self.genre = genre

self.publisher = publisher

self.available = available

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

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

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

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

f'available - {self.available}')

class Customers(Base):

\_\_tablename\_\_ = 'customers'

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

phone\_number = db.Column(db.String)

email = db.Column(db.String)

book\_of\_customer = db.orm.relationship("Books", secondary="books\_customers")

def \_\_init\_\_(self, name, phone\_number, email):

self.name = name

self.phone\_number = phone\_number

self.email = email

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

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

f'phone\_number - {self.phone\_number}\n'

f'email - {self.email}')

class Publisher(Base):

\_\_tablename\_\_ = 'publisher'

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

address = db.Column(db.String)

def \_\_init\_\_(self, title, address):

self.title = title

self.address = address

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

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

f'address - {self.address}')

class Author\_Books(Base):

\_\_tablename\_\_ = 'author\_books'

author\_pen\_name = db.Column(db.String, db.ForeignKey('author.author\_pen\_name'))

book\_title = db.Column(db.String, db.ForeignKey('books.title'))

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

author\_of\_book = db.orm.relationship(Author, backref=db.orm.backref('author\_books', cascade="all, delete-orphan"))

books\_of\_author = db.orm.relationship(Books, backref=db.orm.backref('author\_books', cascade="all, delete-orphan"))

\_\_table\_args\_\_ = (db.PrimaryKeyConstraint('author\_pen\_name', 'book\_title', name='author\_books\_pkey'),)

def \_\_init\_\_(self, author\_pen\_name, book\_title, publication\_date):

self.author\_pen\_name = author\_pen\_name

self.book\_title = book\_title

self.publication\_date = publication\_date

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

return (f'author\_pen\_name - {self.author\_pen\_name}\n'

f'book\_title - {self.book\_title}\n'

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

class Books\_Customers(Base):

\_\_tablename\_\_ = 'books\_customers'

customer = db.Column(db.String, db.ForeignKey('customers.name'))

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

price = db.Column(db.Numeric)

book\_title = db.Column(db.String, db.ForeignKey('books.title'))

customer\_of\_book = db.orm.relationship(Customers, backref=db.orm.backref('books\_customers', cascade="all, delete-orphan"))

book\_of\_customer = db.orm.relationship(Books, backref=db.orm.backref('books\_customers', cascade="all, delete-orphan"))

\_\_table\_args\_\_ = (db.PrimaryKeyConstraint('customer', 'book\_title', name='books\_customers\_pkey'),)

def \_\_init\_\_(self, customer, purchase\_date, price, book\_title):

self.customer = customer

self.purchase\_date = purchase\_date

self.price = price

self.book\_title = book\_title

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

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

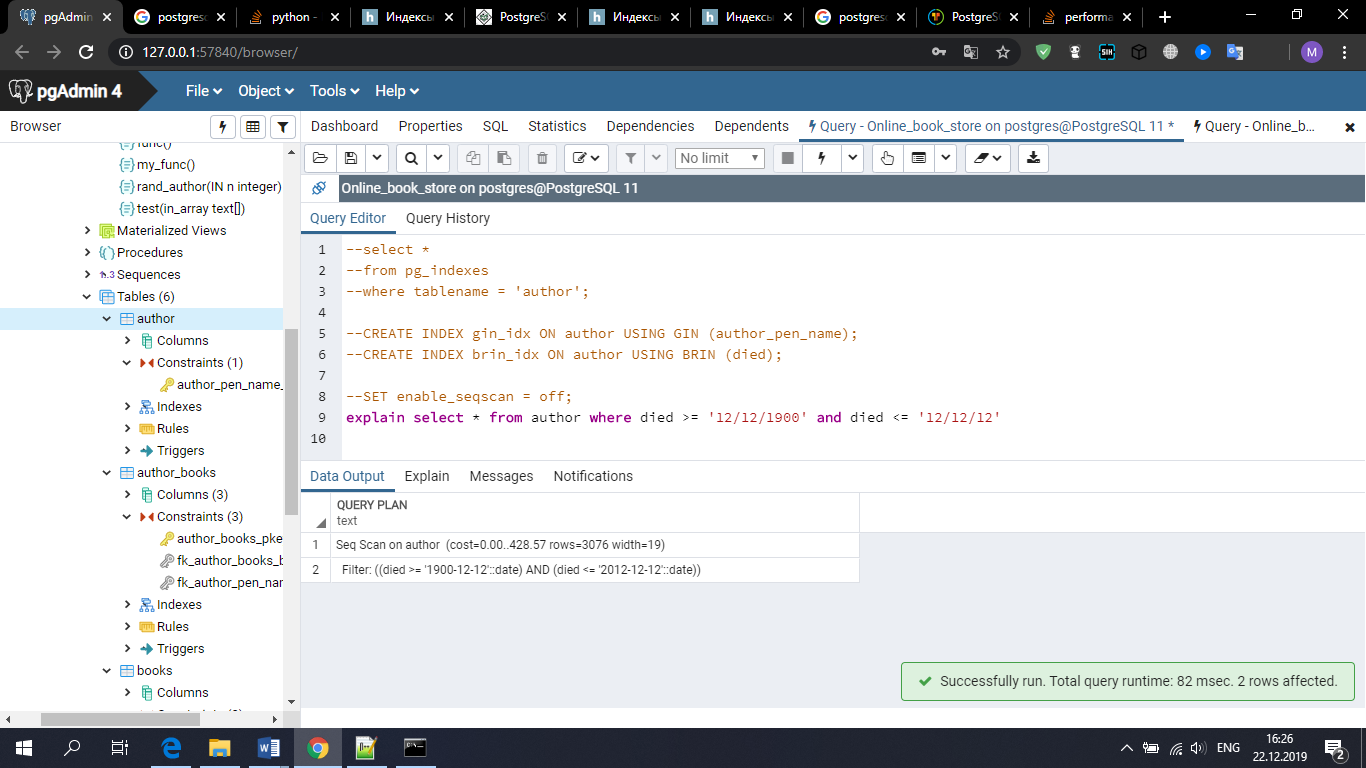
f'purchase\_date - {self.purchase\_date}\n'

