

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

Чернявський Максим Перевірив:

 $Mетою poботи \in 3$ добуття 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.

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 ϵ реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не ϵ обов'язковою.

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

Вимоги до пункту завдання №2

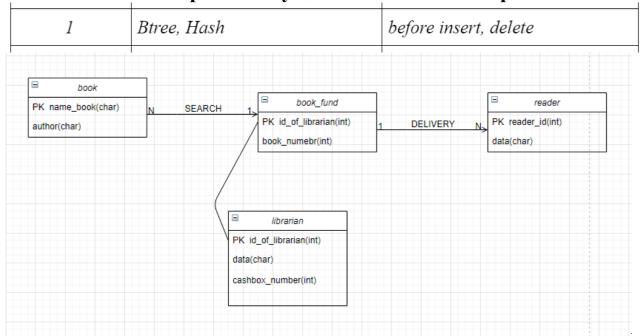
Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування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

По заліковій 28 варіант тому циклічно взяв 1-й варіант



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:localh123123qwert@ost:5432/MyData'
engine = create_engine(DATABASE_URI)
Session = sessionmaker(bind=engine)

Base = declarative_base()
```

Класи сутності

```
class Book(Base):
    __tablename__ = 'book'
    name_book = Column(String,primary_key = True)
    author = Column(String)
    def __init__(self,name_book,author):
        self.name_book = name_book
        self.author = author
```

```
class Book fund(Base):
              _tablename__ = 'book_fund'
             id_of_librarian = Column(Integer,primary_key = True)
             book_number = Column(Integer)
             def __init__(self,id_of_librarian,book_number):
                  self.id_of_librarian=id_of_librarian
                  self.book number=book number
class Librarian(Base):
     _tablename__ = 'librarian'
   id of librarian = Column(Integer, primary key = True, , ForeignKey('book fund.id of librarian'))
   data = Column(String)
   cashbox number = Column(Integer)
   def __init__(self,id_of_librarian,data,cashbox_number):
      self.id_of_librarian = id_of_librarian
       self.data = data
       self.cashbox_number = cashbox_number
            class Reader(Base):
                __tablename__ = 'reader'
                reader_id = Column(Integer,primary_key = True)
                data = Column(String)
                def __init__(self,reader_id,data):
                     self.reader_id = reader_id
                     self.data = data
```

Функції Update ,Insert , Delete

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

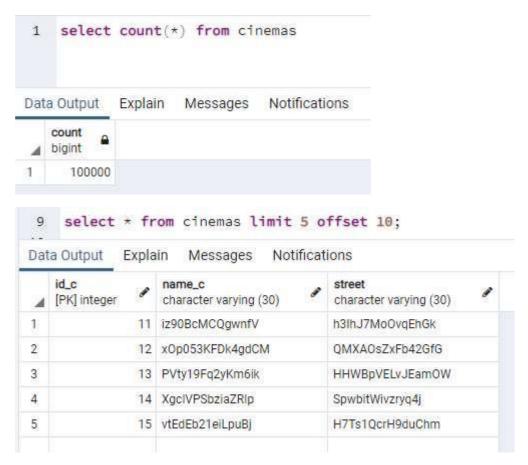
def Insert(self,table,values):
    try:
        keys = [x.lstrip("!") if x.starswitch("!") else "'{}'".format(x) for x in values]
        return table.insert(self.session,values = keys)
        except Exception as err:
        print(err)

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

Завдання 2 Створення та аналіз індексів Btree та Hash

Btree

Створимо 100000 випадкових рядків в таблиці cinemas(id_c, name_c,street).



Виконаємо запит без індексу по стовпчику name_c запису 'iz90BcMCQgwnfV'



Створимо індекс "cinemas btree" для таблиці "cinemas" по стовпчику "name c":

```
drop index if exists "cinemas_btree";

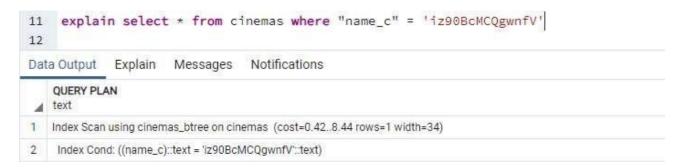
create index "cinemas_btree" on "cinemas" using btree("name_c");

