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 та засоби управління»**

**ТЕМА: «ЗАСОБИ ОПТИМІЗАЦІЇ РОБОТИ СУБД POSTGRESQL»**

Виконав: студент ІІI курсу

ФПМ групи КВ-04

Оніщук А.О.

Київ – 2022

*Метою роботи* є здобуття практичних навичок використання засобів оптимізації 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 пункту завдання №1*

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlAlchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:М, М:М та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.

*Вимоги до пункту завдання №2*

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

*Вимоги до пункту завдання №3*

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

*Вимоги до пункту завдання №4*

Проаналізувати на прикладах використання рівнів ізоляції транзакцій READ COMMITTED, REPEATABLE READ та SERIALIZABLE, продемонструвавши феномени, які виникають, і спосіб їх уникнення завдяки встановленню відповідного рівня ізоляції транзакцій. Для виконання завдання необхідно відкрити дві транзакції у різних вікнах pgAdmin4 і виконати послідовність запитів INSERT, UPDATE або DELETE у обох транзакціях, що доводять наявність або відсутність певних феноменів.

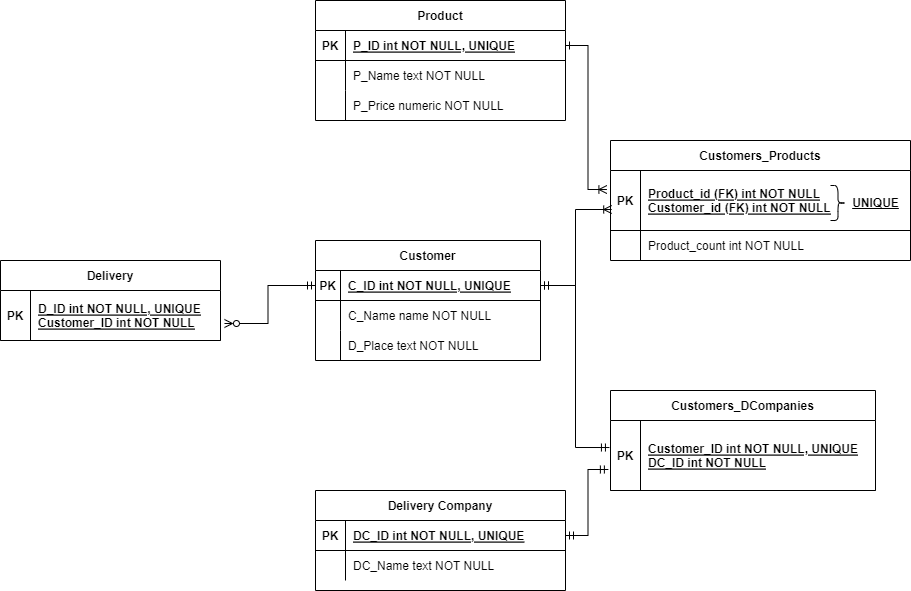
**Варіант 17**



Бібліотека для реалізації ORM - SQLAlchemy

Git-репозиторій: https://github.com/ViDjet-Git/Data-Base-University

**Схема бази даних**



**Завдання №1**

**Реалізація класів ORM**

**Delivery:**

**class** delivery(Base):

\_\_tablename\_\_ = 'Delivery'

D\_ID = Column('D\_ID', Integer, primary\_key=True)

Customer\_ID = Column('Customer\_ID', Text, ForeignKey('Customer.C\_ID'), primary\_key=True)

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

**return** "<Delivery(Customer\_ID='{}')>" \

.format(self.Customer\_ID)

**Product:**

**class** product(Base):

\_\_tablename\_\_ = 'Product'

P\_ID = Column('P\_ID', Integer, primary\_key=True)

P\_Name = Column('P\_Name', Text)

P\_Price = Column('P\_Price', Numeric)

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

**return** "<Product(P\_Name='{}', P\_Price='{}')>" \

.format(self.P\_Name, self.P\_Price)

**Customer:**

**class** customer(Base):

\_\_tablename\_\_ = 'Customer'

C\_ID = Column('C\_ID', Integer, primary\_key=True)

C\_Name = Column('C\_Name', Text)

D\_Place = Column('D\_Place', Text)

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

**return** "<Customer(C\_Name='{}', D\_Place='{}')>" \

.format(self.C\_Name, self.D\_Place)

**Delivery\_Company:**

**class** delivery\_company(Base):

\_\_tablename\_\_ = 'Delivery Company'

