

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»**

**Тема:** «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав:  
студент групи КВ-84  
Вазерцев Д  
Перевірив:  
Петрашенко А.В.

Київ 2020

*Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL*

## **Завдання**

*Завдання роботи полягає у наступному:*

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

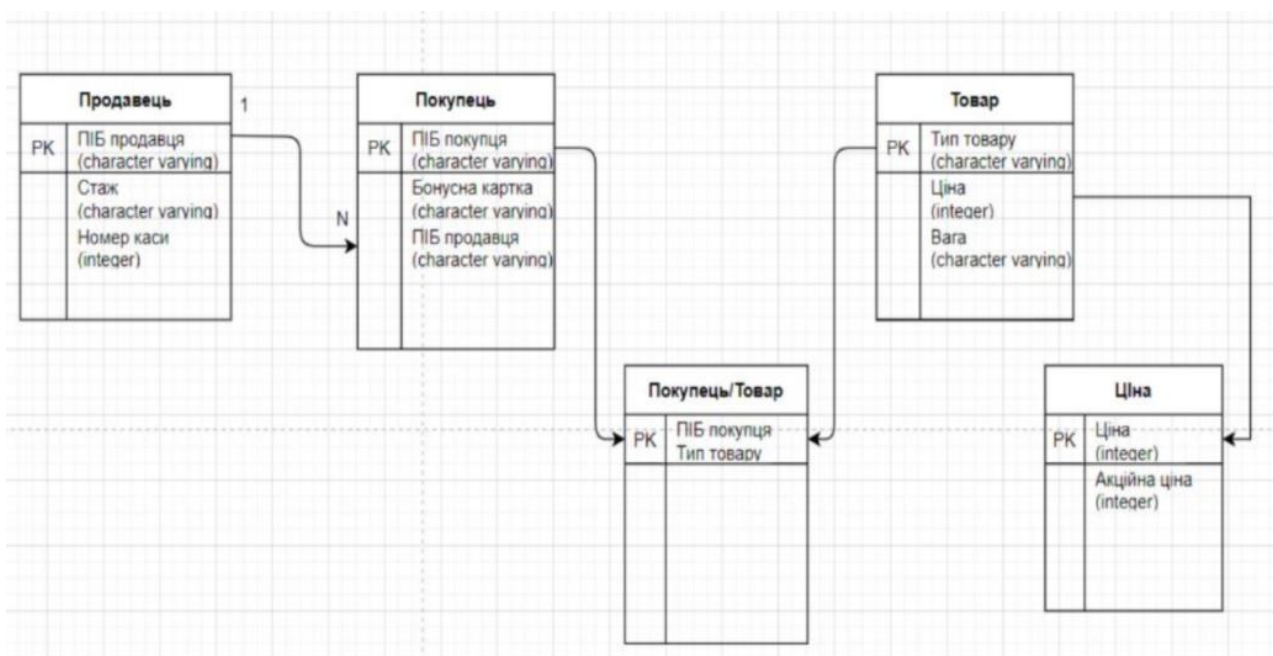
### *Вимоги до пункту завдання №2*

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.

7	<i>GIN, BRIN</i>	<i>before insert, delete</i>
---	------------------	------------------------------



# Завдання 1

## Класи сутності

```
class Seller(Base):
    __tablename__ = 'seller'
    seller_data = Column(String, primary_key = True)
    experience = Column(String)
    cash_register_num = Column(Integer)
    def __init__(self, seller_data, experience, cash_register_num):
        self.seller_data = seller_data
        self.experience = experience
        self.cash_register_num = cash_register_num

class Shopper(Base):
    __tablename__ = 'shopper'
    shopper_data = Column(String, primary_key = True)
    bonus_card = Column(String)
    seller_data = Column(String)
    def __init__(self, shopper_data, bonus_card, seller_data):
        self.shopper_data = shopper_data
        self.bonus_card = bonus_card
        self.seller_data = seller_data

class Product(Base):
    __tablename__ = 'product'
    product_type = Column(String, primary_key = True)
    price = Column(Integer)
    weight = Column(String)
    def __init__(self, product_type, price, weight):
        self.product_type = product_type
        self.price = price
        self.weight = weight

class Price(Base):
    __tablename__ = 'price'
    price = Column(Integer, ForeignKey('product.price'), primary_key = True)
    new_price = Column(Integer)
    def __init__(self, price, new_price):
        self.price = price
        self.new_price = new_price
```

## Функції Insert, Delete, Update

```
def delete(self, table, key):
    try:
        return table.delete(self.session).where(key)
    except Exception as err:
        print(err)
```

```
def insert(self, table, values):
    try:
        keys = [x.lstrip("!") if x.startswith("!") else "{}".format(x) for x in values]
        return table.insert(self.session, values = keys)
    except Exception as err:
        print(err)
```

```
def update(self,table,relay,values):
    try:
        value = [x.lstrip("!") if x.startswith("!") else "{}".format(x) for x in values]
        return table.update(self.sesion).where(relay).values(value)
    except Exception as err:
        print(err)
```

## Завдання 2

### GIN

Без індексу

Query Editor   Query History

```
1 explain analyze select * from test where column_1 = 'zbit:1'
```

Data Output   Explain   Messages

	<b>QUERY PLAN</b> text	
1	Seq Scan on test (cost=0.00..25.00 rows=6 width=40) (actual time=0.062..0.063 rows=1 loops=1)	
2	Filter: (column_1 = "zbit":1::tsvector)	
3	Rows Removed by Filter: 8	
4	Planning Time: 0.057 ms	
5	Execution Time: 0.082 ms	

З індексом

Query Editor
Query History

```

1 create index my_index_column on test using gin("column_1")
2 explain analyze select * from test where column_1 = 'zbit:1'

```

Data Output
Explain
Messages

CREATE INDEX

Query returned successfully in 79 msec.

