

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»**

Виконав: студент ІII курсу

ФПМ групи КВ-73

Єрмоленко Денис Вадимович

Перевірив(ла):

Київ – 2019

**Засоби оптимізації роботи СУБД PostgreSQL**

*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).

1. Створити та проаналізувати різні типи індексів у PostgreSQL.
2. Розробити тригер бази даних PostgreSQL.

**Варіант 7**

У другому завданні проаналізувати індекси GIN, BRIN.

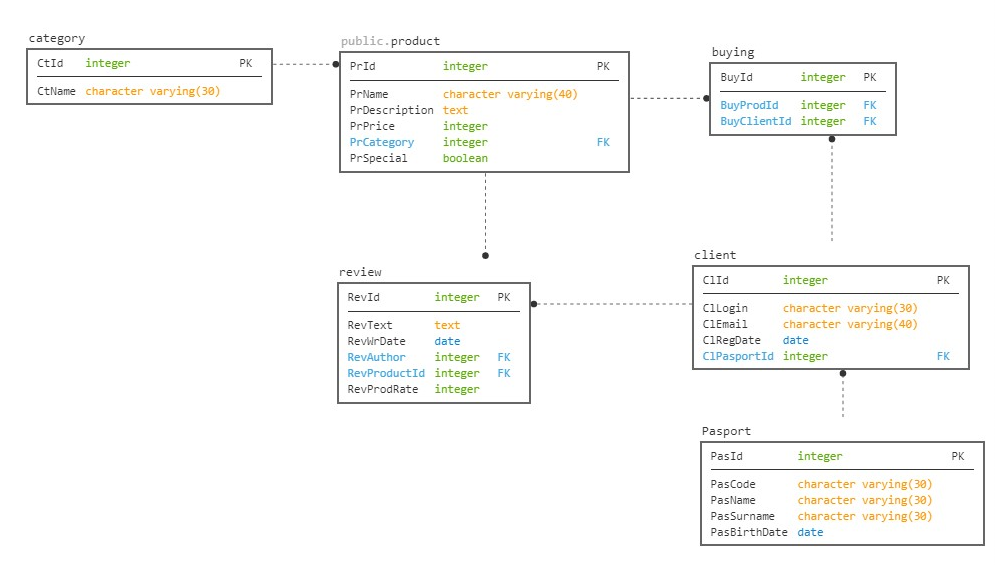
Умова для тригера – before insert, delete

**Навігація по звіту:**

1. [**Завдання 1**](#Завдання1)
2. [**Завдання 2**](#Завдання2)
3. [**Завдання 3**](#Завдання3)

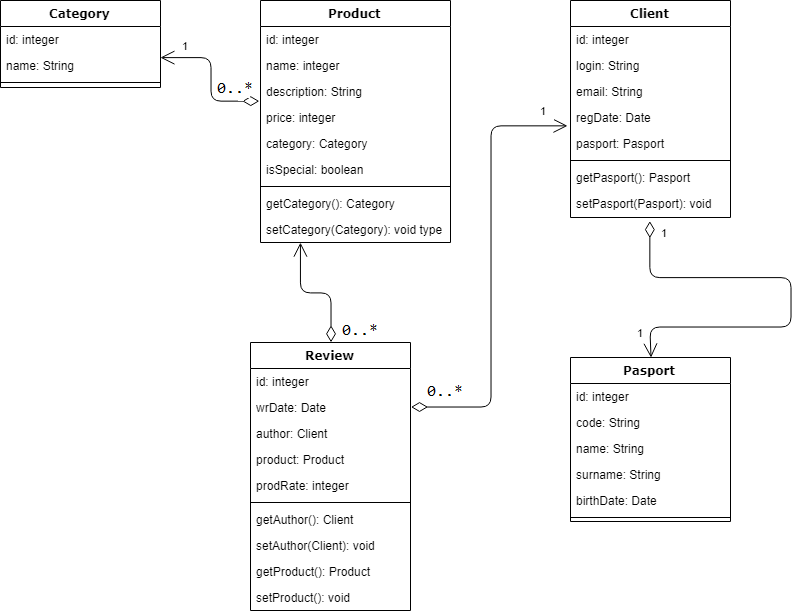
**Завдання 1**

Логічна схема бази даних “Інтернет магазин”

****

Для перетворення модулю “Model” програми, створеної в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі використовую бібліотеку Hibernate , яка є найпопулярнішою реалізацією специфікації JPA, створеної для вирішення задач об’єктно-реляційного відображення.