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

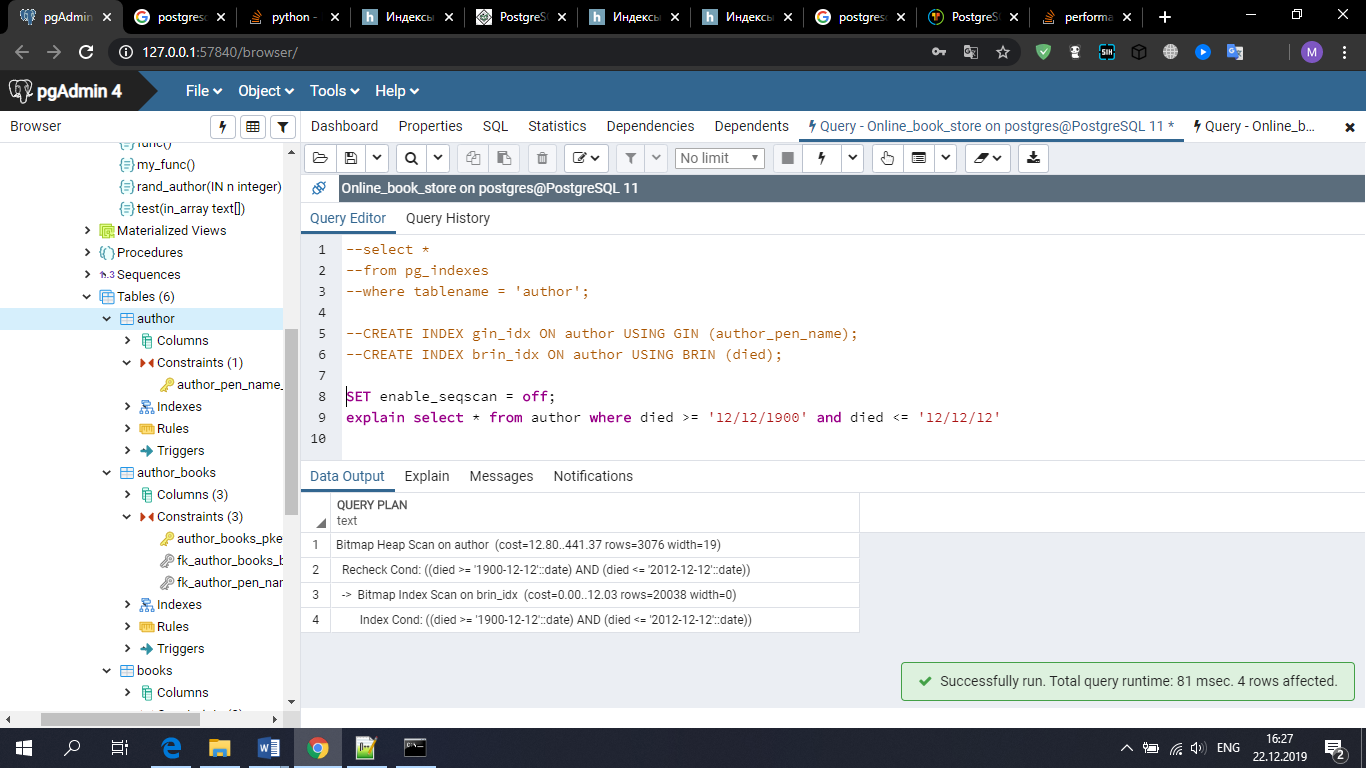
f'book\_title - {self.book\_title}')

