



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 3 курсу
ФПМ групи КВ-84

Байдаус Михайло
Перевірів:

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 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

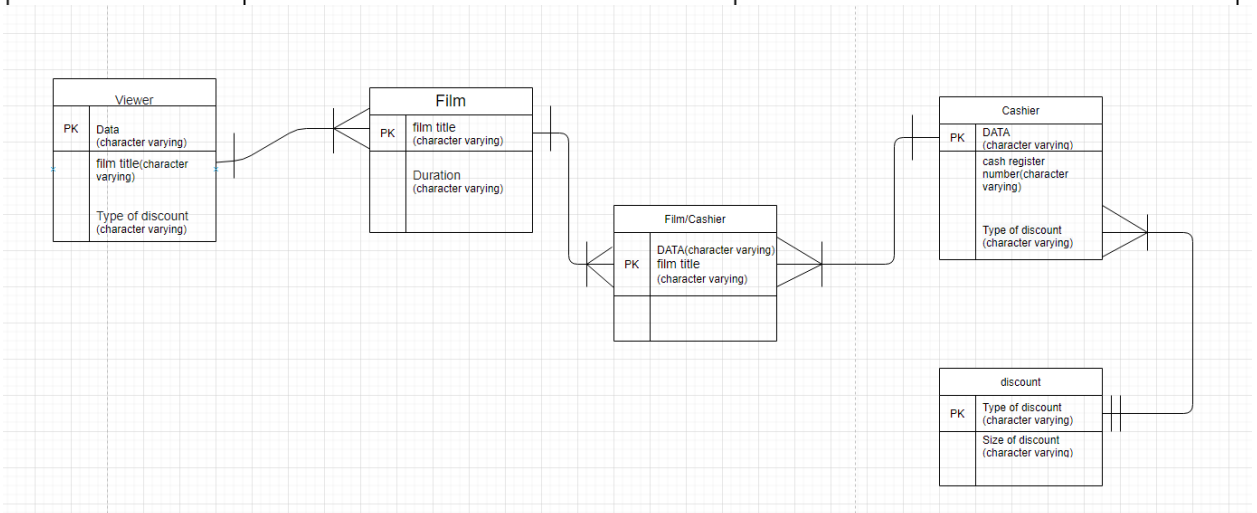
Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування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

3	GIN, Hash	<i>before delete, update</i>
---	-----------	------------------------------



Модель бази даних

Завдання 1

```

import psycopg2
import sqlalchemy
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker
from sqlalchemy import Column, String, Integer, ForeignKey
  
```

```

DATABASE_URI = 'postgres+psycopg2://postgres:localhqwerty121@ost:5432/MyData'
engine = create_engine(DATABASE_URI)
Session = sessionmaker(bind=engine)
  
```

```

Base = declarative_base()
  
```

Класи сутності

```
film/cashier = Table('film/cashier',Base.metadata,Column('film_title',String,ForeignKey('film.film_title')),Column('data1',String,ForeignKey('cashier.data1'))))

class Cashier(Base):
    __tablename__ = 'cashier'
    data1 = Column(String,primary_key = True)
    cash_register_number = Column(String)
    type_of_discount = Column(String,ForeignKey('discount.type_of_discount'))
    def __init__(self,data1,cash_register_number,type_of_discount):
        self.data1 = data1
        self.cash_register_number = cash_register_number
        self.type_of_discount = type_of_discount
```

```
class Discount(Base):
    __tablename__ = 'discount'
    type_of_discount = Column(String,primary_key = True)
    size_of_discount = Column(String)
    def __init__(self,type_of_discount,size_of_discount):
        self.type_of_discount=type_of_discount
        self.size_of_discount=size_of_discount
```

```
class Film(Base):
    __tablename__ = 'film'
    film_title = Column(String,primary_key = True)
    duration = Column(String)
    def __init__(self,film_title,duration):
        self.film_title = film_title
        self.duration = duration
```

```
class Viewer(Base):
    __tablename__ = 'viewer'
    data = Column(String,primary_key = True)
    film_title = Column(String,ForeignKey('film.film_title'))
    def __init__(self,data,film_title):
        self.data = data
        self.film_title = film_title
```

Функції Update,Delete, Insert

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

```
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)
```

Завдання 2

Створення та аналіз індексів GIN та Hash

GIN - це Generalized Inverted Index, або обернений індекс. Його основною задачею є прискорення повнотекстового пошуку.