Зобразимо сутнісні класи програми.



Продемонструємо код лише для одного класу Category:

Сам клас Category, де через анотації описується зв’язок класу із таблицею із бази даних.

package org.lab2.model;  
  
import lombok.\*;  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "Category")  
@Data  
@ToString  
@EqualsAndHashCode  
@NoArgsConstructor  
@AllArgsConstructor  
public class Category {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*SEQUENCE*, generator = "cat\_gen")  
 @SequenceGenerator(name = "cat\_gen" , sequenceName = "\"Category\_CtId\_seq\"", allocationSize = 1)  
 @Column(name = "\"CtId\"")  
 private int id;  
  
 @Column(name = "\"CtName\"")  
 private String name;  
  
 public Category(String name){  
 this.name = name;  
 }  
}

Функція для вставки у таблицю

public void insert\_in\_table\_category(String categoryName)  
{  
 Session session = factory.openSession();  
 Transaction tx = null;  
 try{  
 tx = session.beginTransaction();  
 Category category = new Category(categoryName);  
 session.save(category);  
 tx.commit();  
 System.*out*.println("Insertion has been successfully maden!");  
 } catch (HibernateException e) {  
 if (tx!=null) tx.rollback();  
 e.printStackTrace();  
 } finally {  
 session.close();  
 }  
}

Функція для модифікації даних

public void update\_in\_table\_category(String ctName, String ctNameNew){  
 Session session = factory.openSession();  
 Transaction tx = null;  
  
 try{  
 tx = session.beginTransaction();  
 String sql = "FROM Category WHERE name = ?1";  
 Query query = session.createQuery(sql);  
 query.setParameter(1 , ctName);  
 Category category = (Category) query.getSingleResult();  
 category.setName(ctNameNew);  
 session.update(category);  
 tx.commit();  
 }catch (HibernateException e){  
 if (tx!=null) tx.rollback();  
 e.printStackTrace();  
 } finally {  
 session.close();  
 }  
}

Та функція для видалення даних.

public void delete\_from\_table\_category(String catName){  
 Session session = factory.openSession();  
 Transaction tx = null;  
 try{  
 tx = session.beginTransaction();  
 String sql = "FROM Category WHERE name = ?1";  
 Query query = session.createQuery(sql);  
 query.setParameter(1 , catName);  
 Category category = (Category) query.getSingleResult();  
 session.delete(category);  
 tx.commit();  
 }catch (HibernateException e){  
 if (tx!=null) tx.rollback();  
 e.printStackTrace();  
 } finally {  
 session.close();  
 }  
}

Інші класи та функції реалізовані по аналогії.

