

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**

*на тему: “Засоби оптимізації роботи СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-01

Орел І. В.

Перевірив:

Київ – 2022

*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.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Варіант 8**

У другому завданні проаналізувати індекси BTree, GIN.

Умова для тригера – after insert, update.

**Інформація про модель та структуру бази даних**

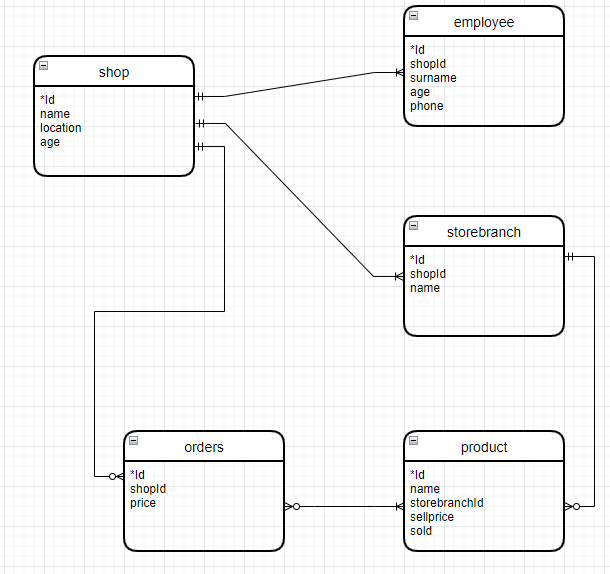


Рисунок 1 - Концептуальна модель предметної області “Магазини”

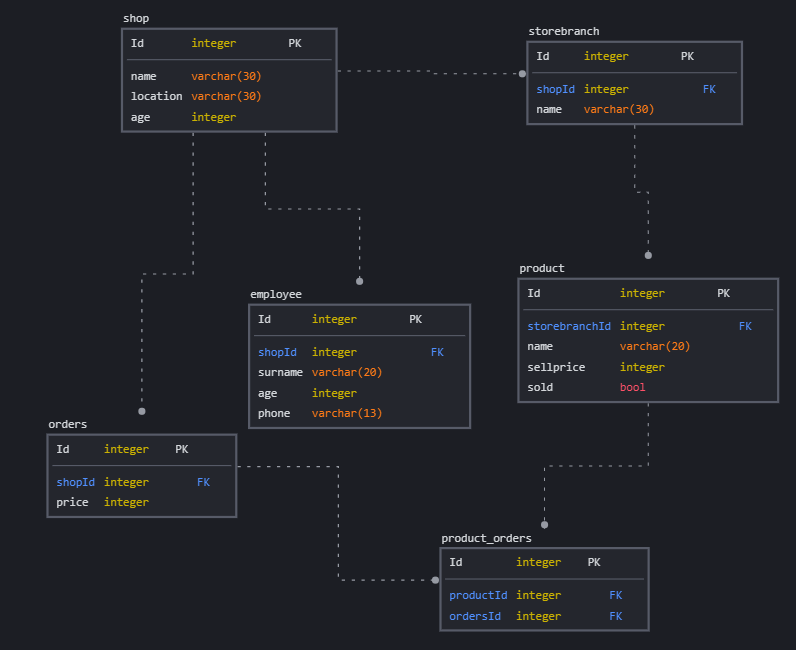


Рисунок 2 - Логічна модель предметної області “Магазини ”

**1 Завдання**

**Класи ORM**

from django.db import models  
# Create your models here.  
  
