

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

# «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Тема:

Засоби оптимізації роботи СУБД PostgreSQL

Виконала:

Студенка групи КВ-81

Сущенко Д. В.

Перевірив:

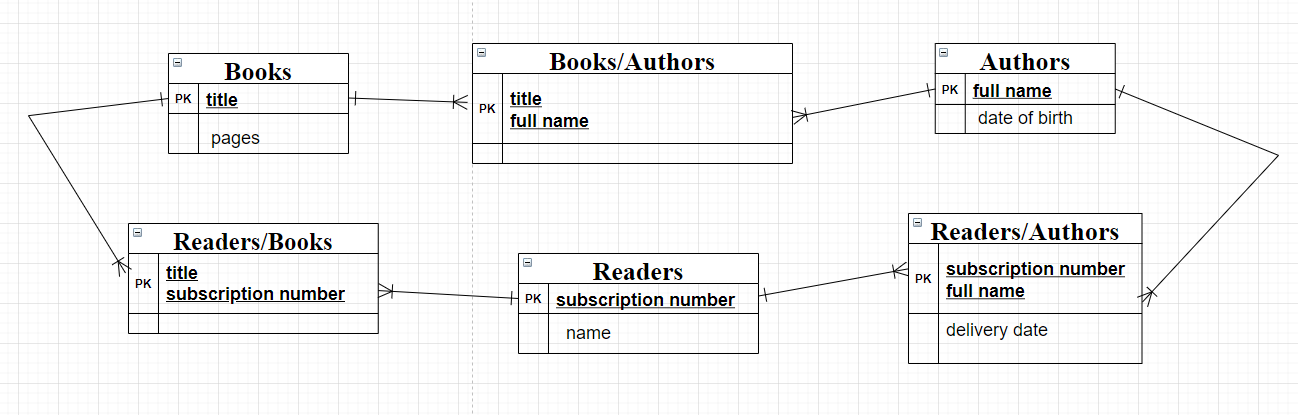
Петрашенко А.В.

**Київ – 2020**

**Варіант 23**

|  |  |  |
| --- | --- | --- |
| *23* | *GIN, Hash* | *before update, delete* |

**Завдання 1**

****

Для роботи з sqlaclemy було імпортовано такі засоби:

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy.orm import relationship

from sqlalchemy import Column, Integer, String, Date, ForeignKey

from sqlalchemy import text

Підключення до бази та створення сесії:

engine = create\_engine('postgresql://postgres:nvidia123@localhost:5432/postgres2')

Session = sessionmaker(bind=engine)

Base = declarative\_base()

Створені класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками.

class Authors(base.Base):  
 \_\_tablename\_\_ = 'Authors'  
 full\_name = Column('full\_name', String(50), primary\_key=True)  
 date\_of\_birth = Column('date\_of\_birth', Date)  
 books\_authors = relationship('BooksAuthors')  
 readers\_authors = relationship('ReadersAuthors')  
  
 def \_\_repr\_\_(self):  
 return "<Author(full\_name='{}', date\_of\_birth='{})>"\  
 .format(self.full\_name, self.date\_of\_birth)  
  
class Books(base.Base):  
 \_\_tablename\_\_ = 'Books'  
 title = Column('title', String(50), primary\_key=True)  
 pages = Column('pages', Integer)  
 books\_authors = relationship('BooksAuthors')  
 readers\_books = relationship('ReadersBooks')  
  
 def \_\_repr\_\_(self):  
 return "<Book(title='{}', pages={})>".format(self.title, self.pages)  
  
class Readers(base.Base):  
 \_\_tablename\_\_ = 'Readers'  
 subscription\_number = Column('subscription\_number', Integer, primary\_key=True);  
 name = Column('name', String(50))  
 readers\_authors = relationship("ReadersAuthors")  
 readers\_books = relationship("ReadersBooks")  
  
 def \_\_repr\_\_(self):  
 return "<Readers(subscription\_number='{}', name='{}')>"\  
 .format(self.subscription\_number, self.name)  
  