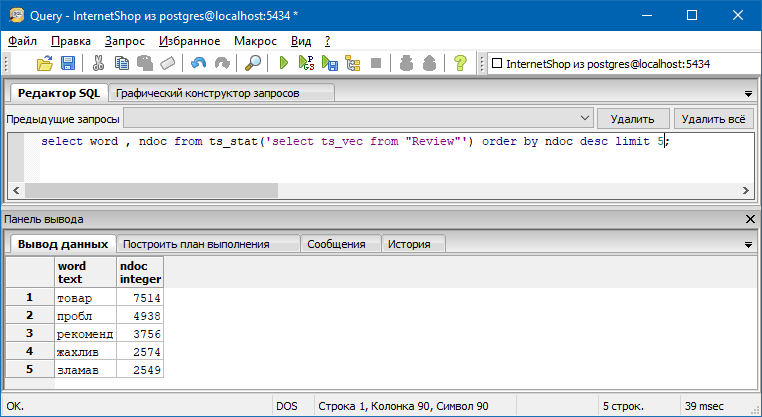
**Завдання 2**

**Створення та аналіз індекса GIN**

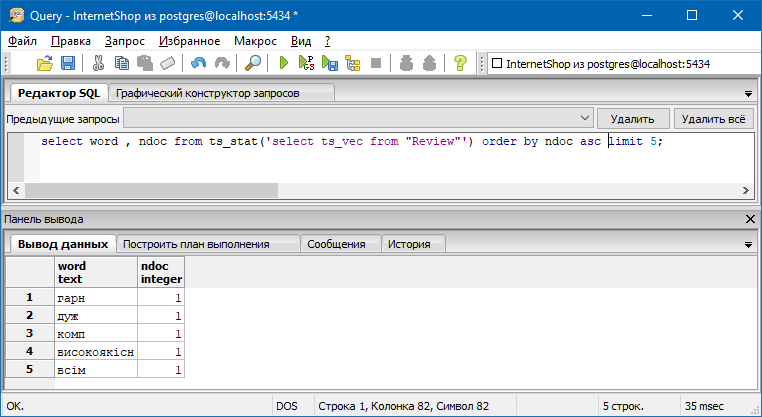
**GIN** - це Generalized Inverted Index, або обернений індекс. Його основною задачею є прискорення повнотекстового пошуку і тому вивчати даний індекс будемо на цьому прикладі.

Додамо до таблиці Review колонку ts\_vec с типом tsvector та заповнимо його за допомогою функції to\_tsvector, який буде приймати як аргумент значення колонки RevText . Саме на колонці ts\_vec будемо створювати індекс, але спочатку подивимось на швидкість повнотекстового пошуку без індекса.

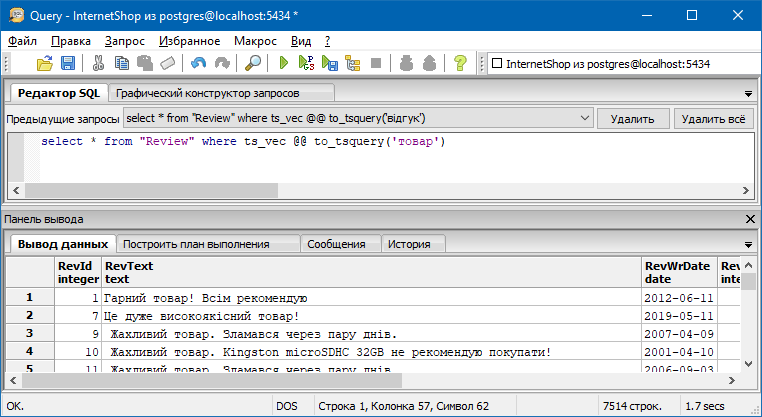
Для початку дізнаємось, які лексеми найчастіше зустрічаються, а які ні.

