



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

Групи КВ-84

Ніколайчук Данило

Перевірив:

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.

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:M, M:M 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.

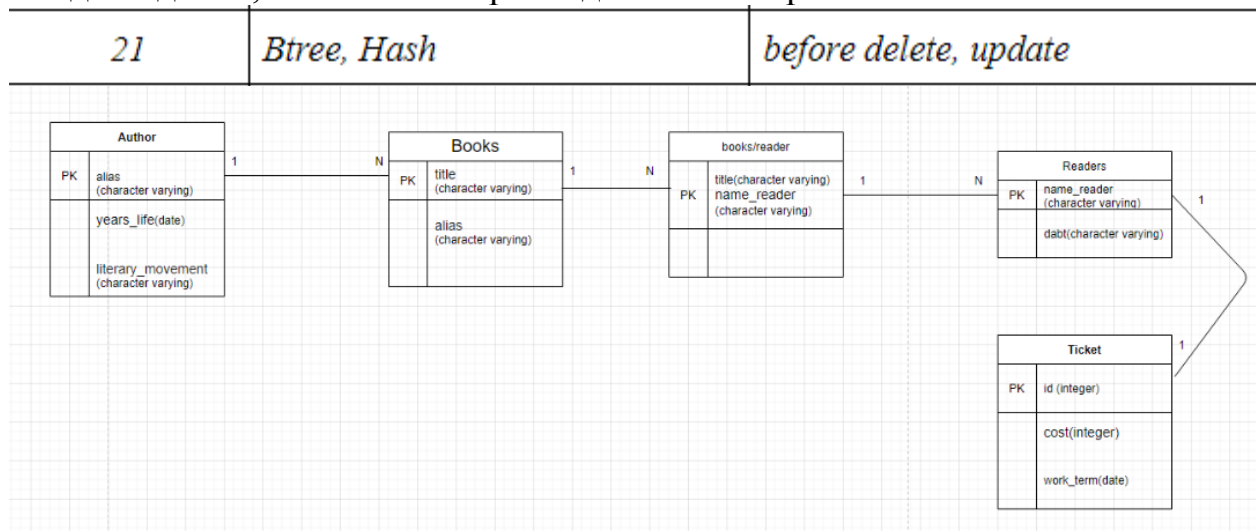
Вимоги до пункту завдання №2

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.



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:localhqwerty12345@ost:5432/LabData'
engine = create_engine(DATABASE_URI)
Session = sessionmaker(bind=engine)

Base = declarative_base()
  
```

Класи сутності

```

books_reader = Table('books_reader', Base.metadata, Column('title', String, ForeignKey('books.title')), Column('name_reader', String, ForeignKey('readers.name_reader')))

class Author(Base):
    __tablename__ = 'author'
    alias = Column(String, primary_key = True)
    years_life = Column(Date)
    literary_movement = Column(String)
    def __init__(self, alias, years_life, literary_movement):
        self.alias = alias
        self.years_life = years_life
        self.literary_movement = literary_movement
  
```

```

class Books(Base):
    __tablename__ = 'books'
    title = Column(String, primary_key = True)
    alias = Column(String, ForeignKey('author.alias'))
    def __init__(self, title, alias):
        self.title=title
        self.alias=alias
  
```

```
class Readers(Base):
    __tablename__ = 'readers'
    name_reader = Column(String, primary_key = True)
    debt = Column(String)
    def __init__(self, film_title, debt):
        self.name_reader = name_reader
        self.debt = debt
```

```
class Ticket(Base):
    __tablename__ = 'ticket'
    id = Column(Integer, primary_key = True)
    cost = Column(Integer)
    work_term = Column(Date)
    def __init__(self, id, cost, work_term):
        self.id = id
        self.cost = cost
        self.work_term = work_term
```

Функції Insert ,Update,Delete

```
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.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)
```

Завдання 2 Створення та аналіз індексів Btree та Hash

Btree

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

1 `select count(*) from cinemas`

Data Output Explain Messages Notifications

	count	
	bigint	
1	100000	

9

select * from cinemas limit 5 offset 10;

Data Output

Explain

Messages

Notifications

	id_c [PK] integer	name_c character varying (30)	street character varying (30)
1	11	iz90BcMCQgwnfV	h3lhJ7MoOvqEhGk
2	12	xOp053KFDk4gdCM	QMXAOsZxFb42GfG
3	13	PVty19Fq2yKm6ik	HHWBpVELvJEamOW
4	14	XgclVPSbziaZRlp	SpwbitWivzryq4j
5	15	vtEdEb21eiLpuBj	H7Ts1QcrH9duChm

Виконаємо запит без індексу по стовпчику name_c запису 'iz90BcMCQgwnfV'

11	<code>explain select * from cinemas where "name_c" = 'iz90BcMCQgwnfV'</code>
12	
<div>Data Output Explain Messages Notifications</div>	
<div>QUERY PLAN</div>	
<div>text</div>	
1	Seq Scan on cinemas (cost=0.00..2084.00 rows=1 width=34)
2	Filter: ((name_c)::text = 'iz90BcMCQgwnfV'::text)

Створимо індекс “cinemas_btree” для таблиці “cinemas” по стовпчику “name_c”:

13	<code>drop index if exists "cinemas_btree";</code>
14	<code>create index "cinemas_btree" on "cinemas" using btree("name_c");</code>
15	
16	
17	

Data Output	Explain	<u>Messages</u>	Notifications
-------------	---------	-----------------	---------------

CREATE INDEX

Query returned successfully in 1 secs 173 msec.

Виконаємо пошук 'iz90BcMCQgwnfV' знову.