class Shop(models.Model):  
 id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=30)  
 location = models.CharField(max\_length=30)  
 age = models.IntegerField(null=True)  
  
 class Meta:  
 db\_table = 'shop'  
  
  
class Employee(models.Model):  
 id = models.AutoField(primary\_key=True)  
 shopid = models.ForeignKey(Shop, on\_delete=models.CASCADE, db\_column="shopid")  
 surname = models.CharField(max\_length=20)  
 age = models.IntegerField()  
 phone = models.CharField(unique=True, max\_length=13)  
  
 class Meta:  
 db\_table = 'employee'  
  
  
class Storebranch(models.Model):  
 id = models.AutoField(primary\_key=True)  
 shopid = models.ForeignKey(Shop, on\_delete=models.CASCADE, db\_column="shopid")  
 name = models.CharField(max\_length=30)  
  
 class Meta:  
 db\_table = 'storebranch'  
  
  
class Product(models.Model):  
 id = models.AutoField(primary\_key=True)  
 name = models.CharField(unique=True, max\_length=20)  
 storebranchid = models.ForeignKey(Storebranch, on\_delete=models.CASCADE, db\_column="storebranchid")  
 sellprice = models.IntegerField()  
 sold = models.BooleanField(null=True)  
  
 class Meta:  
 db\_table = 'product'  
  
  
class Orders(models.Model):  
 id = models.AutoField(primary\_key=True)  
 shopid = models.ForeignKey(Shop, on\_delete=models.CASCADE, db\_column="shopid")  
 price = models.IntegerField()  
  
  
class Product\_orders(models.Model):  
 ordersid = models.ForeignKey(Orders, primary\_key=True, on\_delete=models.CASCADE, unique=False, db\_column="ordersid")  
 productid = models.ForeignKey(Product, on\_delete=models.CASCADE, unique=False, db\_column="productid")  
  
 class Meta:  
 unique\_together = (('ordersid', 'productid'),)  
 db\_table = 'product\_orders'

**2 Завдання**

**BTree**

Для дослідження індексу була створена таблиця, яка має дві колонки: числову і текстову. Вони проіндексовані як BTree. У таблицю було занесено 1000000 записів.

**Створення таблиці та її заповнення:**

CREATE TABLE btree\_table(

id bigserial PRIMARY KEY,

test\_text varchar(255));

INSERT INTO btree\_table(test\_text)

SELECT substr(characters, (random() \* length(characters) + 1)::integer, 10)

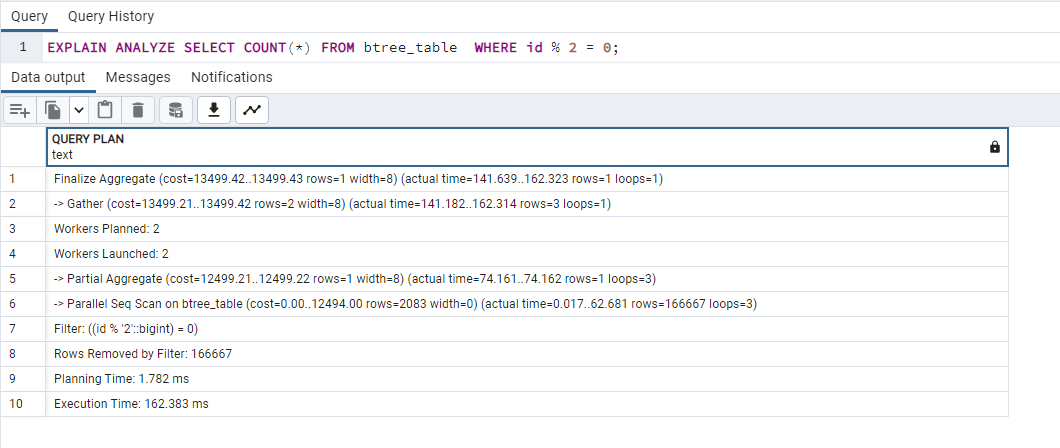
FROM

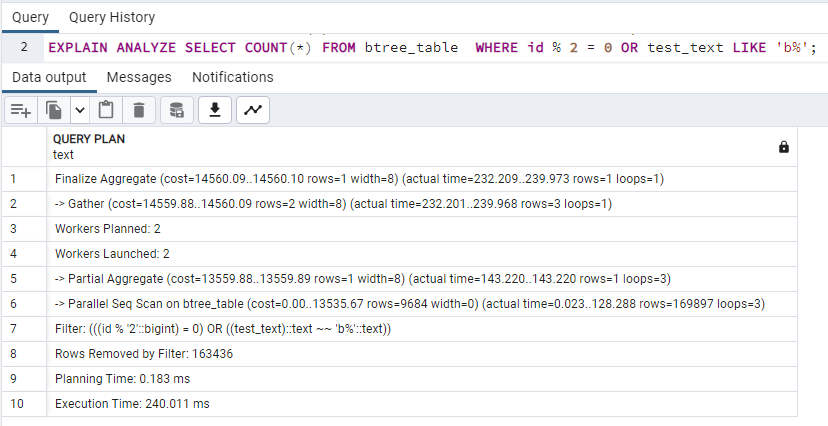
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),

generate\_series(1, 1000000) as q;