1
2

```

create index my_index_column on test using gin("column_1")
explain analyze select * from test where column_1 = 'zbit:1'

```

Data Output
Explain
Messages

	QUERY PLAN	
	text	🔒
1	Seq Scan on test (cost=0.00..1.11 rows=1 width=40) (actual time=0.025..0.026 rows=1 loops=1)	
2	Filter: (column_1 = "zbit":1'::tsvector)	
3	Rows Removed by Filter: 8	
4	Planning Time: 3.055 ms	
5	Execution Time: 0.045 ms	

Індекс GIN покращив пошук, його основною задачею є прискорення повнотекстового пошуку. GIN добре підходить для даних, які не часто оновлюються. Для таблиць де зберігаються часто змінні дані не рекомендовано використовувати цей індекс, бо переіндексація може займати багато часу.

BRIN

## Без індексу

Query Editor

Query History

1

explain analyse select \* from test where "column\_2" = '20:30:31.271692'

Data Output

Explain

Messages

QUERY PLAN

text

1

Seq Scan on test (cost=0.00..1.11 rows=1 width=40) (actual time=0.011..0.012 rows=1 loops=1)

2

Filter: (column\_2 = '20:30:31.271692':time without time zone)

3

Rows Removed by Filter: 8

4

Planning Time: 0.188 ms

5

Execution Time: 0.024 ms

## З індексом

Query Editor

Query History

1

create index my\_index\_for\_column2 on test using brin("column\_2")

2

explain analyse select \* from test where "column\_2" = '20:30:31.271692'

Data Output

Explain

Messages

CREATE INDEX

Query returned successfully in 72 msec.

1

create index my\_index\_for\_column2 on test using brin("column\_2")

2

explain analyse select \* from test where "column\_2" = '20:30:31.271692'

Data Output

Explain

Messages

QUERY PLAN

text

1

Seq Scan on test (cost=0.00..1.11 rows=1 width=40) (actual time=0.016..0.017 rows=1 loops=1)

2

Filter: (column\_2 = '20:30:31.271692':time without time zone)

3

Rows Removed by Filter: 8

4

Planning Time: 1.269 ms

5

Execution Time: 0.031 ms

Спостерігаємо те, що застосування індексу BRIN пришвидщило пошук заданого значення.

BRIN відмінно працює для стовпців, значення яких корелюють з їх фізичним розташуванням в таблиці.

BRIN в основному призначений для таблиць великих і навіть величезних розмірів, які або взагалі не оновлюються, або оновлюються дуже незначно.

## Завдання 3

### Код тригера:

```
func()
1 begin
2   if (TG_OP = 'DELETE') then
3     declare
4       del_cur cursor for
5       select * from trigger_test where old.count1 = any(count2::int[]);
6       begin
7         for record in del_cur loop
8           begin
9             update trigger_test set count2 = array_remove(count2,old.count1) where count1 = record.count1;
10          end;
11        end loop;
12      end;
13      return old;
14    elsif (TG_OP = 'INSERT') then
15      begin
16        if ((select count(*) from trigger_test where count1 = any(new.count2::int[])) != array_length(new.count2::int[],1))
17          raise exception 'error';
18        end if;
19        return new;
20      end;
21    end if;
22 end
```

→ Create - Trigger

General Definition Events Transition Code SQL

Row trigger?

Yes

Constraint trigger?

No

Deferrable?

No

Deferred?

No

Trigger function

public.func



→ Create - Trigger

×

General

Definition

Events

Transition

Code

SQL

Fires

BEFORE

Events

INSERT

Yes

UPDATE

No

DELETE

Yes

TRUNCATE

No

→ Create - Trigger

General

Definition

Events

Transition

Code

SQL

```

1 CREATE TRIGGER my_trgr
2   BEFORE INSERT OR DELETE
3   ON public.trigger_test
4   FOR EACH ROW
5   EXECUTE PROCEDURE public.func();

```

	Data Output	Explain	Mes:
	count1 integer	count2 integer[]	
1	31	{32,32}	
2	32	{33}	
3	33	{32}	
4	37	[null]	

**Перевірка роботи:**

Query Editor   Query History

```
1  INSERT INTO public.trigger_test(  
2      count1, count2)  
3      VALUES (38, '{null}');
```

Data Output   Explain   Messages   Notifications

ERROR: ПОМИЛКА: error

CONTEXT: Функція PL/pgSQL func() рядок 18 в RAISE

SQL state: P0001

Query Editor   Query History

```
1  INSERT INTO public.trigger_test(  
2      count1, count2)  
3      VALUES (40, '{31,32}');
```

Data Output   Explain   Messages   Notifications

INSERT 0 1

Query returned successfully in 50 msec.

	Data Output	Explain	Mess
	count1 integer	count2 integer[]	
1	31	{32,32}	
2	32	{33}	
3	33	{32}	
4	37	[null]	
5	40	{31,32}	

Query Editor    Query History




```
1 delete from trigger_test where count1 = 32
```

Data Output    Explain    Messages    Notifications

DELETE 1

Query returned successfully in 269 msec.

Data Output Explain Message

	 <b>count1</b> integer		<b>count2</b> integer[]	
1		37		[null]
2		31		{}
3		33		{}
4		40		{31}