



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 III курсу

ФПМ групи КВ-84

Азиранкулов Є.А.

Перевірив:

Київ – 2020

*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, видалення 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 видалюється), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

*Вимоги до інструментарію*

1. Бібліотека для реалізації ORM - [SQLAlchemy для Python](#) або інша з подібною функціональністю.
2. Середовище для відлагодження SQL-запитів до бази даних – pgAdmin 4.
3. СУБД - PostgreSQL 11-12.

## Вимоги до оформлення лабораторної роботи у електронному вигляді

Опис та вміст репозиторію лабораторної роботи у **репозиторії GitHub** включає: оновлені файли додатку, назву лабораторної роботи та варіант студента.

Звіт лабораторної роботи має містити:

- a. титульний аркуш затвердженого зразка;
- b. завдання на лабораторну роботу з обов'язковим наведенням варіанту студента;
- c. копії екрану (скріншоти), що **підтверджують вимоги 1-3 завдання**, а також:
  - для завдання №1: схему бази даних у вигляді таблиць і зв'язків між ними, а також класи ORM і зв'язки між ними, що відповідають таблицям бази даних. Навести приклади запитів у вигляді ORM.
  - для завдання №2: команди створення індексів, тексти, результати і час виконання запитів SQL, пояснити чому індекси прискорюють (або не прискорюють) швидкість виконання запитів.
  - для завдання №3: команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних;

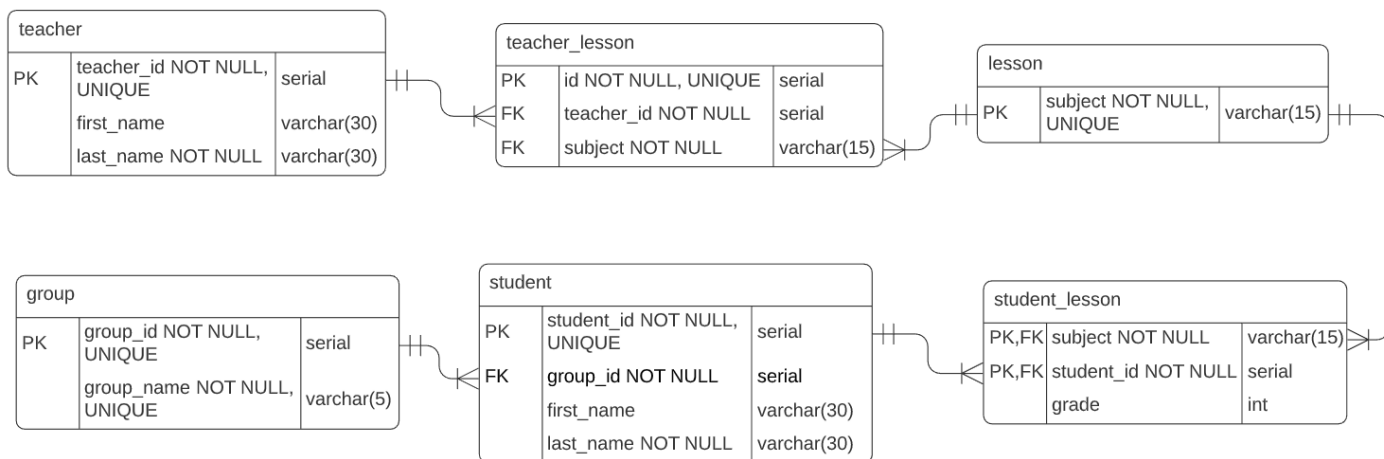
### Варіант 1

№ варіанта	Види індексів	Умови для тригера
1	Btree, Hash	before insert, delete

[GitHub](#) репозиторій

### Пункт №1

- Схема бази даних у вигляді таблиць і зв'язків між ними:



- Класи ORM:

```
Base = declarative_base()

class Group(Base):
    __tablename__ = 'group'
    group_id = Column(Integer, primary_key=True)
    group_name = Column(String)
    students = relationship('Student')

    def __repr__(self):...

class Student(Base):
    __tablename__ = 'student'
    student_id = Column(Integer, primary_key=True)
    group_id = Column(Integer, ForeignKey('group.group_id'))
    first_name = Column(String)
    last_name = Column(String)
    students_lessons = relationship('StudentLesson')

class Lesson(Base):
    __tablename__ = 'lesson'
    subject = Column(String, primary_key=True)
    students_lessons = relationship('StudentLesson')
    teachers_lessons = relationship('TeacherLesson')

class StudentLesson(Base):
    __tablename__ = 'student_lesson'
    student_id = Column(Integer, ForeignKey('student.student_id'), primary_key=True)
    subject = Column(Integer, ForeignKey('lesson.subject'), primary_key=True)
    grade = Column(Integer)

class Teacher(Base):
    __tablename__ = 'teacher'
    teacher_id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    teachers_lessons = relationship('TeacherLesson')

class TeacherLesson(Base):
    __tablename__ = 'teacher_lesson'
    id = Column(Integer, primary_key=True)
    teacher_id = Column(Integer, ForeignKey('teacher.teacher_id'))
    subject = Column(Integer, ForeignKey('lesson.subject'))
```