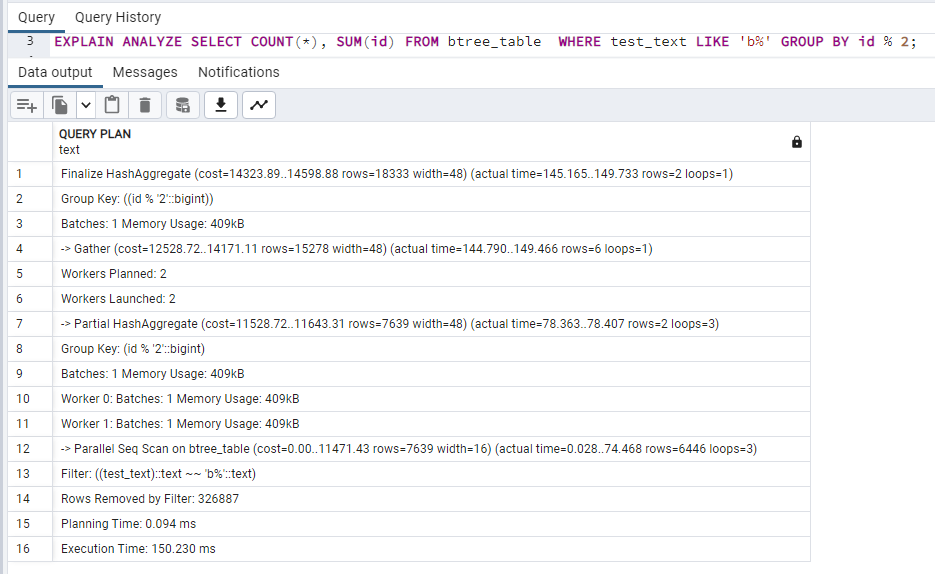
**Вибір даних без індексу:**

SELECT COUNT(\*) FROM btree\_table WHERE id % 2 = 0;



SELECT COUNT(\*) FROM btree\_table WHERE id % 2 = 0 OR test\_text LIKE 'b%';

SELECT COUNT(\*), SUM(id) FROM btree\_table WHERE test\_text LIKE 'b%' GROUP BY id % 2;