```
11 explain select * from cinemas where "name_c" = 'iz90BcMCQgwnfV'
```

```
12
```

Data Output Explain Messages Notifications

QUERY PLAN	
text	
1	Index Scan using cinemas_btree on cinemas (cost=0.42..8.44 rows=1 width=34)
2	Index Cond: ((name_c)::text = 'iz90BcMCQgwnfV'::text)

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`.

```
1 select count(*) from cinemas
```

Data Output Explain Messages Notifications

count	
bigint	
1	100000

```
1 select * from cinemas limit 4 offset 30000
```

Data Output Explain Messages Notifications

	id_c [PK] integer	name_c character varying (30)	street character varying (30)
1	30001	S6PFcV5Br05PCZI	pJcDMTd5FUcNxum
2	30002	3Zny2PaewQjuVMe	avKUaT98jEg1tZa
3	30003	VYdWI1TxsJ3exPB	7smGNIQzsHZwdtg
4	30004	oRzHhQ1jYlkakuU	PVpB2IZ5umpbrvt

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

1	<code>explain select * from cinemas where "street" = 'pJcDMTd5FUcNxum'</code>
<div>Data Output Explain Messages Notifications</div>	
	<div>QUERY PLAN text</div>
1	Seq Scan on cinemas (cost=0.00..2084.00 rows=1 width=34)
2	Filter: ((street)::text = 'pJcDMTd5FUcNxum'::text)

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

6	<code>create index "cinemas_hash" on "cinemas" using hash("street");</code>
7	
-	
<div>Data Output Explain Messages Notifications</div>	
CREATE INDEX	
Query returned successfully in 340 msec.	

Виконаємо пошук "pJcDMTd5FUcNxum" знову.

1	<code>explain select * from cinemas where "street" = 'pJcDMTd5FUcNxum'</code>
2	
3	
<div>Data Output Explain Messages Notifications</div>	
	<div>QUERY PLAN text</div>
1	Index Scan using cinemas_hash on cinemas (cost=0.00..8.02 rows=1 width=34)
2	Index Cond: ((street)::text = 'pJcDMTd5FUcNxum'::text)

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

Завдання 3

Необхідно створити тригери before delete та before update.
Для трегера створено таблицю test_tr

	id [PK] integer	data character varying	size integer
1	1	ertertwrTERYGER	2000000
2	23	ASDWADA	800000
3	45	ghjghjgyjtyjvbn	50000
4	123	hfgghfrkj	123567
5	213	DFBfnvcx	256
6	233	36ryty45	12345
7	674	gdfgvbnfgjkcvcvbn	31235
8	1111	XCVfghjkaaaa	1000000
9	3321	dfikjhn	111111
10	4567	fghkbnm	23

Тригер при видаленні рядка записує інформацію в таблицю logs.

▼ logs
▼ Columns (3)
id
log
deletedat

Trigger :


```
1 create or replace function trigger_test() returns trigger as $$
2 begin
3     if (tg_op = 'DELETE') then
4         insert into logs(log, deletedat) values ('Success delete from table test_tr', now()::timestamp);
5         raise notice 'Delete Successfull';
6         return null;
7     elseif tg_op = 'UPDATE' then
8         if (new.capacity < 100000) then
9             insert into logs(log, deletedat) values('WRONG DATA when update a row in table test_tr', now()::timestamp);
10            raise exception 'Capacity must be more than 100000';
11            return null;
12        end if;
13        insert into logs(log, deletedat) values('Success Update from table test_tr', now()::timestamp);
14        return new;
15    else return null;
16    end if;
17 end;
18 $$language plpgsql;
19
20 create trigger trigger_test before delete or update on public.test_tr
21 for each row execute procedure trigger_test();
```

При роботі Delete в logs іде запис про дату видалення і час. При update виконується перевірка, щоб в оновленого рядка значення size було більшим за 100 тисяч, якщо воно менше то виникає помилка з повідомленням.

Випадки роботи тригера.

Для випадку DELETE:



```
DELETE FROM test_tr WHERE id = 1111;
SELECT * FROM logs;
```

tput	Explain	Messages	Notifications
log text			deletedat timestamp with time zone
Success delete from table test_tr			2020-12-22 23:01:08.462607+03

При UPDATE:

Якщо size > 100000 :

```
UPDATE test_tr SET size = 121000 WHERE id = 1;
SELECT * FROM logs;
```

ut	Explain	Messages	Notifications
log text			deletedat timestamp with time zone
Success delete from table test_tr			2020-12-22 23:01:08.462607+03
Success Update from table test_tr			2020-12-22 23:05:06.529916+03

Якщо size < 100000 (помилка):

```
1 UPDATE test_tr SET size = 1234 WHERE id = 1111;
2 SELECT * FROM logs;
```

Data Output	Explain	Messages	Notifications
ERROR: ОШИБКА: Capacity must be more than 100000 CONTEXT: функция PL/pgSQL trigger_test(), строка 10, оператор RAISE			