- Приклади запитів:

1. вставки:

```
def insert_group(values: tuple):  
    try:  
        group = Group(group_id=values[0], group_name=values[1])  
        s.add(group)  
        s.commit()  
    except SQLAlchemyError as e:  
        print(e)
```

2. видалення:

```
def delete(t_name, column, value):  
    try:  
        t_class = getattr(sys.modules[__name__], t_name.capitalize())  
        t_class_col = getattr(t_class, column)  
        s.query(t_class).filter(t_class_col == value).delete()  
        s.commit()  
    except SQLAlchemyError as e:  
        print(e)
```

3. редагування:

```
def update(t_name, column, value, cond):  
    try:  
        t_class = getattr(sys.modules[__name__], t_name.capitalize())  
        t_class_col = getattr(t_class, column)  
        t_class_cond_col = getattr(t_class, cond[0])  
        s.query(t_class).filter(t_class_cond_col == cond[1]).update({t_class_col: value})  
        s.commit()  
    except SQLAlchemyError as e:  
        print(e)
```

## Пункт №2

### Створення індексу

#### Hash

```
1 CREATE INDEX hash_on_text ON index_test USING HASH(t_text)
2
```

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 578 msec.

#### B-tree

```
1 CREATE INDEX btree_on_text ON index_test USING Btree(t_text)
```

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 557 msec.

## Приклад фільтрації даних

### Без індексації

```
1 explain analyze select * from index_test where t_text = 'IGHDU'
2
```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Seq Scan on index_test (cost=0.00..1791.00 rows=1 width=14) (actual time=0.022..10.781 rows=1 loops=1)
2	Filter: ((t_text)::text = 'IGHDU'::text)
3	Rows Removed by Filter: 99999
4	Planning Time: 1.897 ms
5	Execution Time: 10.801 ms

### Hash

```
1 -- CREATE INDEX hash_on_text ON index_test USING HASH(t_text)
2 explain analyze select * from index_test where t_text = 'IGHDU'
```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Index Scan using hash_on_text on index_test (cost=0.00..8.02 rows=1 width=14) (actual time=0.028..0.028 rows=1 loops=1)
2	Index Cond: ((t_text)::text = 'IGHDU'::text)
3	Planning Time: 1.328 ms
4	Execution Time: 0.062 ms

### B-tree





```
1 explain analyze select * from index_test where t_text = 'IGHDU'
```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Index Scan using btree_on_text on index_test (cost=0.29..8.31 rows=1 width=14) (actual time=0.161..0.163 rows=1 loops=1)
2	Index Cond: ((t_text)::text = 'IGHDU'::text)
3	Planning Time: 0.771 ms
4	Execution Time: 0.195 ms

Результат

```
1 select * from index_test where t_text = 'IGHDU'
```

Data Output					Explain	Messages	Notifications
	 id [PK] integer	 t_text character varying (5)	 t_num integer				
1	2	IGHDU	65				



## Приклад з агрегатними функціями

### Без індексації

```
1 explain analyze select min(t_text) from index_test
```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Aggregate (cost=1791.00..1791.01 rows=1 width=32) (actual time=36.797..36.798 rows=1 loops=1)
2	-> Seq Scan on index_test (cost=0.00..1541.00 rows=100000 width=6) (actual time=0.018..5.829 rows=100000 loops=1)
3	Planning Time: 1.029 ms
4	Execution Time: 36.823 ms

### Hash

```
1 explain analyze select min(t_text) from index_test
```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Aggregate (cost=1791.00..1791.01 rows=1 width=32) (actual time=45.144..45.145 rows=1 loops=1)
2	-> Seq Scan on index_test (cost=0.00..1541.00 rows=100000 width=6) (actual time=0.016..6.471 rows=100000 loops=1)
3	Planning Time: 1.146 ms
4	Execution Time: 45.175 ms

### B-tree

```
1 explain analyze select min(t_text) from index_test
```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Result (cost=0.32..0.33 rows=1 width=32) (actual time=0.028..0.029 rows=1 loops=1)
2	InitPlan 1 (returns \$0)
3	-> Limit (cost=0.29..0.32 rows=1 width=32) (actual time=0.026..0.026 rows=1 loops=1)
4	-> Index Only Scan using btree_on_text on index_test (cost=0.29..2854.29 rows=100000 width=32) (actual time=0.025..0.025 rows=1 loops=1)
5	Index Cond: (t_text IS NOT NULL)
6	Heap Fetches: 0
7	Planning Time: 0.130 ms
8	Execution Time: 0.043 ms