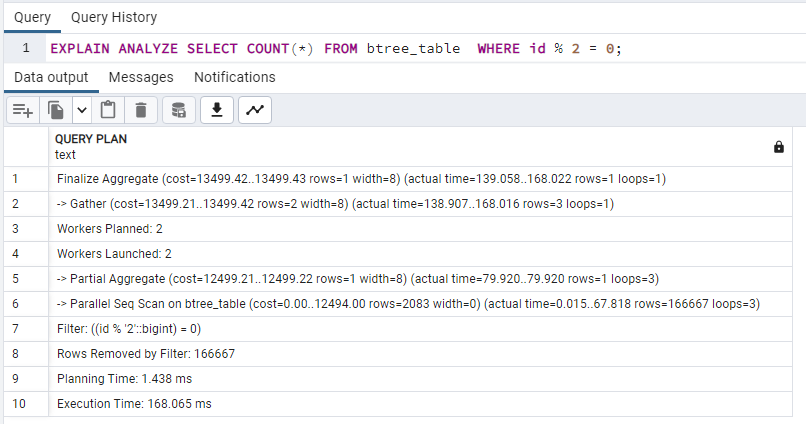
**Створення індексу:**

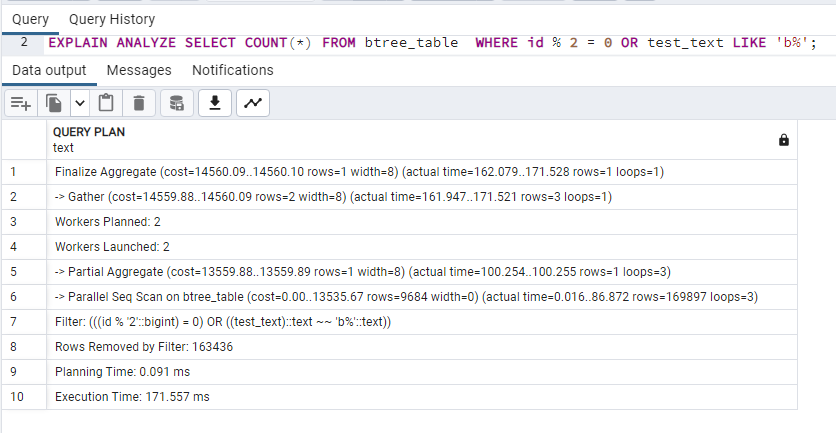
DROP INDEX IF EXISTS btree\_index;

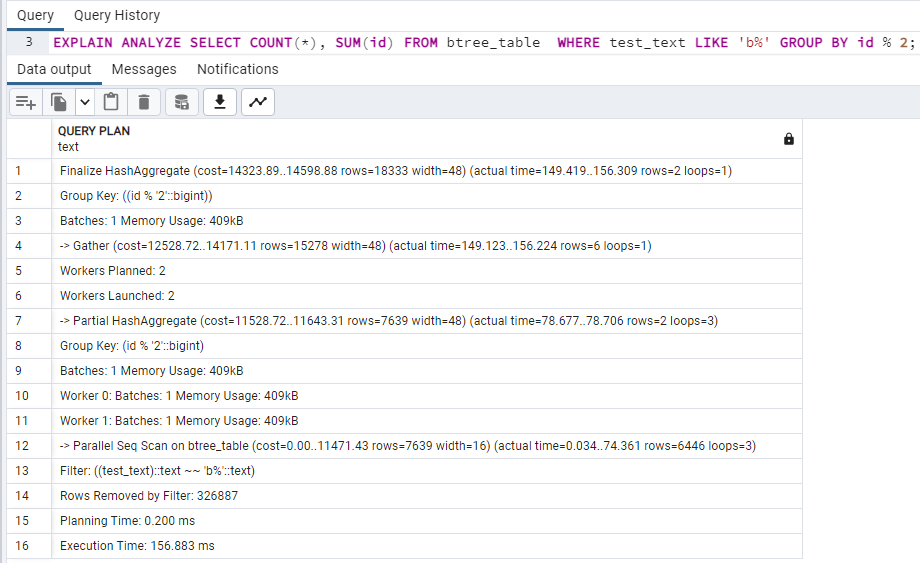
CREATE INDEX btree\_index ON btree\_table USING btree(test\_text);

**Вибір даних із створеним індексом:**

SELECT COUNT(\*) FROM btree\_table WHERE id % 2 = 0;



SELECT COUNT(\*) FROM btree\_table WHERE id % 2 = 0 OR test\_text LIKE 'b%'; 

SELECT COUNT(\*), SUM(id) FROM btree\_table WHERE test\_text LIKE 'b%' GROUP BY id % 2; 

**GIN**

Для дослідження індексу була створена таблиця, яка має дві колонки: id типу integer і test\_text типу varchar. Колонка test\_text проіндексована як GIN. У таблицю занесено 1000000 записів.