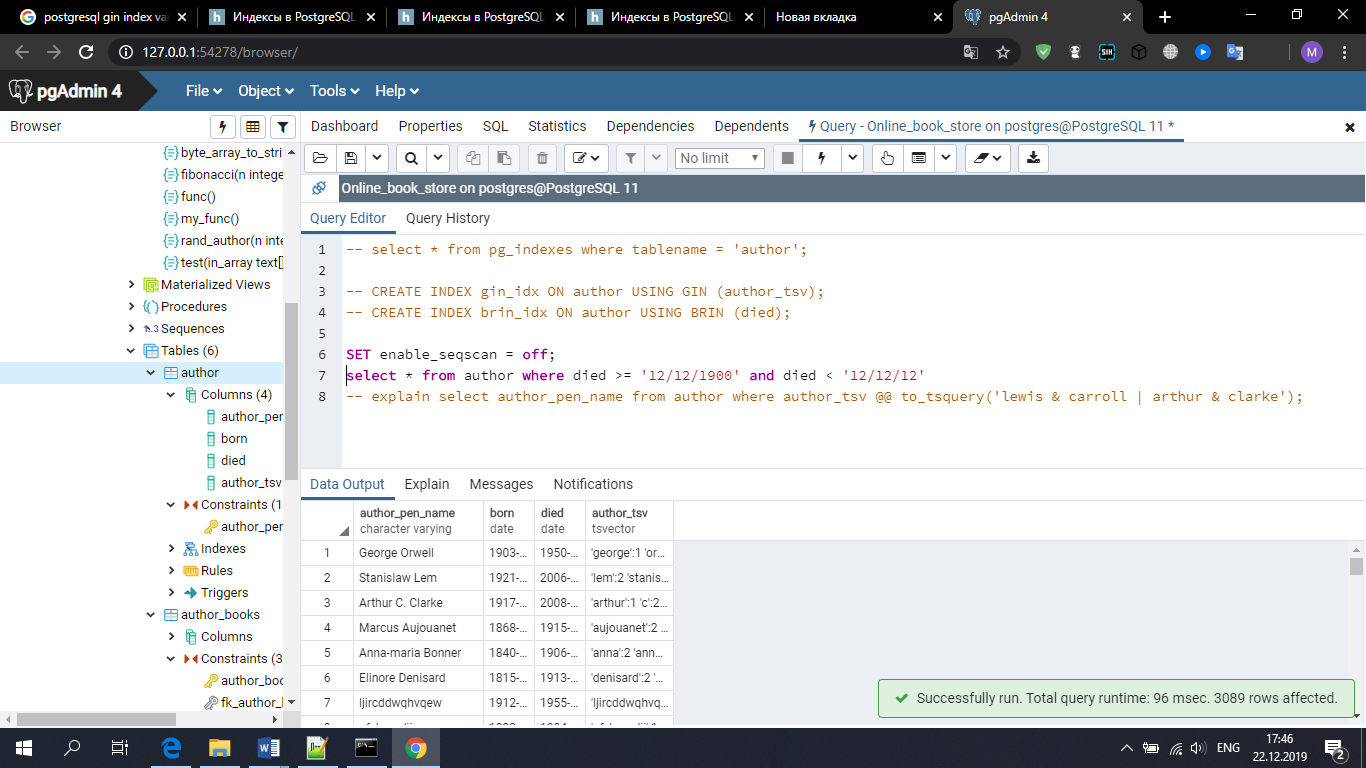
**Дослідження індексів**

Select без індексів, використовується seq scan