class BooksAuthors(base.Base):  
 \_\_tablename\_\_ = 'Books/Authors'  
 title = Column('title', String(50), ForeignKey('Books.title'), primary\_key=True)  
 full\_name = Column('full\_name', String(50), ForeignKey('Authors.full\_name'), primary\_key=True)  
  
class ReadersAuthors(base.Base):  
 \_\_tablename\_\_ = 'Readers/Authors'  
 subscription\_number = Column('subscription\_number', Integer,  
 ForeignKey('Readers.subscription\_number'), primary\_key=True)  
 full\_name = Column('full name', String(50), ForeignKey('Authors.full\_name'), primary\_key=True)  
 delivery\_date = Column('delivery date', Date)  
  
class ReadersBooks(base.Base):  
 \_\_tablename\_\_ = 'Readers/Books'  
 title = Column('title', String(50), ForeignKey('Books.title'), primary\_key=True)  
 subscription\_number = Column('subscription\_number', Integer,  
 ForeignKey('Readers.subscription\_number'), primary\_key=True)  
  
def get\_class\_by\_tablename(tablename):  
  
 for c in base.Base.\_decl\_class\_registry.values():  
 if hasattr(c, '\_\_tablename\_\_') and c.\_\_tablename\_\_ == tablename:  
 return c

Функції, що реалізують вставки, вилучення та редагування екземплярів класів-сутностей.

def add\_row1(self, table\_name, fields\_name):

try:

obj = get\_class\_by\_tablename(table\_name)(\*\*fields\_name)

self.session.add(obj)

self.session.commit()

return True

except(Exception, sqlalchemy.exc.SQLAlchemyError) as error:

print(error)

self.session.rollback()

return False

def del\_row1(self, table\_name, key, value):

try:

table\_class = get\_class\_by\_tablename(table\_name)

filter\_txt = "{} = '{}'".format(key, value)

self.session.query(table\_class).filter(text(filter\_txt)).delete(synchronize\_session=False)

self.session.commit()

except(Exception, sqlalchemy.exc.SQLAlchemyError) as error:

print(error)

self.session.rollback()

return False

def edit\_value1(self, table\_name, key, key\_change, new\_val, key\_val):

try:

table\_class = get\_class\_by\_tablename(table\_name)

filter\_txt = "{} = '{}'".format(key, key\_val)

update\_values = {key\_change: new\_val}

self.session.query(table\_class) \

.filter(text(filter\_txt)) \

.update(update\_values, synchronize\_session=False)

self.session.commit()

except(Exception, sqlalchemy.exc.SQLAlchemyError) as error:

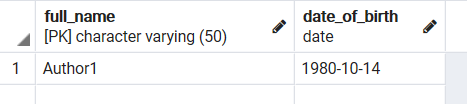
print(error)

self.session.rollback()

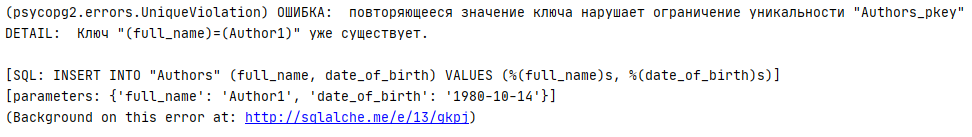
return False

Приклади використання.

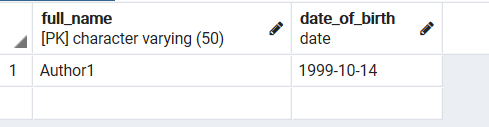
db.add\_row('Authors', {'full\_name': 'Author1', 'date\_of\_birth': '1980-10-14'})



Повторне додавання того ж рядку

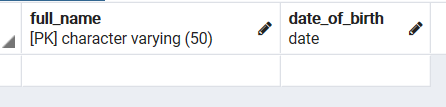


db.edit\_value('Authors', 'full\_name', 'date\_of\_birth', '1999-10-14', 'Author1')



Видалення рядка

db.del\_row('Authors', 'full\_name', 'Author1')