HASH – це режим індексу, який автоматично застосовує хеш-функцію, до даних індексу. Хоча хешування і займає додатковий час, при великій кількості даних, це може значно прискорити виконання запитів, через те, що час доступу хеш-таблиці менший ніж у звичайних колекцій.

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"

	 num integer	 val character varying (50)	
1		60	4ce3387c0261c89c4e9b5307cbcc0118
2		12	46655bf447e64e750865915531460d1c
3		44	037c9fe14eb1f6b2af4ac41020cb4952
4		74	a2b697ded604cc3131b179bdacd64eaa
5		80	40e5378e352c1745428b6129b05bb7da
6		16	db2bf445c3a24ba7ed3ec92bd10474c6
7		62	5ff3f8beadc02173bacaa9d3b58900d5
8		51	78b69c31681eeab1059c387f7aa9843b
9		25	9a06358c7f7758c10d535eabbd8eb336
10		32	ba0371bf5fac833c2763d697dd1a7013

Запит №1

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

Без індексування.

	num integer	val character varying (50)
1	20	00008bc9011032c001710c...
2	49	0000d580d25b45f0fe31f0d...
3	17	00016cf0b43925269fc335a...
4	12	0001890a3a3f43f606728d8...
5	65	0001f295de0c6d38f2a87fc...
6	19	000394c354c3dbb7432c1d...
7	54	0005d150bda23436f21872...
8	35	000608d07b42593e905f9a7...
9		
10		

✓ Successfully run. Total query runtime: 1 secs 16 msec. 100001 rows affected.

Hash

	num integer	val character varying (50)
1	20	00008bc9011032c001710c...
2	49	0000d580d25b45f0fe31f0d...
3	17	00016cf0b43925269fc335a...
4	12	0001890a3a3f43f606728d8...
5	65	0001f295de0c6d38f2a87fc...
6	19	000394c354c3dbb7432c1d...
7	54	0005d150bda23436f21872...
8	35	000608d07b42593e905f9a7...
9		
10		

✓ Successfully run. Total query runtime: 1 secs 120 msec. 100001 rows affected.

GIN

	num integer	val character varying (50)
1	20	00008bc9011032c001710c...
2	49	0000d580d25b45f0fe31f0d...
3	17	00016cf0b43925269fc335a...
4	12	0001890a3a3f43f606728d8...
5	65	0001f295de0c6d38f2a87fc...
6	19	000394c354c3dbb7432c1d...
7	54	0005d150bda23436f21872...
8		

✓ Successfully run. Total query runtime: 1 secs 834 msec. 100001 rows affected.

Заяпит №2

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

Без індексування

	num integer	val character varying (50)
1	55	Search text

✓ Successfully run. Total query runtime: 1 secs 659 msec. 1 rows affected.

Hash

	num integer	val character varying (50)
1	55	Search text

✓ Successfully run. Total query runtime: 1 secs 19 msec. 1 rows affected.

GIN

	num integer	val character varying (50)
1	55	Search text

✓ Successfully run. Total query runtime: 1 secs 776 msec. 1 rows affected.

Запит №3

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

Без індексування

	num integer	count bigint
1	55	988
2	27	967
3	23	992
4	56	996
5	58	1020
6	91	1076
7	8	1018
8	87	9

✓ Successfully run. Total query runtime: 611 msec. 100 rows affected.