DC\_ID = Column('DC\_ID', Integer, primary\_key=True)

DC\_Name = Column('DC\_Name', Text)

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

**return** "<Delivery Company(DC\_Name='{}')>" \

.format(self.DC\_Name)

**Customers\_Products:**

**class** customers\_products(Base):

\_\_tablename\_\_ = 'Customers\_Products'

Product\_id = Column('Product\_id', Integer, ForeignKey('Product.P\_ID'), primary\_key=True)

Customer\_id = Column('Customer\_id', Integer, ForeignKey('Customer.C\_ID'), primary\_key=True)

Product\_count = Column('Product\_count', Integer)

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

**return** "<Customers\_Products(Product\_id='{}', Customer\_id='{}', Product\_count='{}')>" \

.format(self.Product\_id, self.Customer\_id, self.Product\_count)

**Delivery:**

**class** customers\_DCompanies(Base):

\_\_tablename\_\_ = 'Customers\_DCompanies'

Customer\_ID = Column('Customer\_ID', Integer, ForeignKey('Customer.C\_ID'), primary\_key=True)

DC\_ID = Column('DC\_ID', Integer, ForeignKey('Delivery Company.DC\_ID'), primary\_key=True)

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

**return** "<Customers\_DCompanies(Customer\_ID='{}', DC\_ID='{}')>" \

.format(self.Customer\_ID, self.DC\_ID)

**Реалізація функцій**

**Insert:**

**def** add\_row(table\_name, files):

**try**:

**if**(table\_name == 'Customer'):

temp = customer(

C\_Name=list(files.values())[0],

D\_Place=list(files.values())[1]

)

**elif**(table\_name == 'Product'):

temp = product(

P\_Name=list(files.values())[0],

P\_Price=list(files.values())[1]

)

**elif**(table\_name == 'Delivery Company'):

temp = delivery\_company(

DC\_Name=list(files.values())[0]

)

**elif**(table\_name == 'Delivery'):

temp = delivery(

Customer\_ID=list(files.values())[0]

)

**elif**(table\_name == 'Customers\_Products'):

temp = customers\_products(

Product\_id=list(files.values())[0],

Customer\_id=list(files.values())[1],

Product\_count=list(files.values())[2]

)

**elif**(table\_name == 'Customers\_DCompanies'):

temp = customers\_DCompanies(

Customer\_ID=list(files.values())[0],

DC\_ID=list(files.values())[1]

)

s.add(temp)

s.commit()

**except** Exception **as** error:

**print**(error)

s.rollback()

**return** False

**Delete:**

**def** del\_row(table\_name, key, value):

**try**:

**if** type(value) **is** str:

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

**else**:

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

**if** (table\_name == 'Customer'):

s.query(customer).filter(text(filter\_txt)).delete(synchronize\_session=False)

**elif** (table\_name == 'Product'):

s.query(product).filter(text(filter\_txt)).delete(synchronize\_session=False)

**elif** (table\_name == 'Delivery Company'):

s.query(delivery\_company).filter(text(filter\_txt)).delete(synchronize\_session=False)

**elif** (table\_name == 'Delivery'):

s.query(delivery).filter(text(filter\_txt)).delete(synchronize\_session=False)

**elif** (table\_name == 'Customers\_Products'):

s.query(customers\_products).filter(text(filter\_txt)).delete(synchronize\_session=False)

**elif** (table\_name == 'Customers\_DCompanies'):

s.query(customers\_DCompanies).filter(text(filter\_txt)).delete(synchronize\_session=False)

s.commit()

**except** Exception **as** error:

**print**(error)

s.rollback()

**return** False

**Update:**

**def** edit\_value(table\_name, key, key\_change, new\_val, key\_val):

**try**:

**if** (table\_name == 'Customer'):

table\_class = customer

**elif** (table\_name == 'Product'):

table\_class = product

**elif** (table\_name == 'Delivery Company'):

table\_class = delivery\_company

**elif** (table\_name == 'Delivery'):

table\_class = delivery

**elif** (table\_name == 'Customers\_Products'):

table\_class = customers\_products

**elif** (table\_name == 'Customers\_DCompanies'):

table\_class = customers\_DCompanies

**if** key\_val **is** str:

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

**else**:

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

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

s.query(table\_class) \

.filter(text(filter\_txt)) \

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

s.commit()