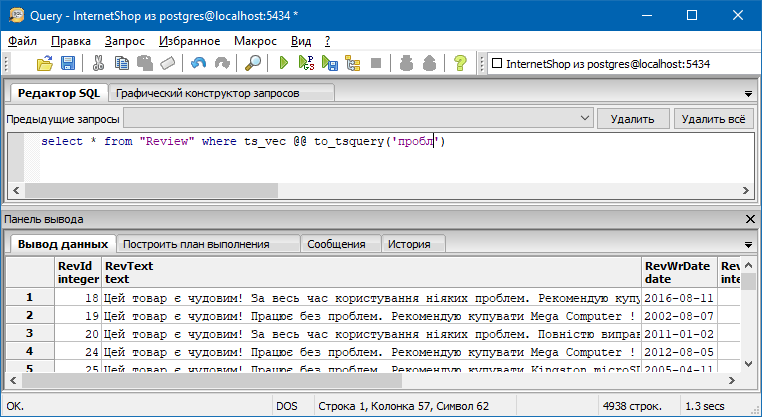


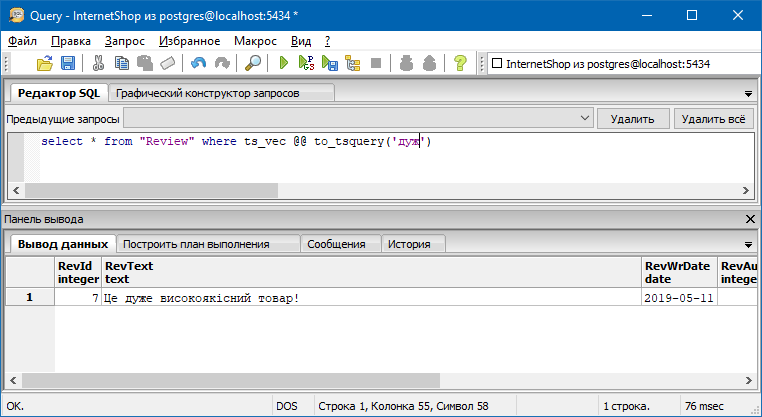
Бачимо, що найчастіше зустрічається “товар” та “пробл”.

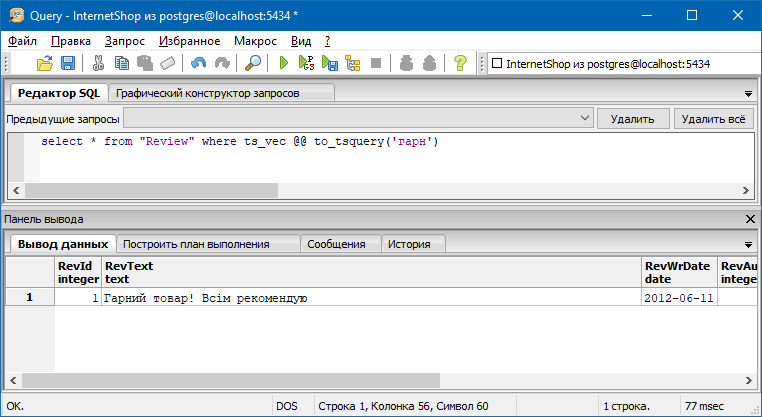


А рідше всього зустрічаються “гарн” та “дуж”.

Тепер виконаємо пошук по цих словах без індексу



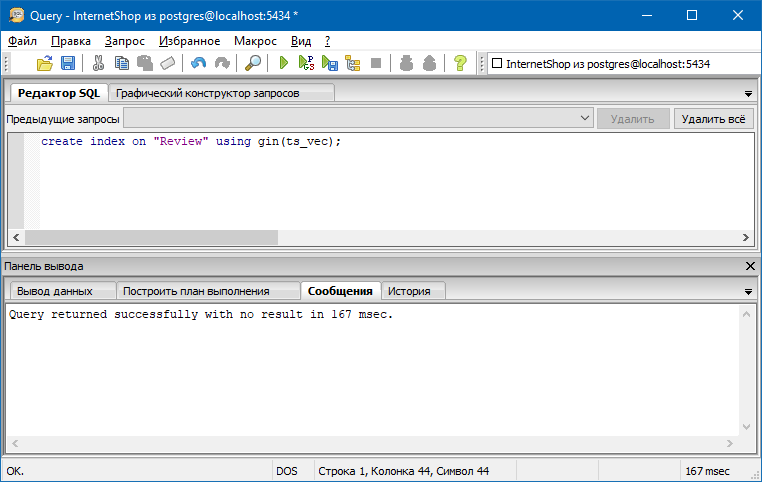




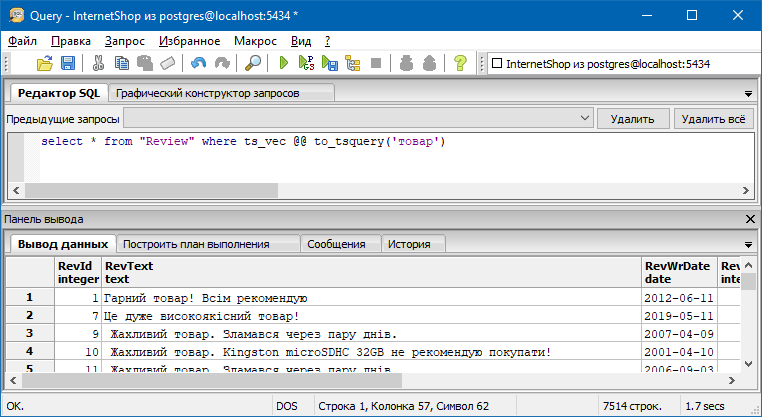
Бачимо, що на пошук та відображення найчастіше зустрічаємої лексеми “товар” було затрачено 1.7 secs , а на “побл” 1.3 secs.