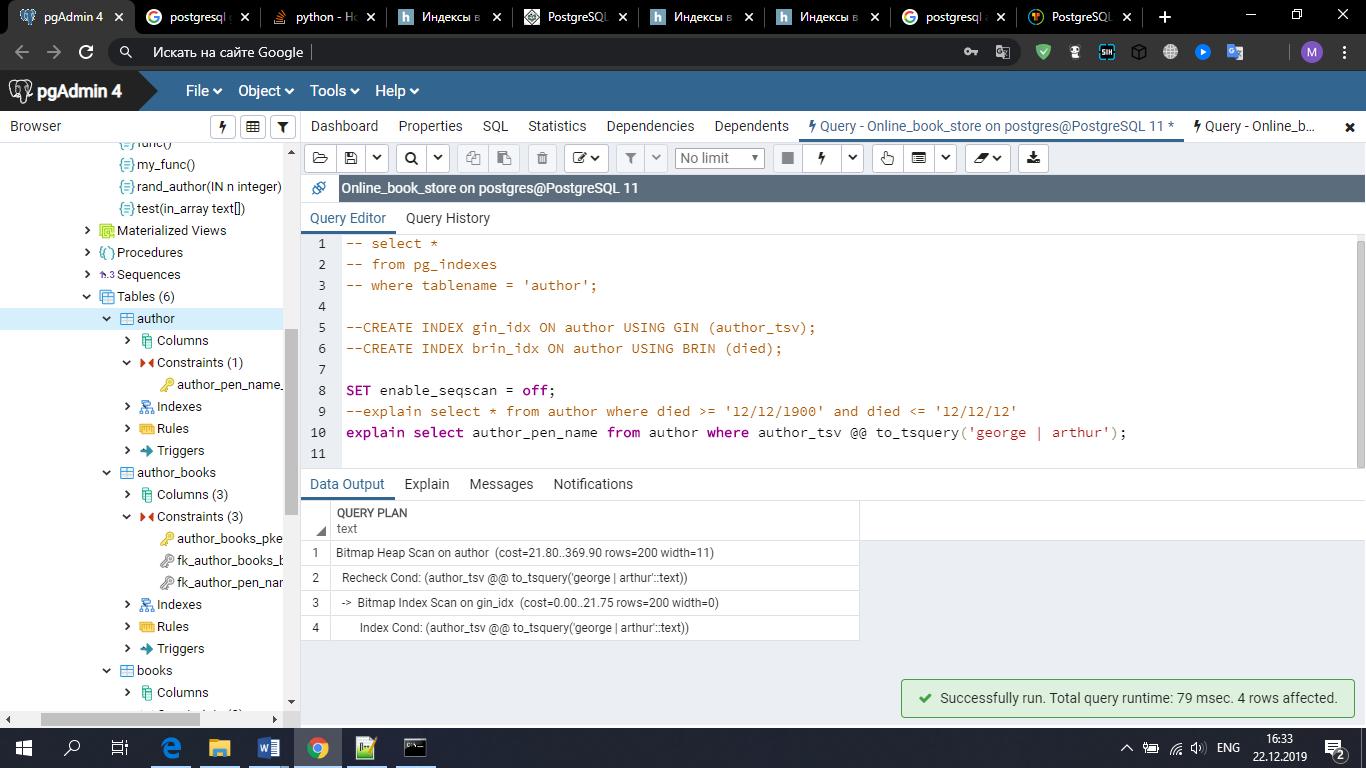


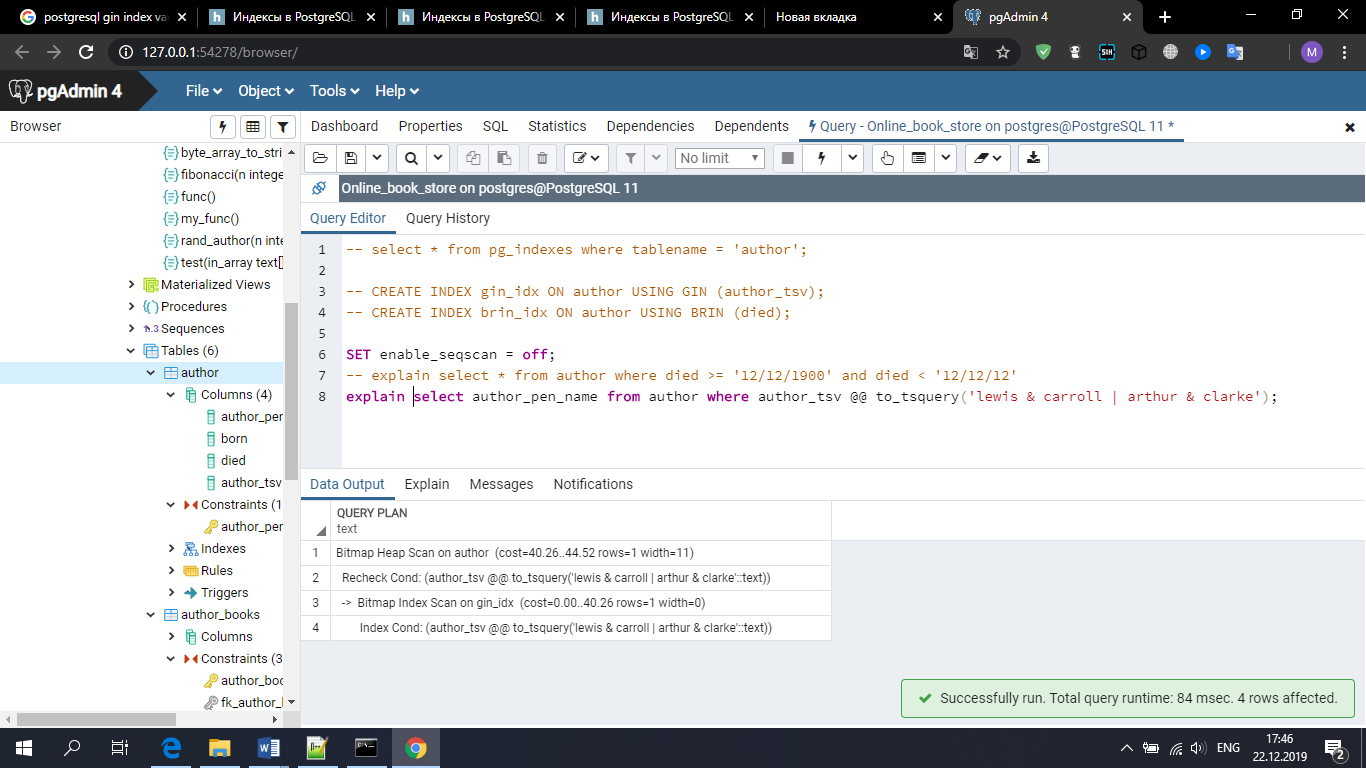
Select з індексом BRIN, використовується Bitmap Heap Scan