**Створення таблиці та її заповнення:**

CREATE TABLE gin\_table(

id bigserial PRIMARY KEY,

test\_text varchar(255));

INSERT INTO gin\_table(test\_text)

SELECT substr(characters, (random() \* length(characters) + 1)::integer, 10)

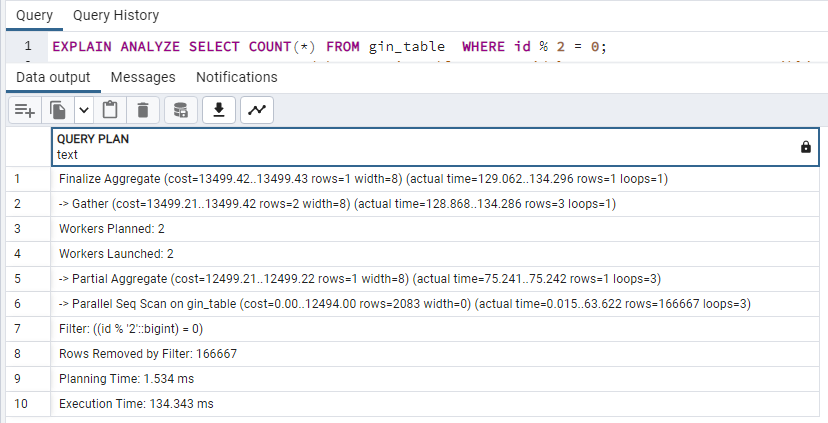
FROM

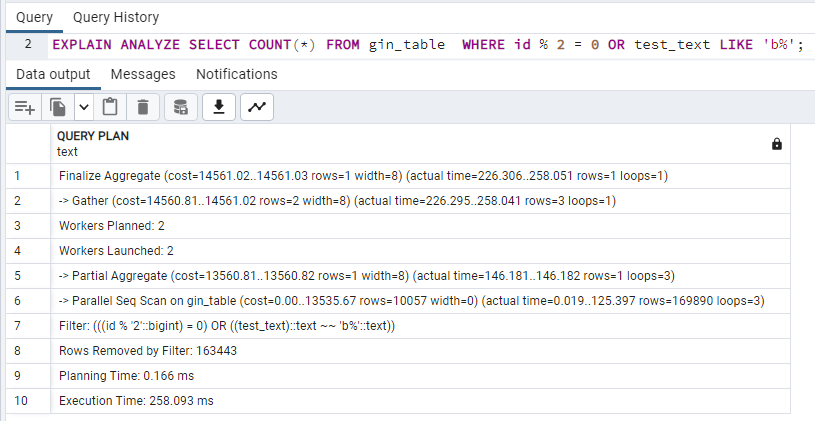
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),

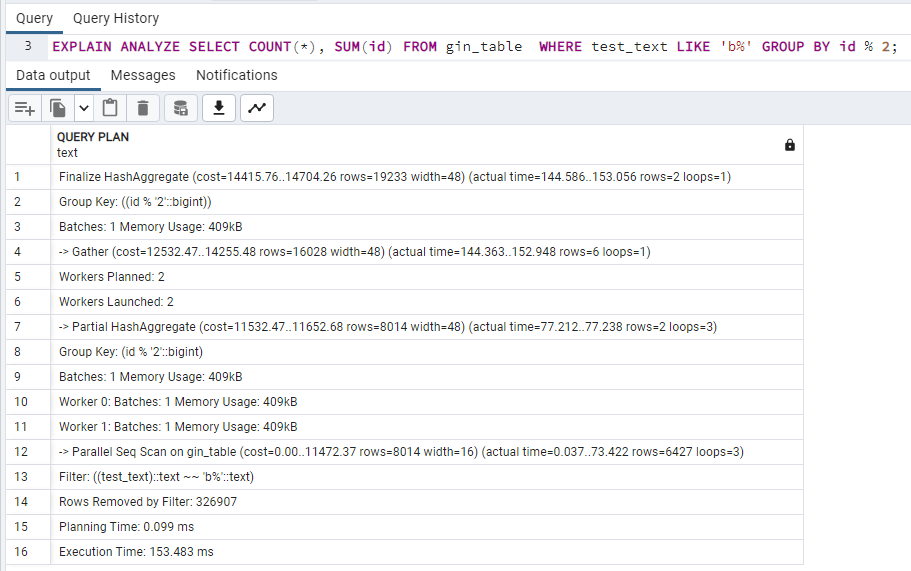
generate\_series(1, 1000000) as q;