Звичайний easy пошук

cur.execute("""SELECT \* FROM {}

WHERE {} = '{}' ORDER BY {}""".format(table\_name, column\_name, value, column\_name))

Там де певний рядок null

cur.execute("""SELECT \* FROM {}

WHERE {} = 'NULL' or {} = '0' or {} IS NULL""" format(table\_name, column\_name, column\_name, column\_name ))

Між певними значеннями

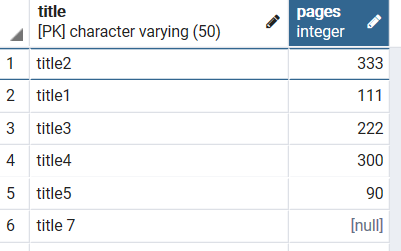
cur.execute("""SELECT \* FROM {}

WHERE {} BETWEEN {} AND {}""".format(table\_name, column\_name, limit1, limin2))

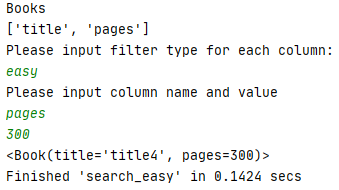
Функції, що реалізують ці запити

@timer  
def search\_easy(con, table\_name, column\_name, value):  
 try:  
 table\_class = get\_class\_by\_tablename(table\_name)  
 col\_name = table\_class.\_\_dict\_\_[column\_name]  
 filter\_txt = '{} = {}'.format(column\_name, value)  
 query = con.query(table\_class).filter(text(filter\_txt)).order\_by(col\_name)  
 print(\*query.all())  
 except (Exception, sqlalchemy.exc.SQLAlchemyError) as error:  
 print(error)  
   
   
@timer  
def search\_is\_null(con, table\_name, column\_name):  
 try:  
 table\_class = get\_class\_by\_tablename(table\_name)  
 filter\_txt = '{} IS NULL'.format(column\_name)  
 query = con.query(table\_class).filter(text(filter\_txt))  
 print(\*query.all())  
 except (Exception, sqlalchemy.exc.SQLAlchemyError) as error:  
 print(error)  
  
  
@timer  
def search\_between(con, table\_name, column\_name, limit1, limit2):  
 try:  
 table\_class = get\_class\_by\_tablename(table\_name)  
 filter\_txt = '{} BETWEEN {} AND {}'.format(column\_name, limit1, limit2)  
 query = con.query(table\_class).filter(text(filter\_txt))  
 print(\*query.all())  
 except (Exception, sqlalchemy.exc.SQLAlchemyError) as error:  
 print(error)

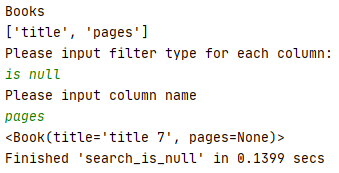
Таблиця для тестування пошуку



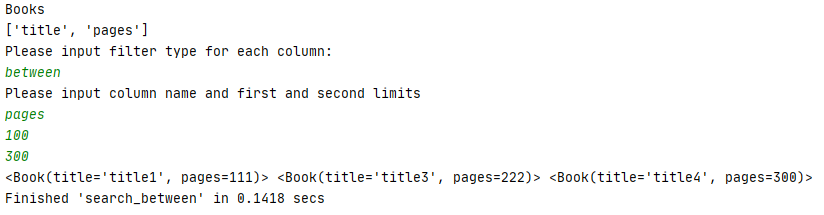
Easy search



Is null



Between



**Завдання 2**

В даному завданні розглянемо використання індексів GIN, Hash. Для створення індексів для текстового поля використовувалися такі команди:

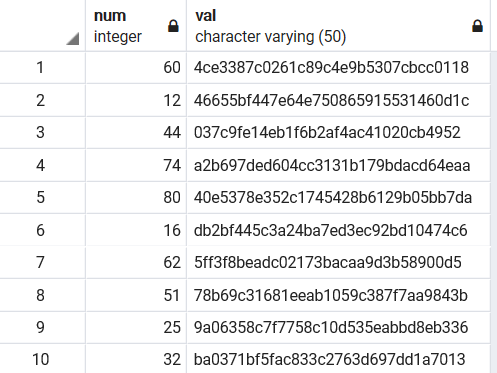
CREATE INDEX txt\_idx ON "Test" USING GIN (val gin\_trgm\_ops);

CREATE INDEX txt\_idx ON "Test" USING Hash (val);

Для створення індексів по числовому полю для Hash можливо скористатися командою:

CREATE INDEX num\_idx ON "Test" USING Hash (num);

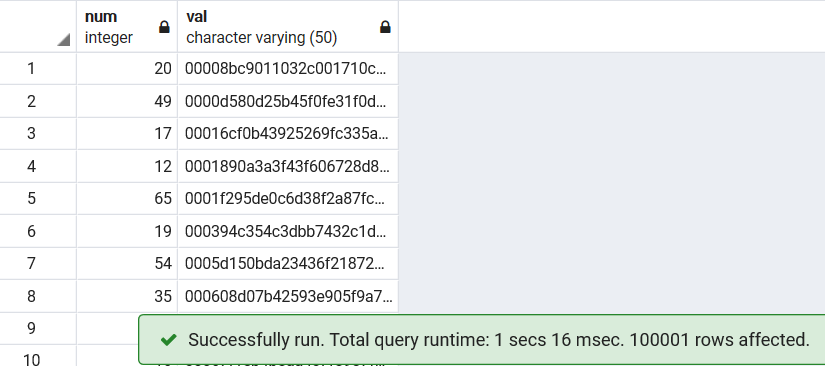
Результат запиту SELECT \* FROM public."Test"



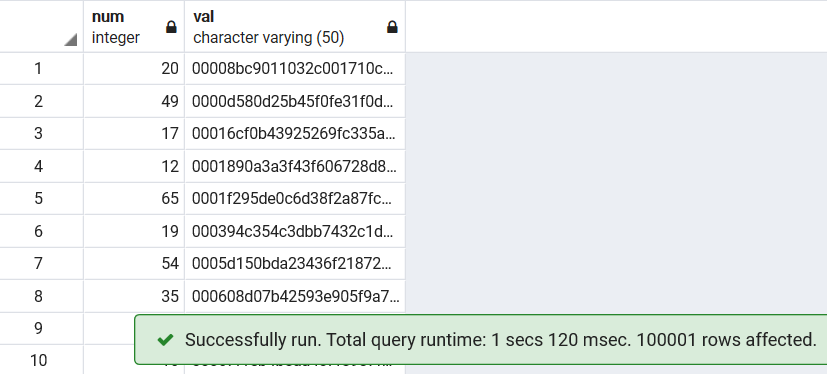
**Запит №1**

SELECT num, val FROM "Test" ORDER BY val;

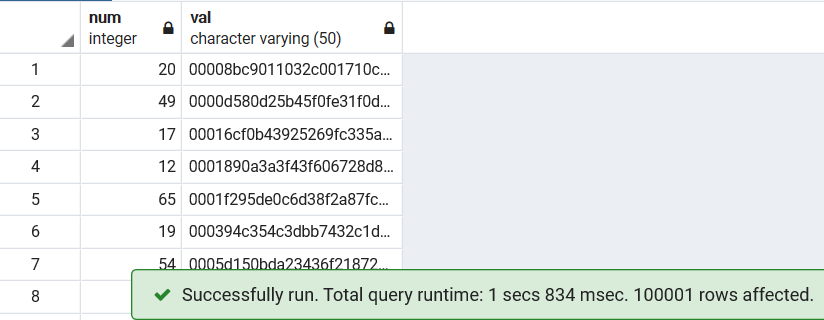
Без індексування.