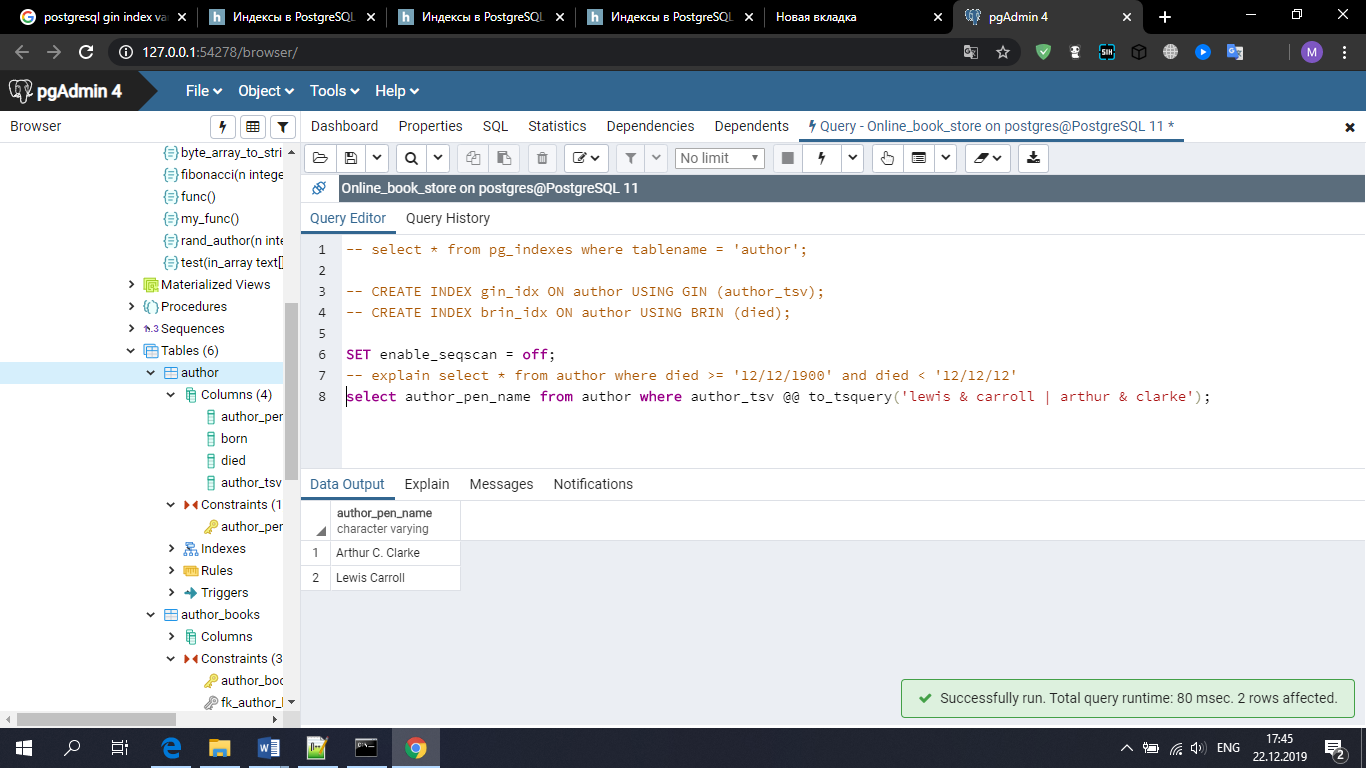




Select з індексом GIN, використовується Bitmap Heap Scan

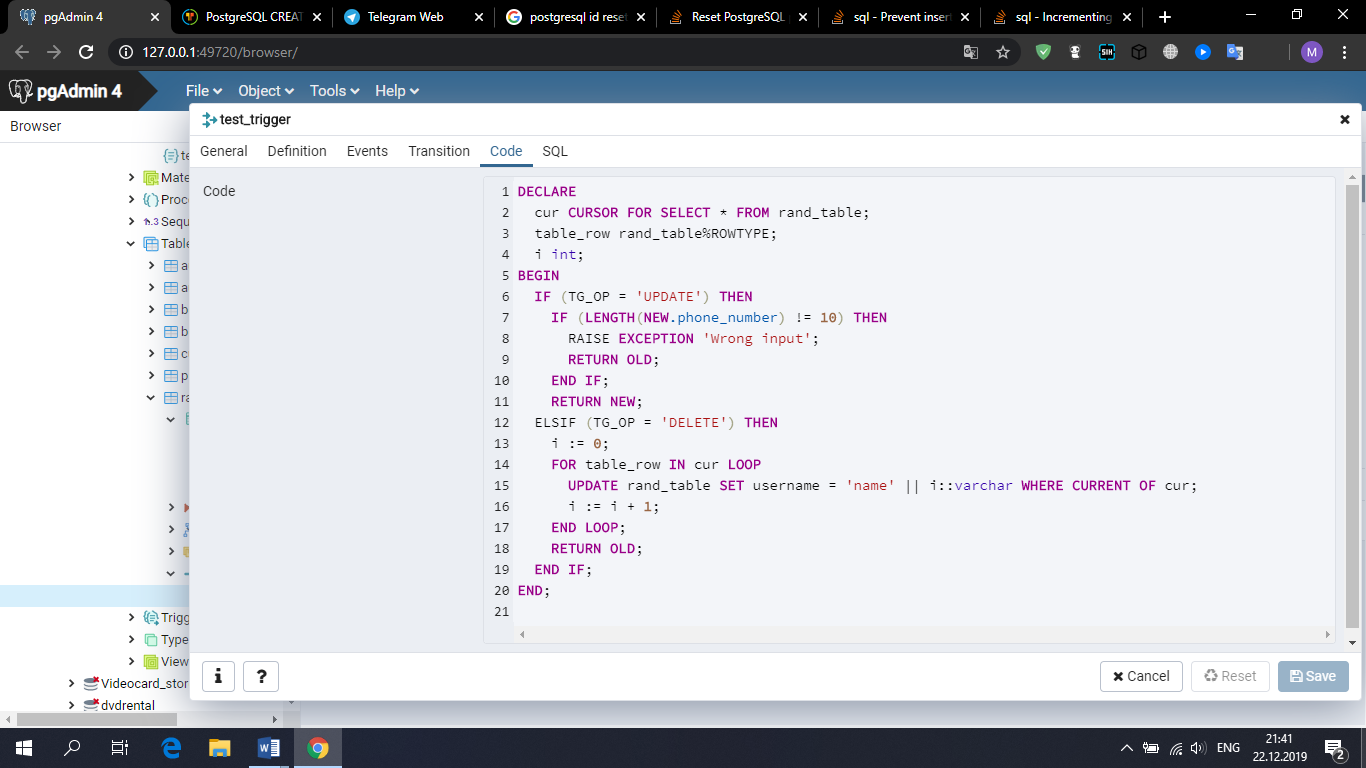


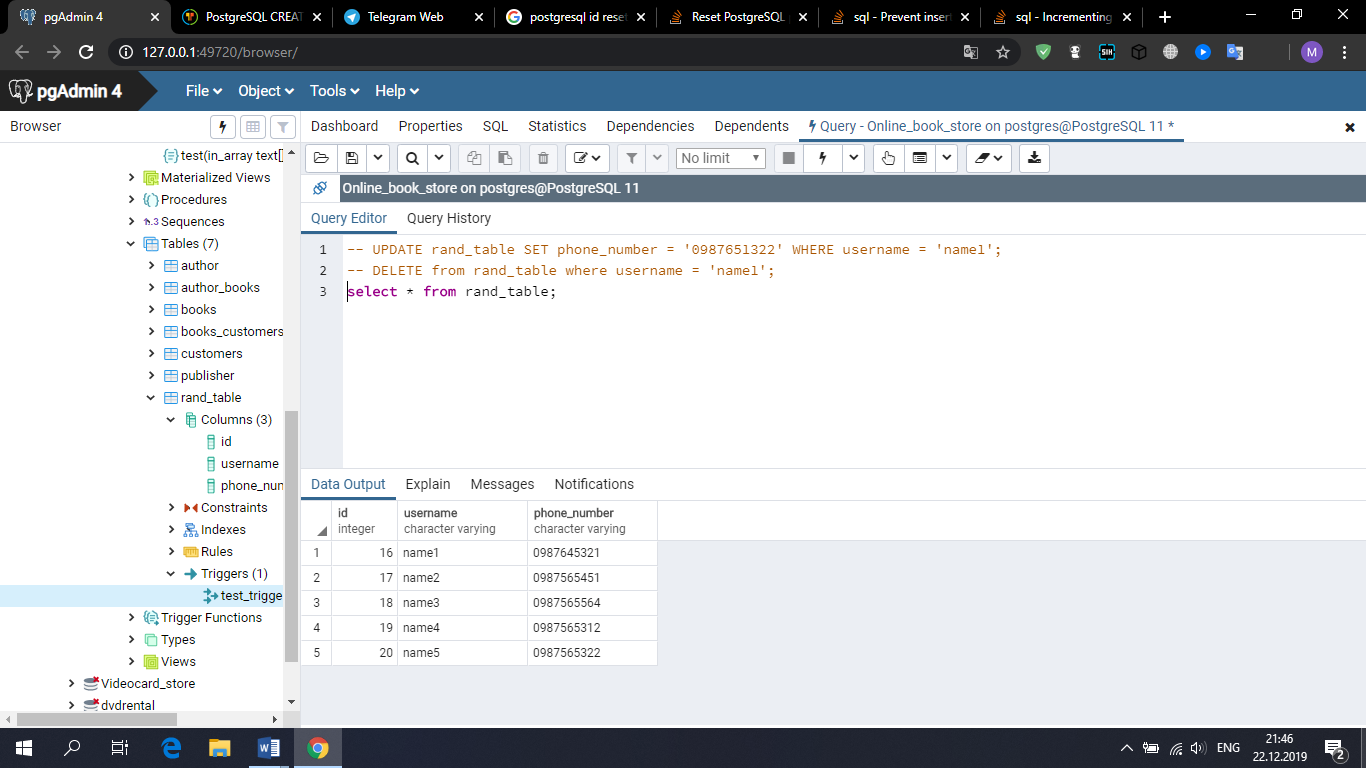




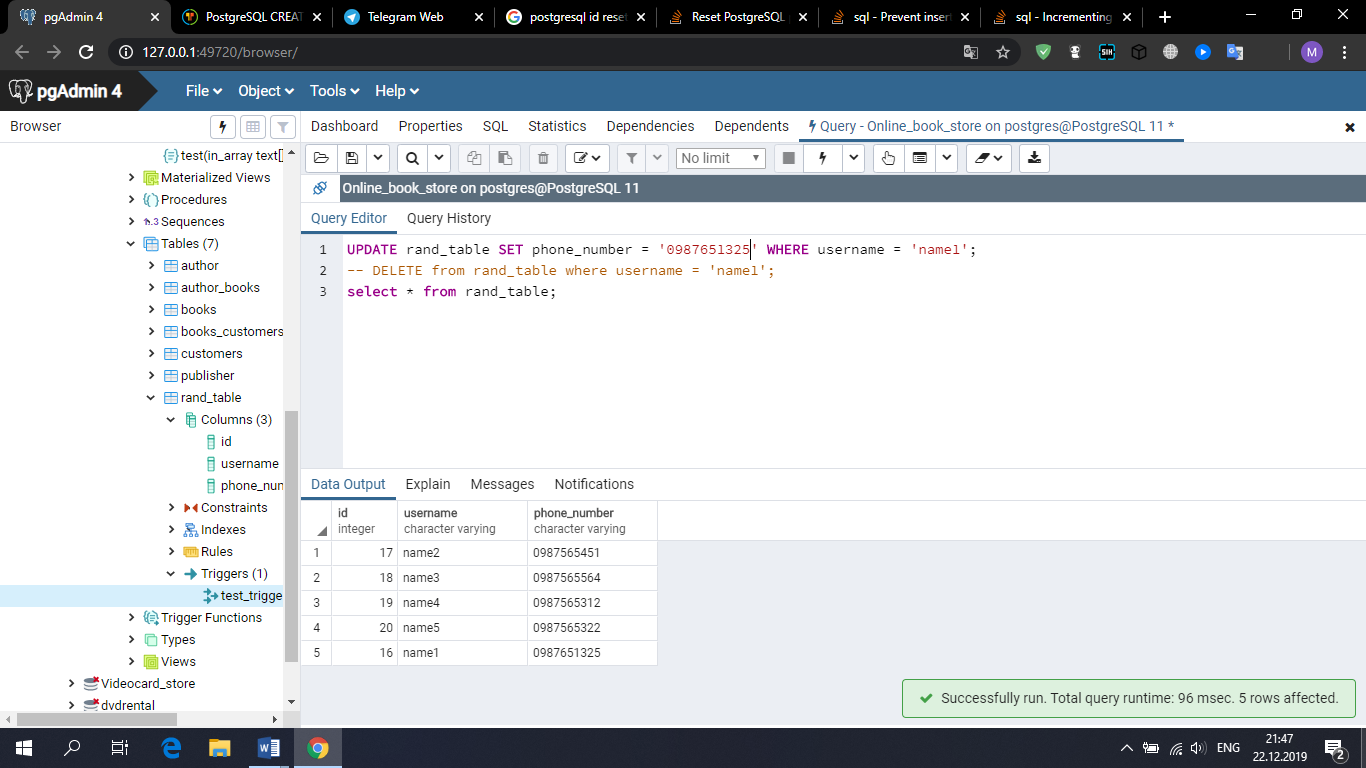
**Тригери**

Тригер при UPDATE перевіряє правильність формату введеного номера, якщо виявлено помилку викликається EXCEPTION. При DELETE тригер оновлює всі поля колонки username і нумерує їх заново.