**Вибір даних без індексу:**

SELECT COUNT(\*) FROM gin\_table WHERE id % 2 = 0;



SELECT COUNT(\*) FROM gin\_table WHERE id % 2 = 0 OR test\_text LIKE 'b%';

SELECT COUNT(\*), SUM(id) FROM gin\_table WHERE test\_text LIKE 'b%' GROUP BY id % 2; 

**Створення індексу:**

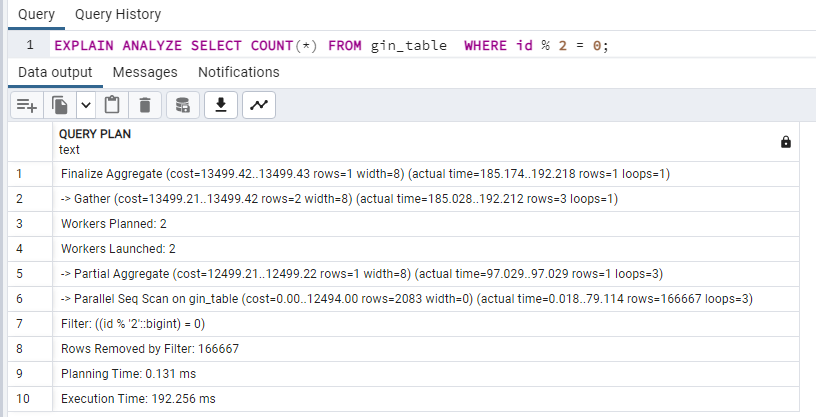
CREATE EXTENSION pg\_trgm;

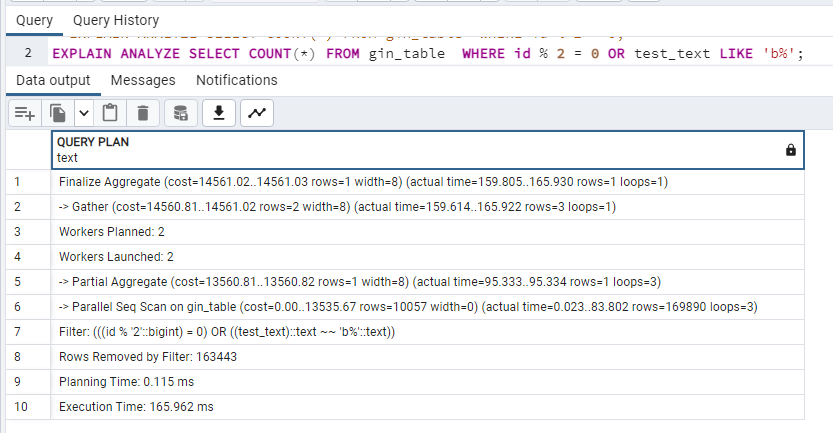
DROP INDEX IF EXISTS gin\_index;

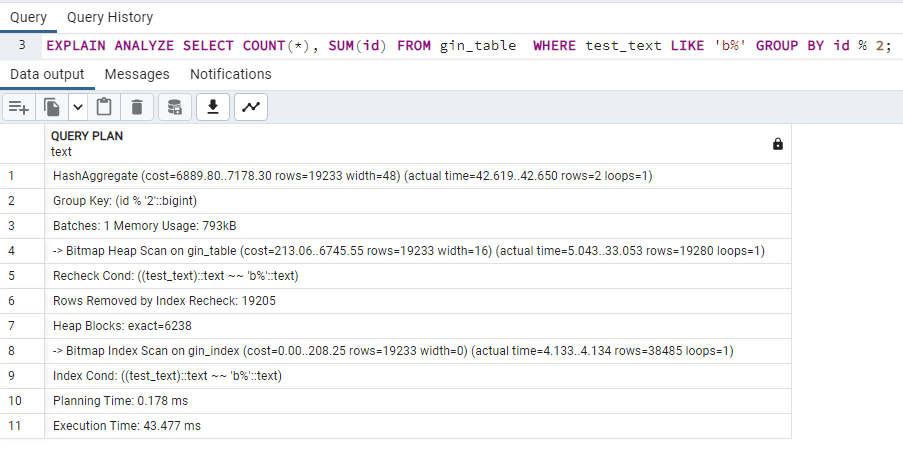
CREATE INDEX gin\_index ON gin\_table USING gin(test\_text gin\_trgm\_ops);