Hash

	num integer	count bigint
1	55	988
2	27	967
3	23	992
4	56	996
5	58	1020
6	91	1076
7	8	1018
8	87	

✓ Successfully run. Total query runtime: 1 secs 226 msec. 100 rows affected.

Запит №4

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

Без індексування

	count bigint
1	687

✓ Successfully run. Total query runtime: 1 secs 429 msec. 1 rows affected.

Hash

	count bigint
1	687

✓ Successfully run. Total query runtime: 1 secs 164 msec. 1 rows affected.

GIN

	count bigint
1	687

✓ Successfully run. Total query runtime: 731 msec. 1 rows affected.

GIN зручно використовувати коли необхідний пошук символів у тексті. Основними недоліками є неможливість використання наприклад числових типів даних і довге створення, переіндексація. GIN добре підходить для даних, які не часто оновлюються і коли в запиті присутні такі операції як LIKE.

Індекс Hash зручно використовувати коли необхідно прискорити пошук і також важливим є розмір індексу. Хешування індексу ефективне, для виконання сортування, групування або пошуку даних), проте воно неефективне, для прямої роботи з даними

,також, хешування займає значний час, тому цей спосіб буде ефективний, лише при великій кількості даних. Nash краще працює з числами ніж з рядками символів.

Завдання 3

Необхідно створити тригери before delete та before update.

Для трегера створено таблицю test

	id bigint	title character varying	capacity integer	city character varying
1	1	Dadd	500000	India
2	54	Awf	1500	Ukraine
3	100	AWDS	80000	Africa
4	20000	HBfdg	225	USA
5	1245	kljk	10	France
6	332	wersdf	120000	Poland
7	4456	erur	8000	Germany
8	23	frter	12000	Belarus
9	886	cvnnm	54	Turkey
10	500001	catttt	99851	England

test
Columns (4)
id
title
capacity
country

Тригер при видаленні рядка записує інформацію в таблицю logs.

logs
Columns (3)
id
log
deletedat

Тригер

```
Query Editor  Query History
1 create or replace function test_trigger() returns trigger as $$
2 begin
3     if (tg_op = 'DELETE') then
4         insert into logs(log, deletedat) values ('deleted from table test', now()::timestamp);
5         raise notice 'Delete Successfull';
6         returning null;
7     elseif tg_op = 'UPDATE' then
8         if (new.capacity < 90000) then
9             insert into logs(log, deletedat) values('incorrect data during updating a row in table test', now()::timestamp);
10            raise exception 'Capacity must be more than 90000';
11            returning null;
12        end if;
13        insert into logs(log, deletedat) values('updated row in test table', now()::timestamp);
14        return new;
15    else return null;
16    end if;
17 end;
18 $$language plpgsql;
19
20 create trigger test_trigger before delete or update on public.test
21 for each row execute procedure test_trigger();
```

Якщо спрацював тригер delete ми записуємо в таблицю logs повідомлення та дату видалення. В випадку update тригера, виконується перевірка, щоб в оновленого рядка значення capacity було більшим за 90 тисяч, якщо воно менше то виникає помилка з відповідним повідомленням.

Розглянемо випадки роботи тригера.

Для випадку видалення:

```
delete from test where id = 886;  
select * from logs;
```

Output	Explain	Messages	Notifications
log text		deletedat timestamp with time zone	
deleted from table test		2020-12-21 00:40:02.722944+03	

При випадку оновлення рядку:

Коли дані коректні:

```
update test set capacity = 99000 where id = 20000;  
select * from logs;
```

ut	Explain	Messages	Notifications
log text		deletedat timestamp with time zone	
deleted from table test		2020-12-21 00:40:02.722944+03	
updated row in test table		2020-12-21 00:43:08.017378+03	

Коли capacity вводиться менше 90000

```
1 update test set capacity = 228 where id = 1;  
2 select * from logs;
```

Data Output	Explain	Messages	Notifications
ERROR: ОШИБКА: Capacity must be more than 90000 CONTEXT: функция PL/pgSQL test_trigger(), строка 10, оператор RAISE			