**except** Exception **as** error:

**print**(error)

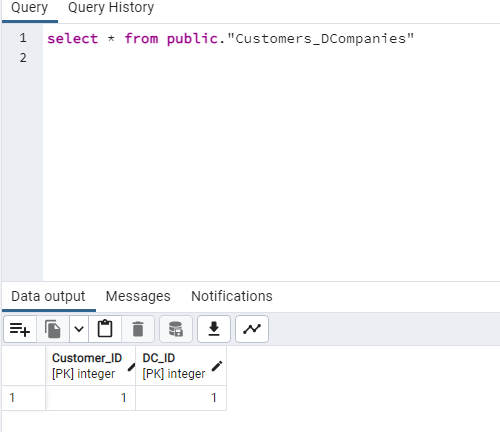
s.rollback()

**return** False

**Приклади виконання запитів**

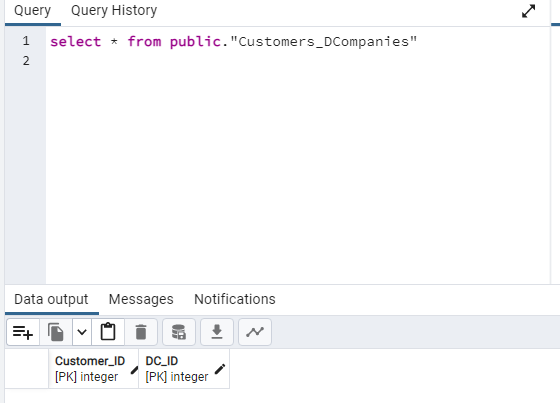
**Вставка**

add\_row("Customers\_DCompanies", {'Customer\_ID': 1, 'DC\_ID': 1})



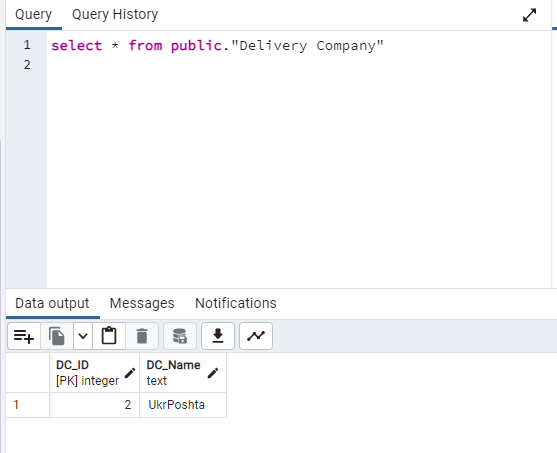
**Видалення**

del\_row('Customers\_DCompanies', 'DC\_ID', '1')



**Редагування**

edit\_value("Delivery Company", "DC\_ID", "DC\_Name", "UkrPoshta", 2)



**Завдання №2**

В даному завданні розглянемо використання індексів GIN, BRIN.

Для тестування була створена таблиця з типом даних tsvector, щоб можна було працювати з індексом GIN:

**CREATE TABLE public.test**

**(**

**ID serial PRIMARY KEY,**

**vect tsvector,**

**txt text COLLATE pg\_catalog.”default”,**

**);**

Далі було згенеровано випадкові дані в таблицю:

**INSERT INTO test(txt, vect)**

**(**

**SELECT**

**chr(trunc(60 + random() \* 25)::int) || chr(trunc(60 + random() \* 50)::int) || chr(trunc(60 + random() \* 50)::int),**

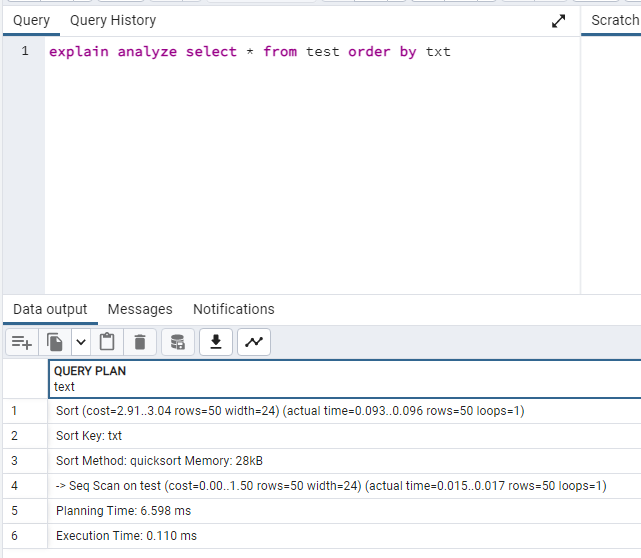
**to\_tsvector(chr(trunc(60 + random() \* 25)::int) || chr(trunc(60 + random() \* 50)::int) || chr(trunc(60 + random() \* 50)::int))**