Результат

1

```
select min(t_text) from index_test
```

Data Output

Explain

Messages

Notifications

min

text

1

AAAEH

# Приклад з агрегатними функціями та групуванням

## Без індексації

1	<code>explain analyze select count(t_text) from index_test group by (t_text)</code>
<div><div>Data Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div>	
	<div><div>QUERY PLAN</div><div>text</div><div>1 HashAggregate (cost=7166.00..8937.32 rows=99007 width=14) (actual time=37.534..72.519 rows=99469 loops=1)</div><div>2 Group Key: t_text</div><div>3 Planned Partitions: 4 Batches: 5 Memory Usage: 4145kB Disk Usage: 1792kB</div><div>4 -&gt; Seq Scan on index_test (cost=0.00..1541.00 rows=100000 width=6) (actual time=0.021..5.579 rows=100000 loops=1)</div><div>5 Planning Time: 1.243 ms</div><div>6 Execution Time: 77.672 ms</div></div>

## Hash

1	<code>explain analyze select count(t_text) from index_test group by (t_text)</code>
<div><div>Data Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div>	
	<div><div>QUERY PLAN</div><div>text</div><div>1 HashAggregate (cost=7166.00..8937.32 rows=99007 width=14) (actual time=35.671..73.736 rows=99469 loops=1)</div><div>2 Group Key: t_text</div><div>3 Planned Partitions: 4 Batches: 5 Memory Usage: 4145kB Disk Usage: 1792kB</div><div>4 -&gt; Seq Scan on index_test (cost=0.00..1541.00 rows=100000 width=6) (actual time=0.015..5.122 rows=100000 loops=1)</div><div>5 Planning Time: 0.073 ms</div><div>6 Execution Time: 78.332 ms</div></div>

## B-tree

1	<code>explain analyze select count(t_text) from index_test group by (t_text)</code>
<div><div>Data Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div>	
	<div><div>QUERY PLAN</div><div>text</div><div>1 GroupAggregate (cost=0.29..4094.36 rows=99007 width=14) (actual time=0.021..46.469 rows=99469 loops=1)</div><div>2 Group Key: t_text</div><div>3 -&gt; Index Only Scan using btree_on_text on index_test (cost=0.29..2604.29 rows=100000 width=6) (actual time=0.015..9.462 rows=100000 loops=1)</div><div>4 Heap Fetches: 0</div><div>5 Planning Time: 0.093 ms</div><div>6 Execution Time: 49.605 ms</div></div>

Результат

1 select count(t_text) from index_test group by(t_text)		
Data Output Explain Messages Notifications		
	count bigint	lock
1	1	
2	1	
3	1	
4	1	
5	1	
6	1	
7	1	
8	1	
9	1	
10	1	
11	1	
12	2	
13	1	
14	1	
15	1	

# Приклад з сортуванням

## Без індексації

1	<code>explain analyze select * from index_test where t_text like 'F%' order by t_text</code>
<div><div>Data Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div>	
<div><div>QUERY PLAN</div><div>text</div></div>	
1	Sort (cost=2033.00..2043.10 rows=4040 width=14) (actual time=21.481..21.611 rows=4013 loops=1)
2	Sort Key: t_text
3	Sort Method: quicksort Memory: 285kB
4	-> Seq Scan on index_test (cost=0.00..1791.00 rows=4040 width=14) (actual time=0.014..8.544 rows=4013 loops=1)
5	Filter: ((t_text)::text ~~ 'F%':text)
6	Rows Removed by Filter: 95987
7	Planning Time: 0.083 ms
8	Execution Time: 21.755 ms

## Hash





1	<code>explain analyze select * from index_test where t_text like 'F%' order by t_text</code>
<div><div>Data Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div>	
<div><div>QUERY PLAN</div><div>text</div></div>	
1	Sort (cost=2033.00..2043.10 rows=4040 width=14) (actual time=21.812..21.961 rows=4013 loops=1)
2	Sort Key: t_text
3	Sort Method: quicksort Memory: 285kB
4	-> Seq Scan on index_test (cost=0.00..1791.00 rows=4040 width=14) (actual time=0.019..7.635 rows=4013 loops=1)
5	Filter: ((t_text)::text ~~ 'F%':text)
6	Rows Removed by Filter: 95987
7	Planning Time: 4.586 ms
8	Execution Time: 22.112 ms