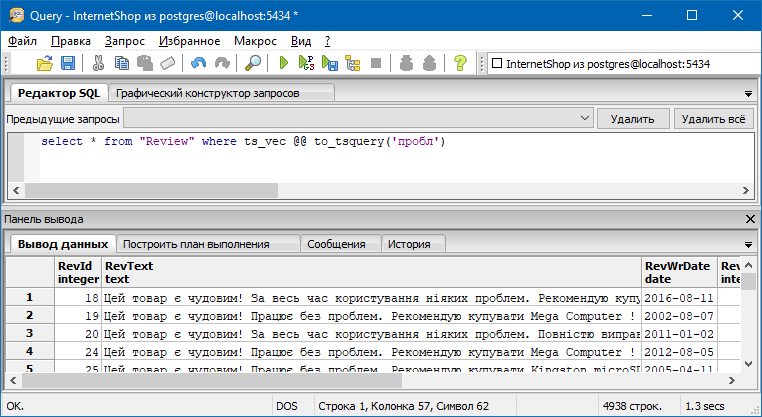
На найрідше зустрічаємий лексемах “дуж” та “гарн” 76 msec та 77 msec.

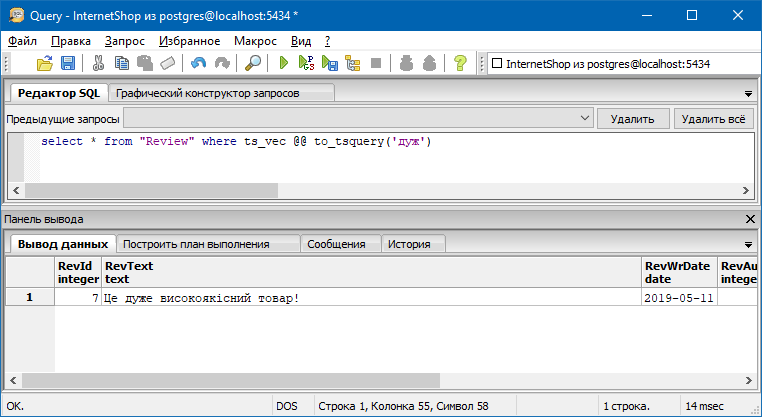
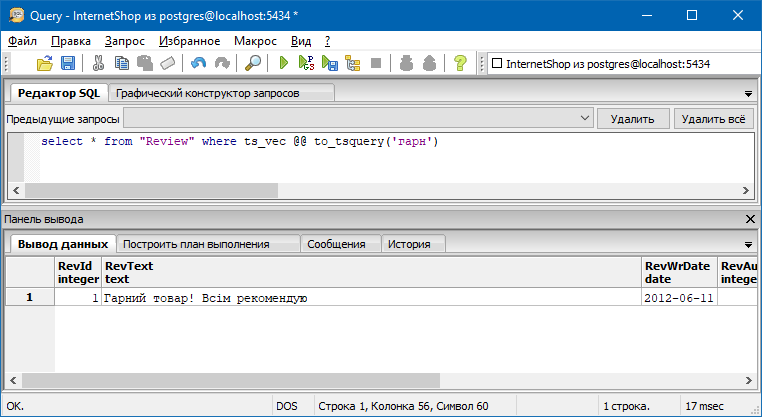
Тепер створимо індекс на колонці ts\_vec таблиці “Review”.