**FROM generate\_series(1,50)**

**);**

Посортуємо дані без використання індексу:

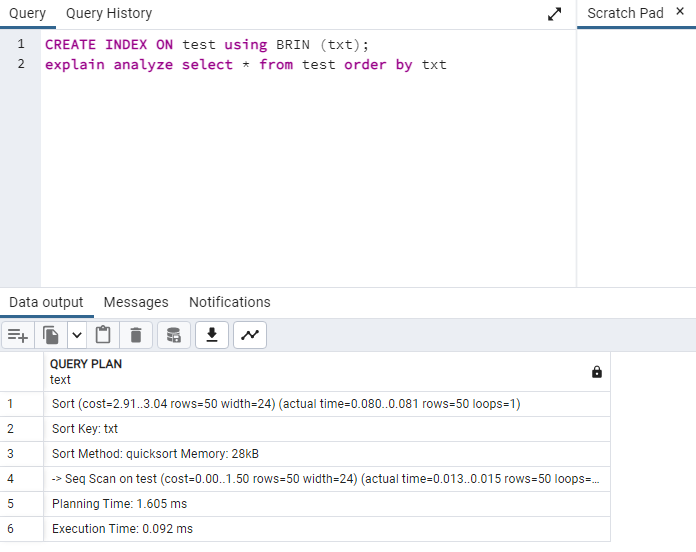
**explain analyze select \* from test order by txt**



Отже час сортування 110 мс, далі виконаємо сортування використовуючи індекс BRIN:

**create index on test using BRIN (txt);**

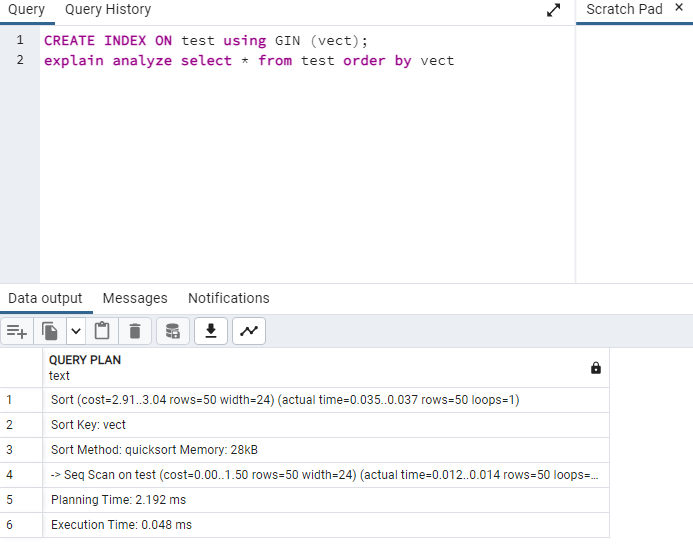
**explain analyze select \* from test order by vect;**



І сортування з індексом GIN:

**create index on test using GIN (vect);**

**explain analyze select \* from test order by vect;**



Як видно з тестів, індекс GIN зменшив час сортування майже в 2,5 рази, що є дуже хорошим результатом. Даний індекс створений для повнотекстового пошуку, тому дуже добре оптимізований. Індекс BRIN теж зменшив час сортування, але на 18 мс. Оскільки індекс BRIN призначений для обробки дуже великих таблиць, то для такої таблиці як в нас, він теж показав гарний результат.

**Завдання №3**

Для тестування тригера було створено дві таблиці:

**DROP TABLE IF EXISTS "trigger";**

**CREATE TABLE "trigger"(**

**"ID" bigserial PRIMARY KEY,**

**"Name" text**

**);**

**DROP TABLE IF EXISTS "triggerLog";**

**CREATE TABLE "triggerLog"(**

**"LogID" bigserial PRIMARY KEY,**

**"TriggerID" bigint,**

**"TriggerName" text**

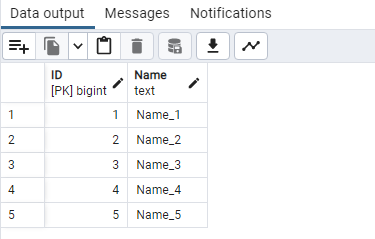
**);**

І в таблицю "trigger" занесено значення:

**insert into public."trigger" ("Name")**

**VALUES ('Name\_1'), ('Name\_2'), ('Name\_3'),**

**('Name\_4'), ('Name\_5')**



Код тригера:

**CREATE OR REPLACE FUNCTION update\_delete\_func() RETURNS TRIGGER as $$**

**BEGIN**

**IF new."Name" = 'ERROR' THEN**

**RAISE EXCEPTION 'NOT CORRECT NAME';**

**ELSE**

**IF old."ID" % 2 = 0 THEN**

**INSERT INTO "triggerLog"("TriggerID", "TriggerName")**

**VALUES (old."ID", old."Name");**

**RETURN NEW;**

**ELSE**

**INSERT INTO "triggerLog"("TriggerID", "TriggerName")**

**VALUES (old."ID", old."Name" || '\_odd');**

**RETURN NEW;**

**END IF;**

**END IF;**

**END;**

**$$ language plpgsql;**

Ініціалізація тригера:

**drop trigger if exists test\_trigger ON public."trigger";**

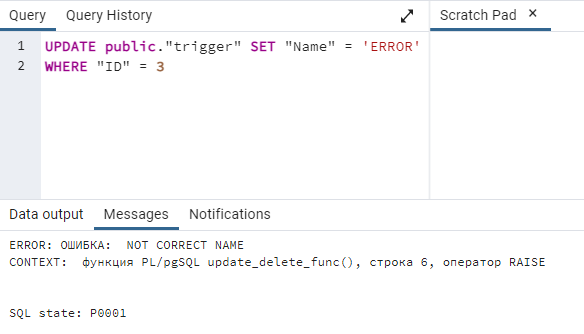
**create trigger test\_trigger before update or delete**

**on public."trigger" for each row**

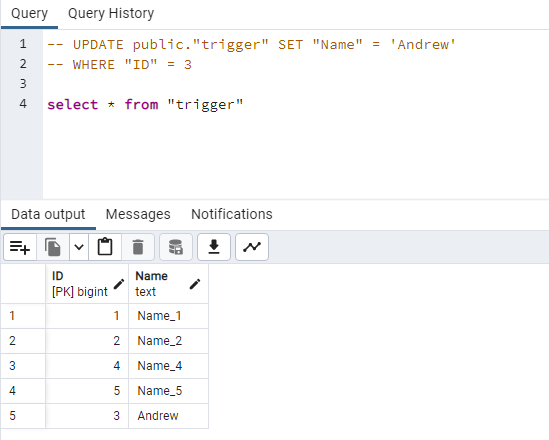
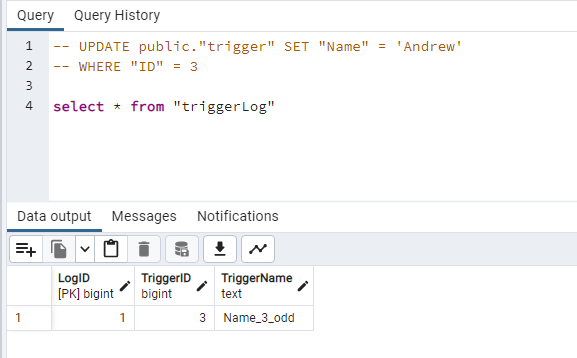
**execute procedure update\_delete\_func();**

Перевірка тригера

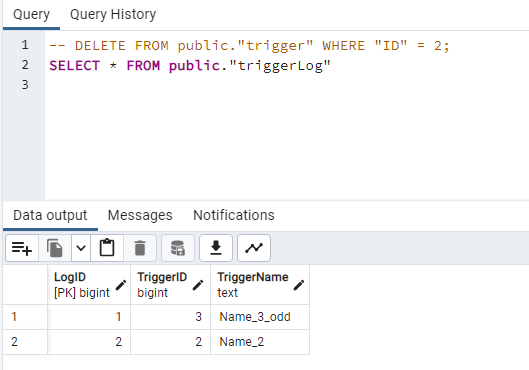
1. Якщо при зміненні значення в таблиці"trigger", нове значення містить заборонене слово, то програма видає помилку *“NOT CORRECT NAME”*



1. При зміні або видалені значення в "trigger", попереднє значення записується в таблицю "triggerLog". Якщо значення знаходиться під непарним “ID”, то до значення в таблиці "triggerLog" додається ‘\_odd’



На цих знімках видно, що при зміні значення під непарним “ID”, значення в таблиці "triggerLog" записано з припискою ‘\_odd’



А на цьому знімкові можна побачити, що при видалені рядка під парним “ID”, значення в таблиці "triggerLog" записано в початковому вигляді.