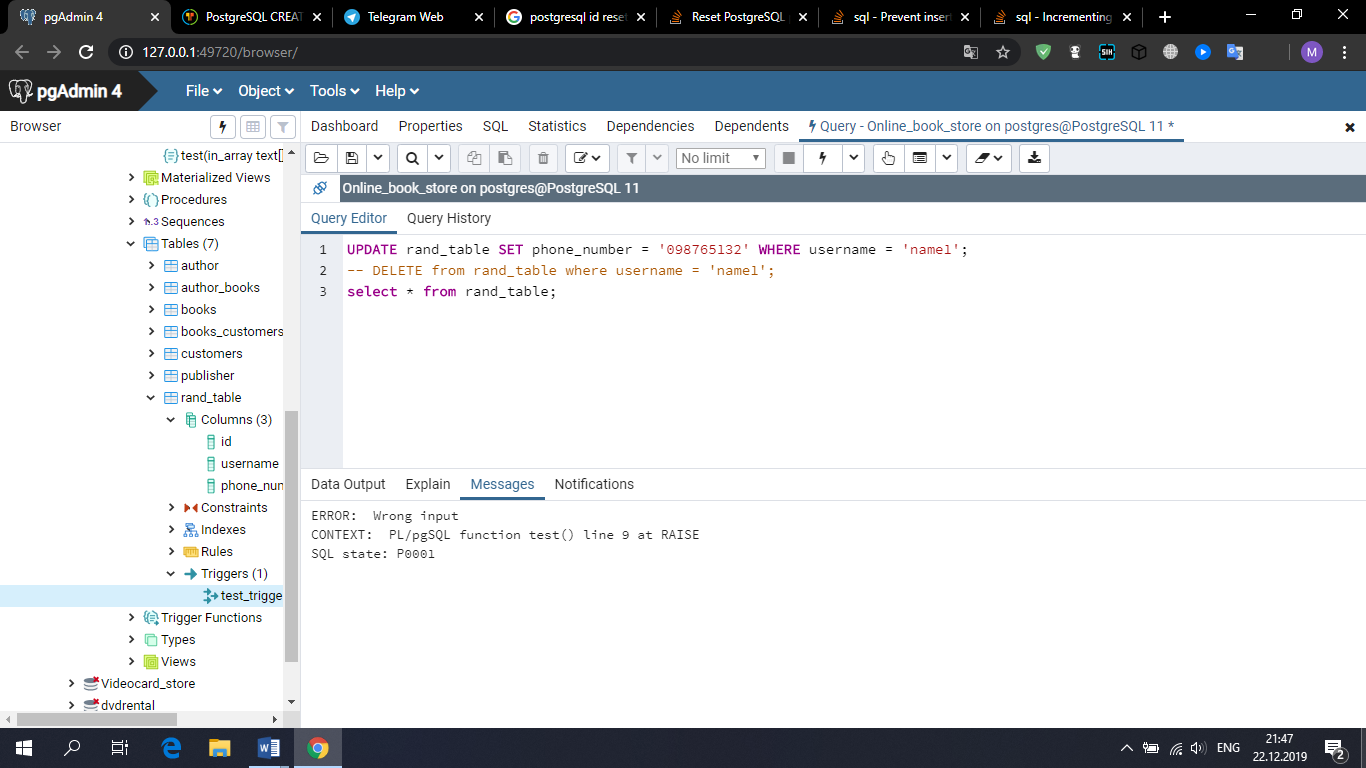




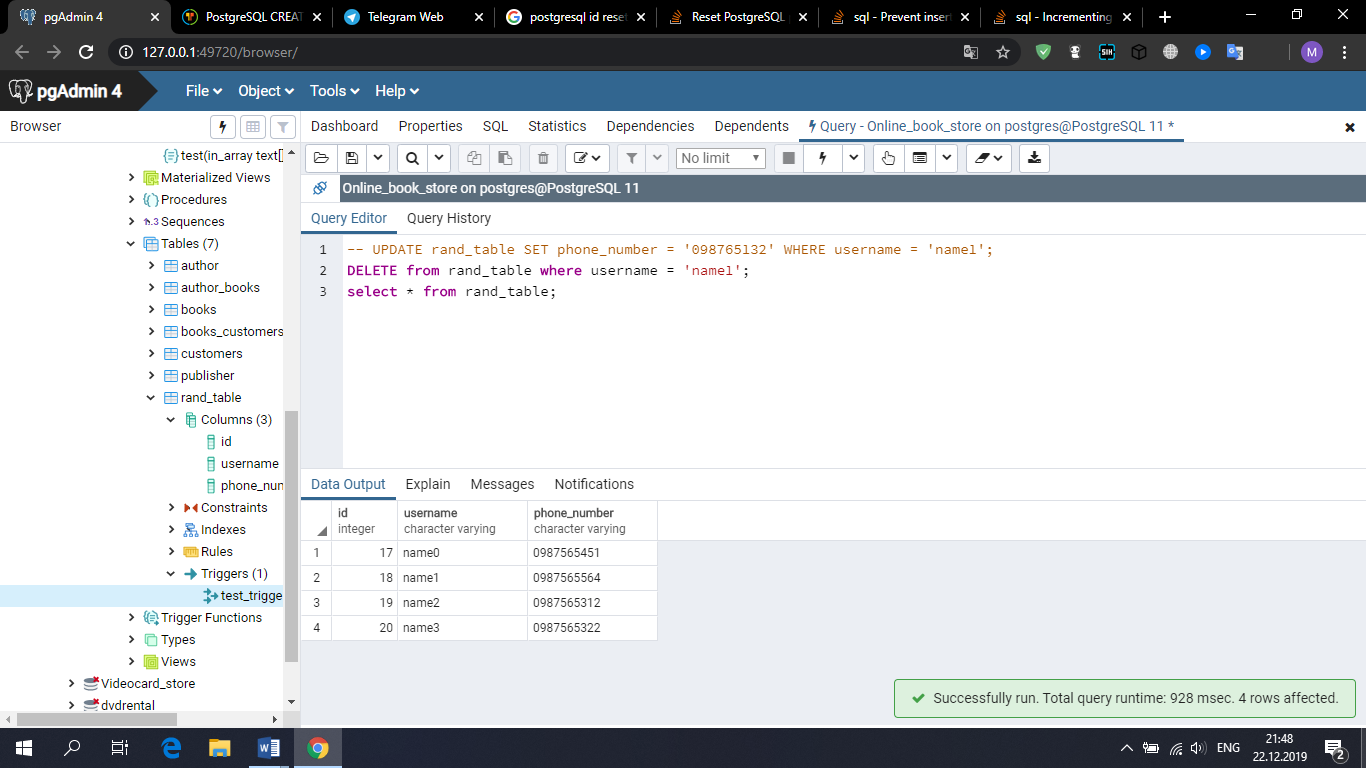
UPDATE, формат номера правильний



UPDATE, формат номера неправильний