Знову виконаємо попередні команди та порівняємо результати:





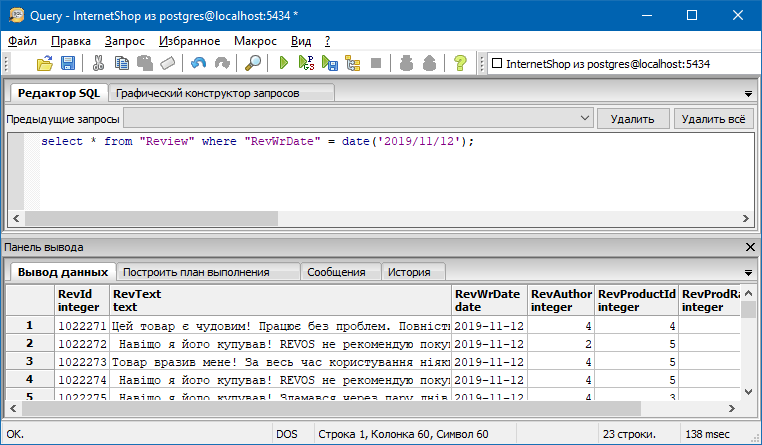


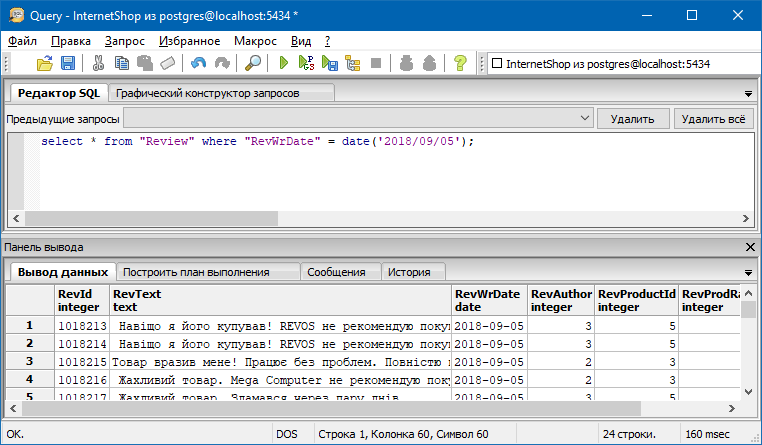
Тепер бачимо, що після створення індекса, пошук та відображення даних для найчастіше зустрічаємих лексем не змінилося, але для рідко зустрічаємих лексем видно вже покращення результату приблизно в 5 разів. Тобто бачимо, що застосування індексу GIN покращило роботу повнотекстового пошуку.

GIN добре підходить для даних, які не часто оновлюються. Якщо поміркувати то для таблиці Review, що зберігає дані про відгуки, які для товарів з’являються часто, то використовувати GIN все таки не рекомендується , бо переіндексація може займати багато часу.

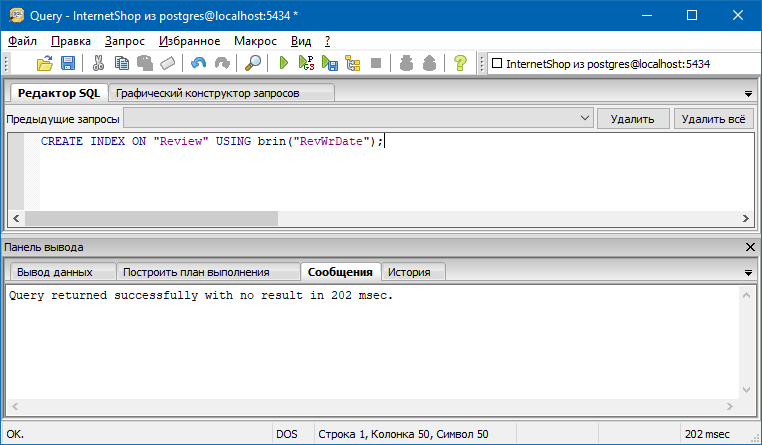
**Створення та аналіз індекса BRIN**

BRIN – це Block Range Index. Він працює добре для тих стовпчиків, де значення корелюють із їх фізичним положенням в таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання.