## B-tree

1	<code>explain analyze select * from index_test where t_text like 'F%' order by t_text</code>
<div><div>Data Output</div><div>Explain</div><div>Messages</div><div>Notifications</div></div>	
<div><div>QUERY PLAN</div><div>text</div></div>	
1	Sort (cost=2033.00..2043.10 rows=4040 width=14) (actual time=21.488..21.671 rows=4013 loops=1)
2	Sort Key: t_text
3	Sort Method: quicksort Memory: 285kB
4	-> Seq Scan on index_test (cost=0.00..1791.00 rows=4040 width=14) (actual time=0.014..8.033 rows=4013 loops=1)
5	Filter: ((t_text)::text ~~ 'F%':text)
6	Rows Removed by Filter: 95987
7	Planning Time: 0.091 ms
8	Execution Time: 21.770 ms

## Результат

```
1 select * from index_test where t_text like 'F%' order by t_text
```

	Data Output	Explain	Messages	Notifications
				
	id [PK] integer	t_text character varying (5)	t_num integer	
1	37776	FAAAG	65	
2	27998	FAAFP	98	
3	33587	FAAKD	46	
4	25260	FAALQ	45	
5	98475	FAAUQ	5	
6	72933	FAAXJ	81	
7	57537	FABEN	77	
8	87199	FABJH	80	
9	98893	FABKC	98	
10	36822	FABRA	72	

## Висновки

1. При фільтрації обидва індекси чудово виконують свою роботу в порівнянні з послідовним пошуком при відсутності індексів.
2. При використанні агрегатних функцій, використання індексів залежить від агрегатної функції. Наприклад, використання функції *min* дає змогу використовувати Btree індекс та збільшує швидкість пошуку оскільки необхідно лише дійти до крайнього лівого «листка». Використання функції *count* не дає змогу використати Btree або Hash індекс тому що в результаті маємо лише один запис.
3. При групуванні та використовуючи функцію *count* ефективним виявився лише Btree індекс. Hash індекс скоріш за все не використовувався оскільки було недостатньо унікальних даних.
4. При сортуванні та використовуючи функцію *like* ні один з даних індексів не виявився ефективним оскільки не використовувались порівняння  $=$ ,  $\leq$ ,  $\geq$ ,  $<$ ,  $>$  або  $=$ .
5. Також ми переконалися, що на великих таблицях ефективніше використовувати індекси ніж звичайний послідовний пошук.

Пункт №3

Команди, що ініціюють виконання тригера, код тригера:

```
1 CREATE OR REPLACE FUNCTION func() RETURNS trigger AS $func$
2 BEGIN
3     IF (TG_OP = 'INSERT') THEN
4         BEGIN
5             IF (
6                 (SELECT count(*) FROM trigger_test WHERE id = ANY(NEW.child_id::int[]))
7                 !=
8                 array_length(NEW.child_id::int[], 1)
9             ) THEN
10                RAISE EXCEPTION 'child must exist';
11            END IF;
12            RETURN NEW;
13        END;
14    ELSEIF (TG_OP = 'DELETE') THEN
15        DECLARE
16            del_cur CURSOR FOR
17            SELECT * FROM trigger_test WHERE OLD.id = ANY(child_id::int[]);
18        BEGIN
19            FOR record IN del_cur LOOP
20                BEGIN
21                    UPDATE trigger_test SET child_id = array_remove(child_id, OLD.id) WHERE id = record.id;
22                END;
23            END LOOP;
24        END;
25        RETURN OLD;
26    END IF;
27 END;
28 $func$ LANGUAGE plpgsql;
29
30 CREATE TRIGGER parent_child
31 BEFORE INSERT OR DELETE
32 ON trigger_test
33 FOR EACH ROW
34 EXECUTE PROCEDURE func();
35
```

Таблиця до змін:

	<div>id</div> <div>[PK] integer</div>	<div>child_id</div> <div>integer[]</div>
1	31	{32,32}
2	32	{33}
3	33	{32}
4	37	[null]

1. Якщо додається новий запис до таблиці, то усі id що містяться у child\_id мають існувати.

	id [PK] integer	child_id integer[]
1	31	{32,32}
2	32	{33}
3	33	{32}
4	37	[null]
5	[default]	{31,1000}

❗ ОШИБКА: child must exist CONTEXT: функция PL/pgSQL func(), строка 10, оператор RAISE .

	id [PK] integer	child_id integer[]
1	31	{32,32}
2	32	{33}
3	33	{32}
4	37	[null]
5	40	{31,32}

✔ Data saved successfully.



2. При видаленні запису, записи що містили у child\_id id видаляемого запису, змінять вмість child\_id на такий самий масив, але вже без id видаляемого запису

```
1 DELETE FROM trigger_test WHERE id=32
```

	id [PK] integer	child_id integer[]
1	31	{}
2	33	{}
3	37	[null]
4	40	{31}