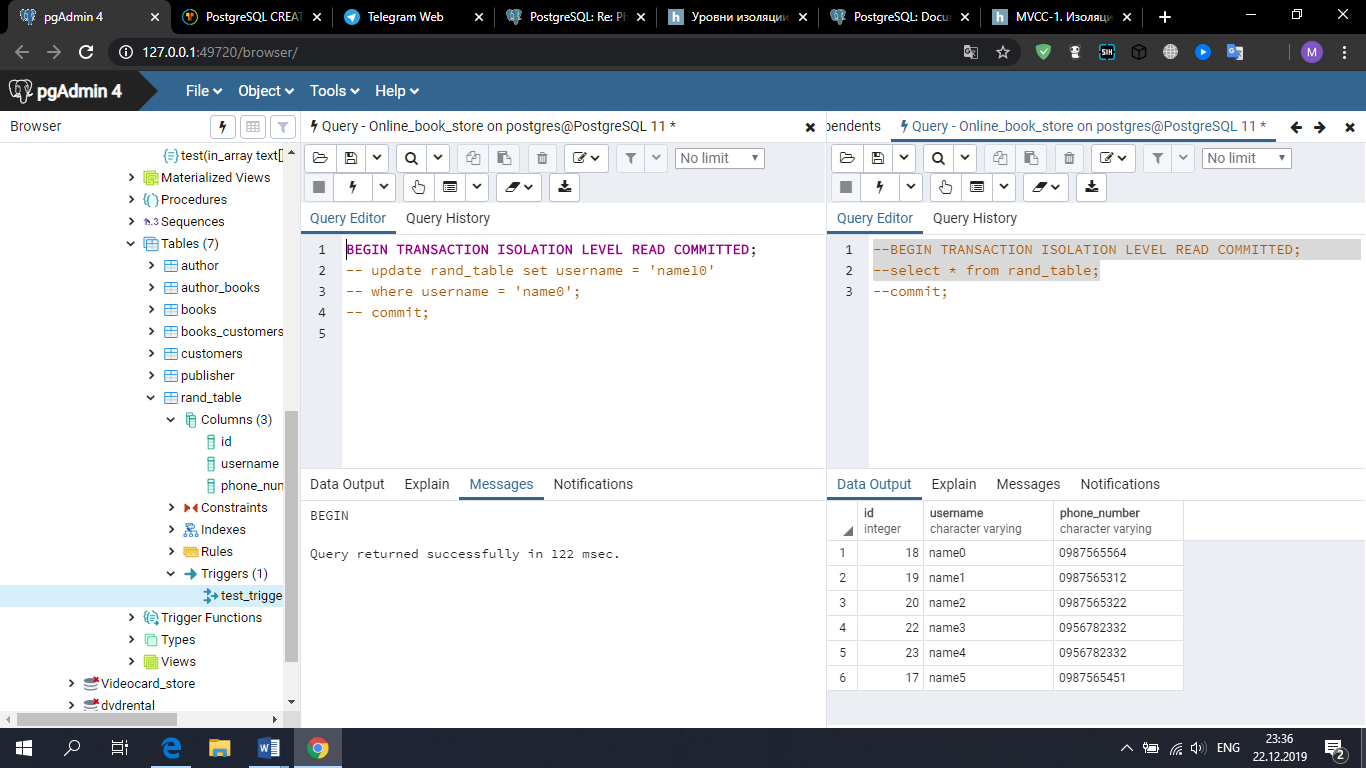
DELETE

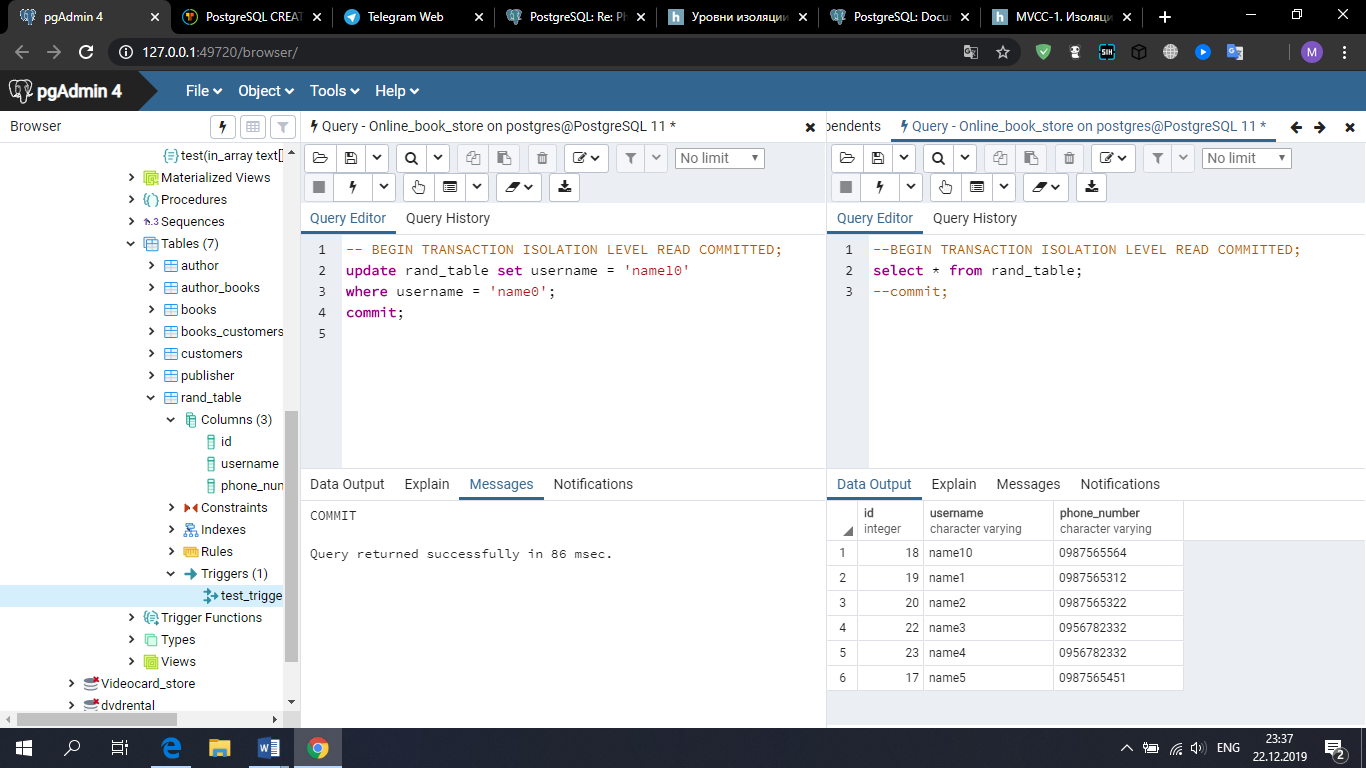


**Дослідження рівнів ізоляції**

**READ COMMITTED**

**Non-repeatable read**





**REPEATABLE READ**