Hash



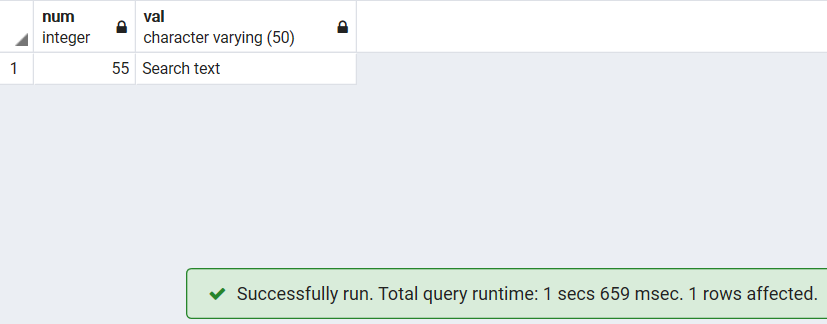
GIN



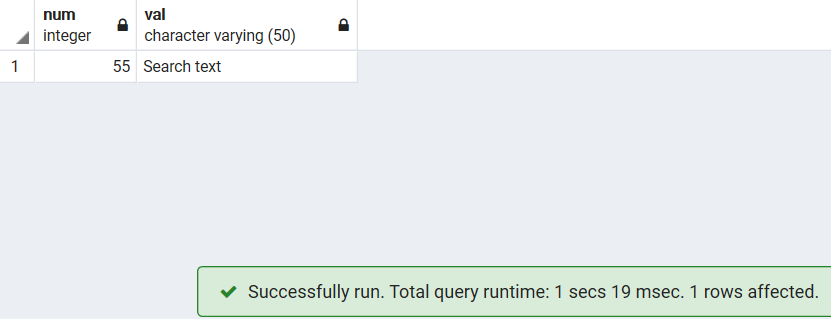
**Запит №2**

SELECT \* FROM "Test" WHERE val = 'Search text';

