



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  
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 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 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

**По заліковій 28 варіант тому циклічно взяв 1-й варіант**

The diagram illustrates the following tables and their attributes:

- book**:
  - PK name\_book(char)
  - author(char)
- book\_fund**:
  - PK id\_of\_librarian(int)
  - book\_numebr(int)
- librarian**:
  - PK id\_of\_librarian(int)
  - data(char)
  - cashbox\_number(int)
- reader**:
  - PK reader\_id(int)
  - data(char)

The relationships are defined as follows:

- SEARCH**: A relationship between **book** and **book\_fund**. The cardinality is N:1.
- DELIVERY**: A relationship between **book\_fund** and **reader**. The cardinality is 1:N.

```
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()
```

```
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.startswith("!") else "{}".format(x) for x in values]
        return table.update(self.session).where(relay).values(value)
    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 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 explain select * from cinemas where "name_c" = 'iz90BcMCQgwnfV'
```

```
12
```

Data Output	Explain	Messages	Notifications
QUERY PLAN text			
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 drop index if exists "cinemas_btree";
```

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

```
15
```

```
16
```

```
17
```

Data Output	Explain	Messages	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

*before insert, delete*

Для трігера створено таблицю table\_test

	Data Output	Explain	Messages	Notifications
	usr_id [PK] integer		usr_data character varying	
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
3     AS $$
4 BEGIN
5     IF (TG_OP = 'INSERT') THEN
6         IF NEW.usr_id < 0 THEN
7             RAISE EXCEPTION 'NOT CORRECT NUMBER(<0). SHOULD NOT BE INSERT';
8         END IF;
9         IF NEW.usr_data IS NULL THEN
10            RAISE EXCEPTION 'NOT CORRECT NAME(NULL). SHOULD NOT BE INSERT';
11        END IF;
12        RETURN NEW;
13    ELSIF (TG_OP = 'DELETE') THEN
14        IF NEW.usr_id < 0 THEN
15            RAISE EXCEPTION 'NOT CORRECT NUMBER(<0). SHOULD NOT BE DELETED';
16        END IF;
17        IF NEW.usr_data IS NULL THEN
18            RAISE EXCEPTION 'NOT CORRECT NAME(NULL). SHOULD NOT BE DELETED';
19        END IF;
20        RETURN OLD;
21    END IF;
22    RETURN NEW;
23 END;
24 $$;
25
26 CREATE TRIGGER trig BEFORE DELETE OR INSERT ON table_test
27 FOR EACH ROW EXECUTE FUNCTION trig();
```



## Приклади використання триггеру

Query Editor Query History

```
1 INSERT INTO table_test values (-1,'data7');
```

Data Output Explain Messages Notifications

ERROR: ОШИБКА: 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: ОШИБКА: 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: ОШИБКА: 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;
```

Data Output Explain Messages Notifications

	 <b>usr_id</b> [PK] integer 	<b>usr_data</b> character varying 	
1	123	data1	
2	324	data2	
3	351	data3	
4	795	data4	
5	1223	data10	
6	563	[null]	
7	228	data228	