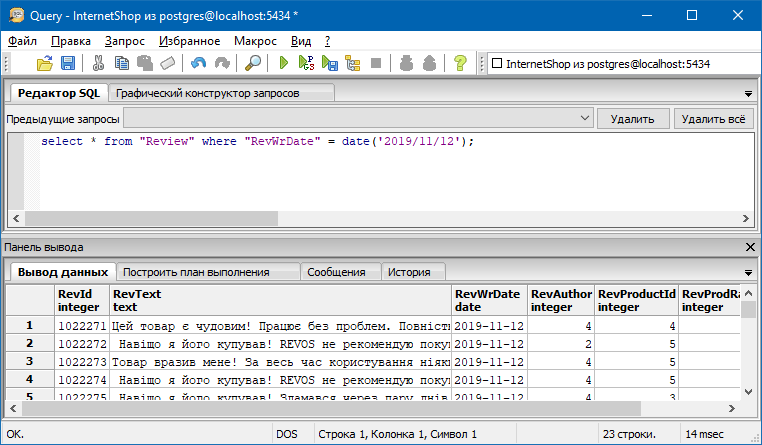
Тому будемо тестувати індекс BRIN на таблиці Review, як і для попереднього індекса, але по колонці RevWrDate, яка зберігає дату, коли був доданий відгук. Дані у стовпчику RevWrDate є впорядковані вже природньо за зростанням дати, тому саме цей стовпчик добре підходить для тестування. Спочатку дізнаємось швидкість запиту рядків з конкретною датою без індексу.

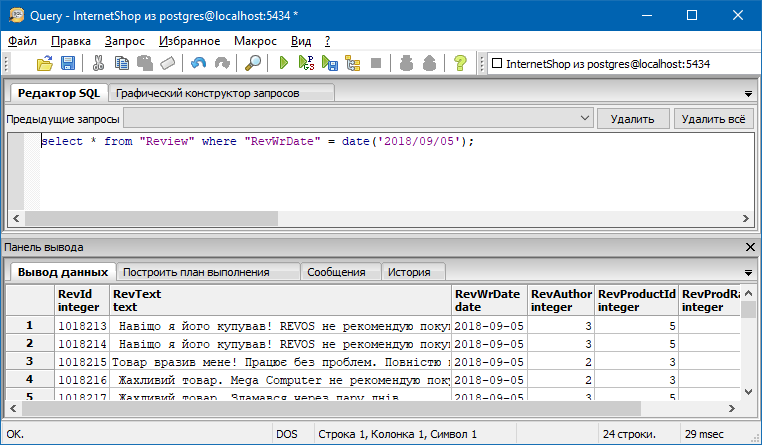


Тепер створимо індекс.



Та виконаємо тіж самі запити та потім порівняємо результати.