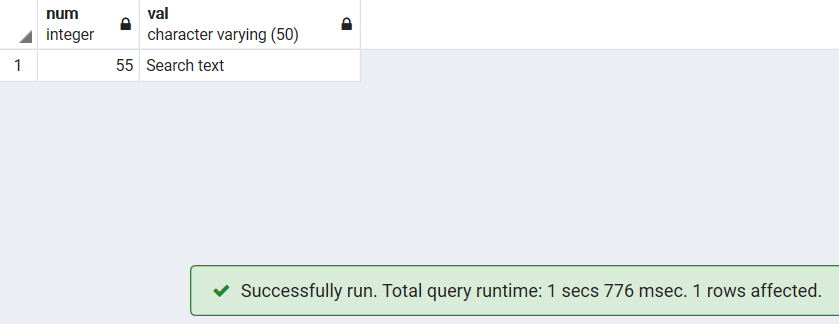
Без індексування



Hash



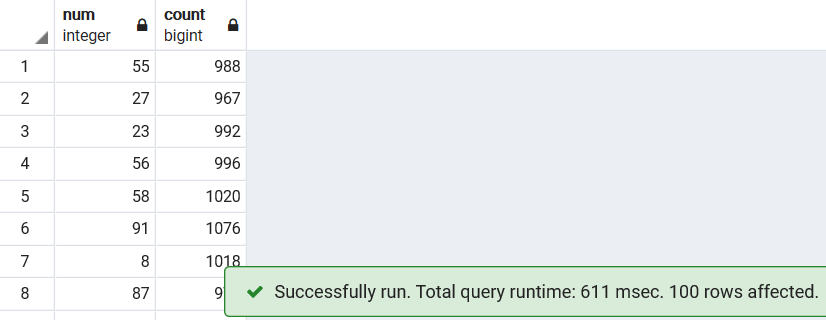
GIN



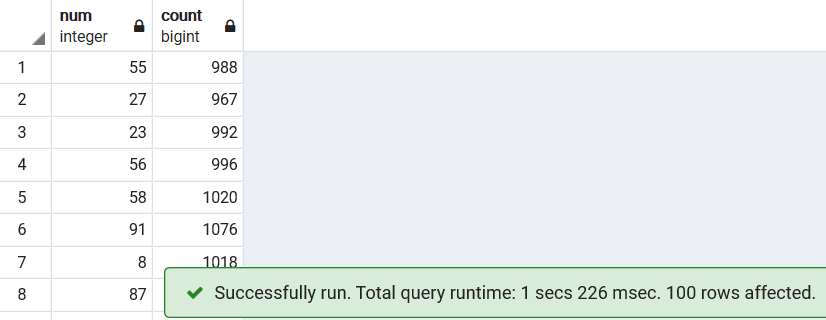
**Запит №3**

SELECT num, count(num) FROM "Test" GROUP BY num

Без індексування



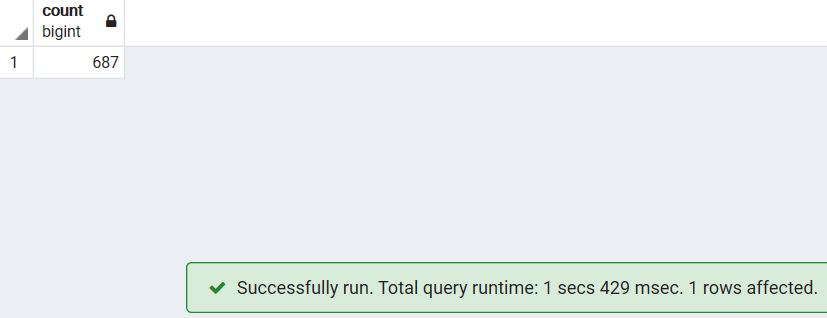
Hash



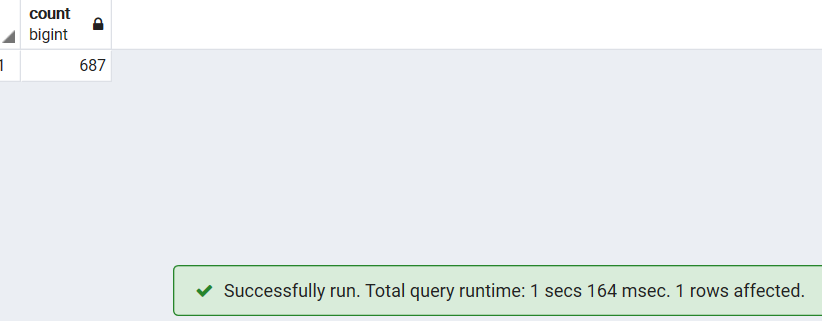
**Запит №4**

SELECT count(\*) FROM "Test" WHERE val ILIKE '%aaa%';

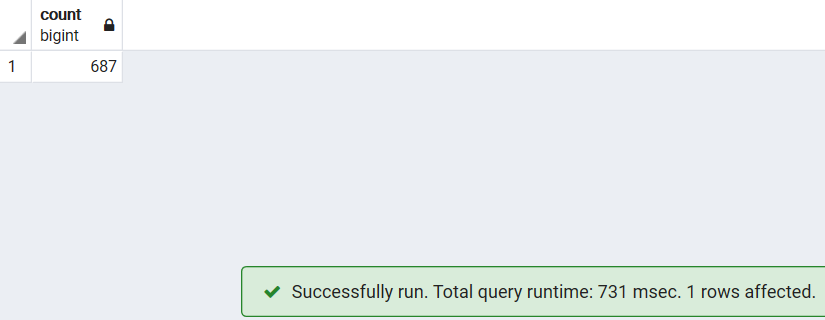
Без індексування



Hash



GIN



**Висновок**

Індекс GIN добре використовувати коли необхідно виконувати пошук символів у тексті, або масиві. Основним недоліком цього індексу є його обмеженість у використання (не може наприклад використовуватися з числовими типами даних) і досить тривале створення. Індекс GIN краще використовувати у запитах в яких присутні такі операції як LIKE, ILIKE.

Індекс Hash добре використовувати коли необхідно прискорити пошук і також важливим є розмір індексу. Що до розміру індексу Hash, то він є «плоским» в порівнянні з індексом Btree, тому є значно меншим за розміром.