**Вибір даних із створеним індексом:**

SELECT COUNT(\*) FROM gin\_table WHERE id % 2 = 0;



SELECT COUNT(\*) FROM gin\_table WHERE id % 2 = 0 OR test\_text LIKE 'b%'; 

SELECT COUNT(\*), SUM(id) FROM gin\_table WHERE test\_text LIKE 'b%' GROUP BY id % 2; 

B-tree доцільно використовувати для роботи зі стовпцями даних, що можна було б сортувати за значеннями, тобто буквально використовувати на них операції більше/менше дорівнює і тд.

Основна сфера застосування методу GIN – прискорення повнотекстового пошуку. Якщо дані змінюються не часто, шукати треба швидко – GIN буде мати кращі результати, ніж GiST.

**3 Завдання**

**Тригер**

Умова: *after insert, update*

**Таблиці:**

CREATE TABLE bank\_account (

id serial PRIMARY KEY,

name varchar NOT NULL,

age integer NOT NULL,

sallary integer NOT NULL,

netto\_sallary integer

);

CREATE TABLE update\_ts\_storage (

stamp timestamp NOT NULL,

bank\_account\_id integer NOT NULL,

name varchar NOT NULL,

age integer NOT NULL

);

**Функція:**

CREATE OR REPLACE FUNCTION update\_ts\_storage\_func() RETURNS trigger AS $update\_ts\_storage\_func$

BEGIN

IF (TG\_OP = 'UPDATE') THEN

INSERT INTO update\_ts\_storage(stamp,bank\_account\_id,"name",age) SELECT now(), NEW.id, NEW.name, NEW.age;

ELSIF (TG\_OP = 'INSERT') THEN

IF NEW.sallary > 2500 THEN

UPDATE bank\_account

SET netto\_sallary = NEW.sallary - 100

WHERE id = NEW.id;

END IF;

END IF;

RETURN NEW;

END;

$update\_ts\_storage\_func$ LANGUAGE plpgsql;

**Тригер:**

CREATE TRIGGER update\_ts\_storage\_trigger AFTER INSERT OR UPDATE ON bank\_account

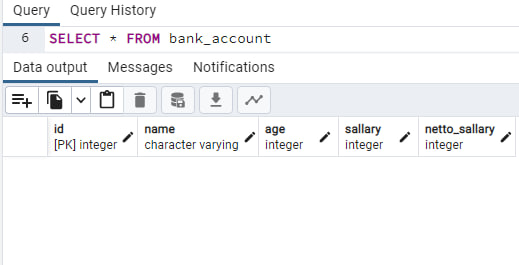
FOR EACH ROW EXECUTE FUNCTION update\_ts\_storage\_func();

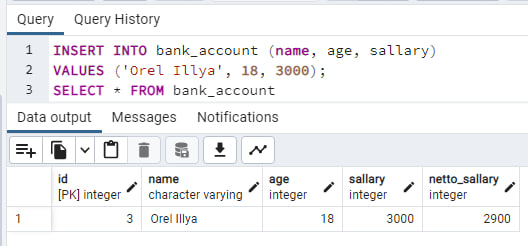
**Принцип роботи тригера:**

Тригер працює після вставки або редагування в таблиці **“bank\_account”**. Після вставки нового рядка буде перевірятись, чи значення sallary більше за 2500, і якщо так – від нього буде відніматись 100. Після кожного редагування таблиці **“bank\_account”** у таблицю **“update\_ts\_storage”** буде заноситись часова мітка відповідної зміни та інформація про відповідного користувача зі зміненого рядка, а саме: bank\_account\_id, name та age.

**Приклад роботи тригера:**

**Таблиця до змін:**



**Таблиця після вставки нового рядка:**

**Таблиці після оновлення певного рядка:**

