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 та засоби управління»**

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

Виконав: студент ІІI курсу

ФПМ групи КВ-11

Дзуман В.

Київ – 2024

*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.

Git-репозиторій: [Rak-vojak/DB\_lab2 (github.com)](https://github.com/Rak-vojak/DB_lab2)

*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*

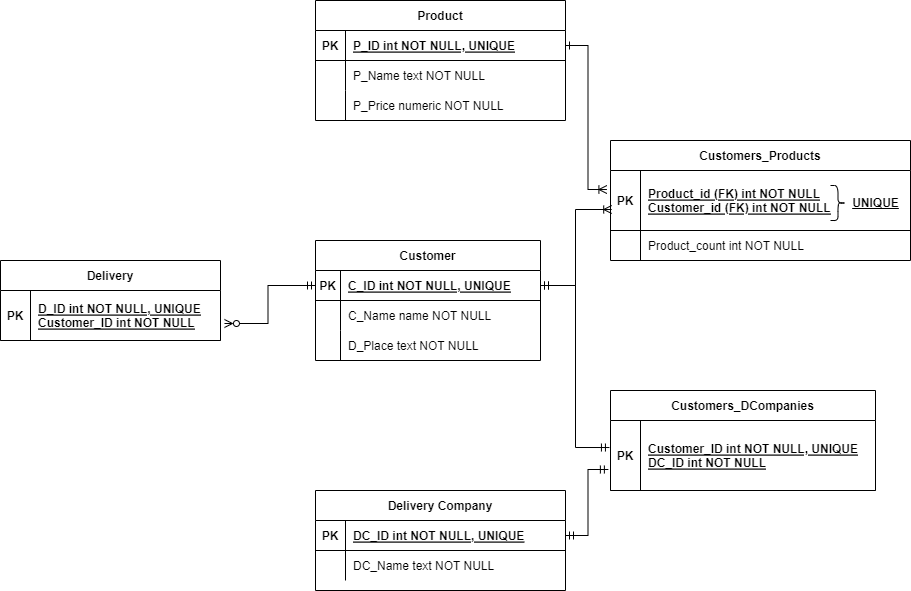
Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування 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).

**Варіант 6**

|  |  |  |
| --- | --- | --- |
| *6* | *BTree, BRIN* | *after update, insert* |

Бібліотека для реалізації ORM - SQLAlchemy

**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)

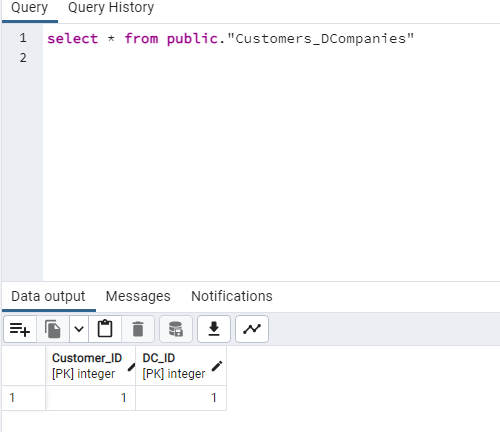
s.rollback()

**return** False

**Приклади виконання запитів**

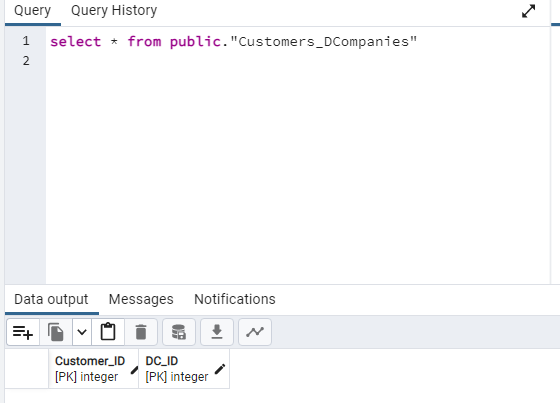
**Вставка**

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



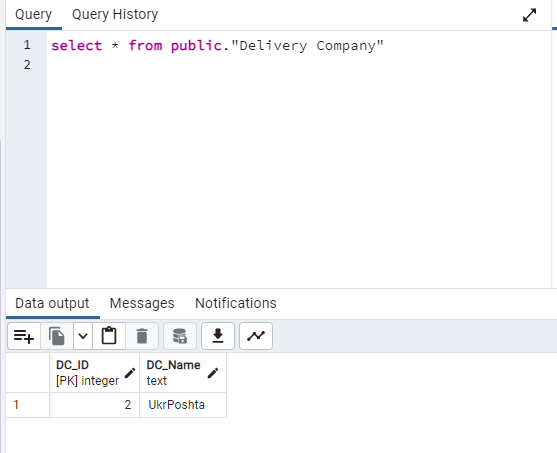
**Видалення**

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 *BTree, BRIN*

**BTree**

BTree індекси найкраще підходять для операцій порівняння, таких як **=**, **<**, **<=**, **>** та **>=**.

**Таблиця Customer**:

* C\_ID вже є первинним ключем, отже, він автоматично використовує BTree індекс.
* C\_Name може мати BTree індекс, якщо часто виконуються запити, які фільтрують або сортують дані за іменем клієнта.

CREATE INDEX idx\_customer\_name ON Customer USING BTREE (C\_Name);

**Таблиця Product**:

* P\_ID є первинним ключем, тож він вже має BTree індекс.
* P\_Price може отримати користь від BTree індексу для запитів, що включають фільтрацію або сортування за ціною.

CREATE INDEX idx\_product\_price ON Product USING BTREE (P\_Price);

**Приклади Запитів**

**Фільтруючий Запит на Customer**

SELECT \* FROM Customer WHERE D\_Place = 'Київ';

**Агрегатна Функція на Product**

SELECT AVG(P\_Price) FROM Product;

**BRIN Індекс**

BRIN індекси ідеально підходять для великих таблиць з впорядкованими або частково впорядкованими даними.

**Таблиця Delivery**:

* Якщо D\_ID вставляється послідовно і таблиця є великою, BRIN індекс може бути дуже ефективним.

CREATE INDEX idx\_delivery\_brin ON Delivery USING BRIN (D\_ID);

**Таблиця Delivery Company**:

* Аналогічно, якщо DC\_ID вставляється послідовно і таблиця є великою, BRIN індекс може бути корисним

CREATE INDEX idx\_delivery\_company\_brin ON Delivery\_Company USING BRIN (DC\_ID);

**Приклад Запиту**

**Складний Запит із JOIN**

SELECT d.\*, dc.DC\_Name FROM Delivery d

JOIN Customer c ON c.C\_ID = d.Customer\_ID

JOIN Delivery\_Company dc ON dc.DC\_ID = d.DC\_ID

WHERE c.C\_Name = 'John Doe'

ORDER BY d.D\_ID;

BRIN індекси на **D\_ID** та **DC\_ID** можуть прискорити цей запит, особливо якщо таблиці **Delivery** та **Delivery Company** є великими та їх ID вставляються послідовно.

Певні обмеження:

Під час використання індексів слід бути обережним, адже вони збільшують час, необхідний для виконання операцій вставки, оновлення та видалення через необхідність підтримки індексів. Також, для маленьких таблиць або стовпців з низькою унікальністю значень, використання індексів може не приносити суттєвої користі.

Top of Form