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 1 secs 173 msec.
```

Виконаємо пошук 'iz90BcMCQgwnfV' знову.

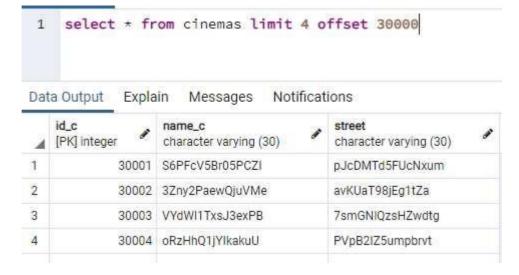


Бачимо, що пошук з індексом працює набагато швидше.

Hash

Для цього завдання знову використаємо таблицю cinemas(id_c, name_c,street). Слід зазначити, що індекс hash найкраще підходить для пошуку з використанням порівняння на "=". Візьмем стовпчик street.

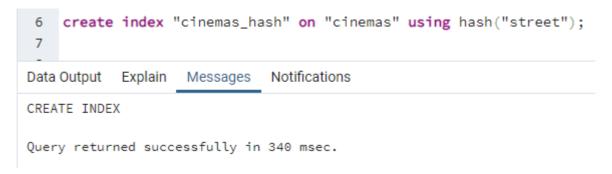




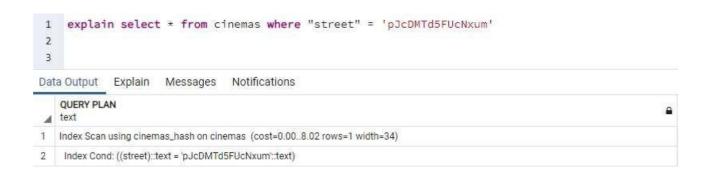
Виберемо одне з значень — "pJcDMTd5FUcNxum". Зробимо пошук по цьому імені.



Створимо індекс "cinemas hash" для таблиці "cinemas" по стовпчику "street":



Виконаємо пошук "pJcDMTd5FUcNxum" знову.



Подивившись на результати пошуку, можна сказати, що використання індексу дало досить значне підвищення швидкодії.

Завдання 3

before insert, delete

Для трегера створено таблицю table_test

Data Output		Expla	in Messages	Notifications	
4	usr_id [PK] integer	Ø.	usr_data character varying	Ø.	
1		123	data1		
2		324	data2		
3		351	data3		
4		563	[null]		
5		795	data4		

Тригер:

```
1 CREATE OR REPLACE FUNCTION trig() RETURNS trigger
2
       LANGUAGE plpgsql
       AS $$
3
4 ▼ BEGIN
    IF (TG_OP = 'INSERT') THEN
6 ₹
       IF NEW.usr_id < 0 THEN</pre>
         RAISE EXCEPTION 'NOT CORRECT NUMBER(<0). SHOULD NOT BE INSERT';
7
8
      END IF;
9 ₩
      IF NEW.usr_data is NULL THEN
         RAISE EXCEPTION 'NOT CORRECT NAME(NULL). SHOULD NOT BE INSERT';
10
      END IF;
11
      RETURN NEW;
12
    ELSIF (TG_OP = 'DELETE') THEN
13
      IF NEW.usr_id < 0 THEN</pre>
         RAISE EXCEPTION 'NOT CORRECT NUMBER(<0). SHOULD NOT BE DELETED';
15
16
       END IF;
      IF NEW.usr_data is NULL THEN
         RAISE EXCEPTION 'NOT CORRECT NAME (NULL). SHOULD NOT BE DELETED';
19 ▼
       END IF;
       RETURN OLD;
     END IF;
21
22
    RETURN NEW;
23 END;
24 $$;
25
26 CREATE TRIGGER trig BEFORE DELETE OR INSERT ON table_test
27 FOR EACH ROW EXECUTE FUNCTION trig();
```

Приклади використання тригеру

```
quoi j cano. quoi j i notoi j
   1 INSERT INTO table_test values (-1,'data7');
  Data Output Explain Messages Notifications
  ERROR: OWNEKA: NOT CORRECT NUMBER. SHOULD NOT BE INSERTED
  CONTEXT: функция PL/pgSQL trig(), строка 5, оператор RAISE
 1 INSERT INTO table_test values (100,NULL);
Data Output Explain Messages Notifications
ERROR: OWNEKA: NOT CORRECT NAME. SHOULD NOT BE INSERTED
CONTEXT: функция PL/pgSQL trig(), строка 8, оператор RAISE
1 DELETE FROM table_test where usr_id = 563;
Data Output Explain Messages Notifications
ERROR: OWNEKA: NOT CORRECT NAME(NULL). SHOULD NOT BE DELETED
CONTEXT: функция PL/pgSQL trig(), строка 16, оператор RAISE
```

```
1 INSERT INTO table_test values(228,'data228');
2 select * from table_test;
```

Dat	Data Output Ex		xplain Messages		Notifications	
4	usr_id [PK] integer		usr_data character varying	Ø.		
1	1	123 d	data1			
2	3	324 d	data2			
3	3	351 d	data3			
4	7	795 d	data4			
5	12	223 0	data10			
6	563		[null]			
7	2	228 0	data228			