Найбільшою проблемою Hash є те що він обмежений лише оператором рівності, тому треба шукати тільки точні збіги. І він краще працює з числами ніж з рядками символів.

**Завдання 3**

***Створення тригерної функції.***

CREATE OR REPLACE FUNCTION trig\_func() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'UPDATE') THEN

IF NEW.subscription\_number < 0 THEN

RAISE EXCEPTION 'NOT CORRECT NUMBER. SHOULD NOT BE UPDATED';

END IF;

IF NEW.name is NULL THEN

RAISE EXCEPTION 'NOT CORRECT NAME. SHOULD NOT BE UPDATED';

END IF;

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

IF NEW.subscription\_number < 0 THEN

RAISE EXCEPTION 'NOT CORRECT NUMBER. SHOULD NOT BE DELETED';

END IF;

RETURN OLD;

END IF;

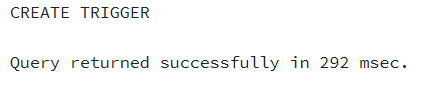
RETURN NEW;

END;

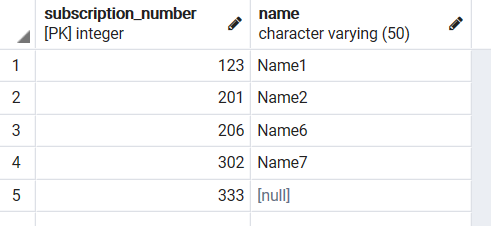
$$;

CREATE TRIGGER trig\_func BEFORE DELETE OR UPDATE ON "Readers"

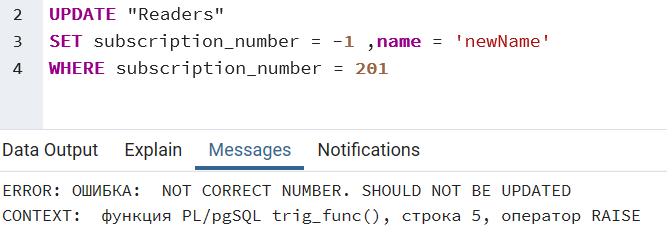
FOR EACH ROW EXECUTE FUNCTION trig\_func();

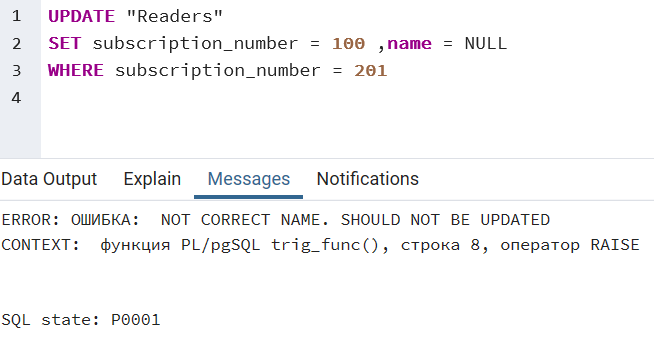


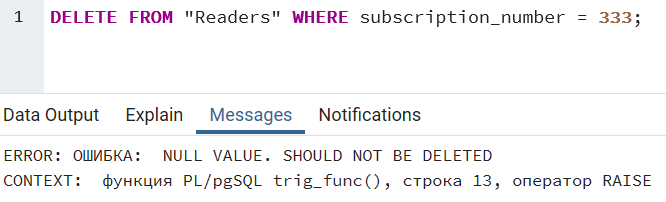
Зміст таблиці "Readers"

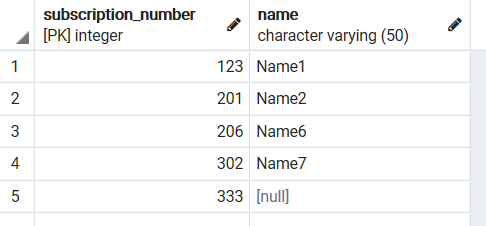


Приклади використання тригеру









**Завдання 4**