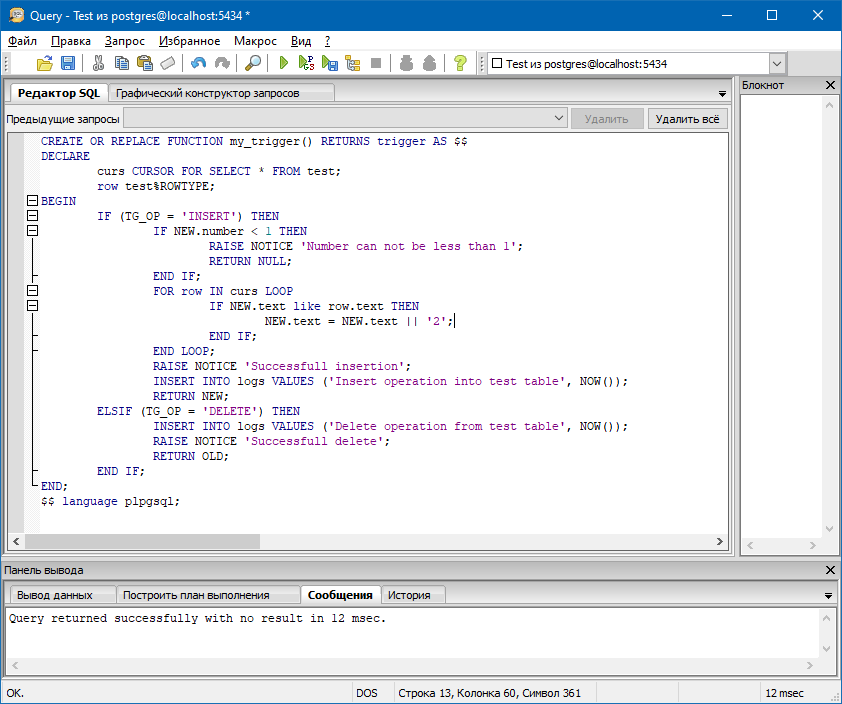


Перший запит без індексу виконався за 138 msec , а після створення індексу вже за 14 msec.

Другий запит без індексу виконався за 160 msec, а після створення індексу вже за 29 msec.

Бачимо, що була отримана оптимізація і даний індекс добре працює, але як було вже зазначено на початку, бо стовпчик “RevWrDate” був уже відсортований. Для таблиць без кореляції певних стовпчиків із фізичним положенням даний індекс не буде добре працювати по причині внутрішньої реалізації даного індексу.

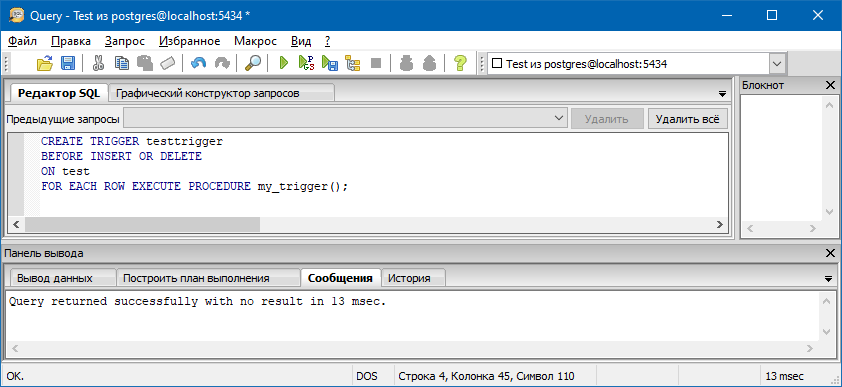
**Завдання 3**

 Команда створення тригеру

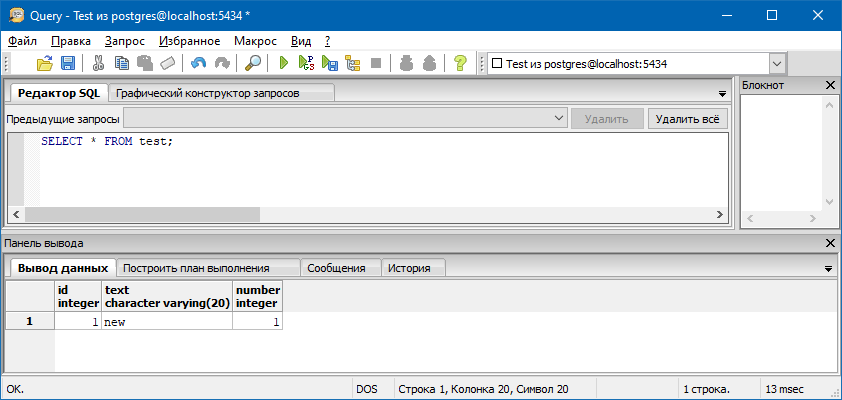
Даний тригер працює з операціями вставки та видалення.

Якщо маємо операцію вставки, то перевіряємо значення поля number у нового кортежу і якщо воно має значення менше за 1, то відміняємо операцію. Далі йде перевірка поля text у нового кортежу і якщо його значення вже є у таблиці test, то додаємо ‘2’ в кінець text. Далі проводимо вставку даних до таблиці logs та повертаємо NEW для вставки у таблицю.

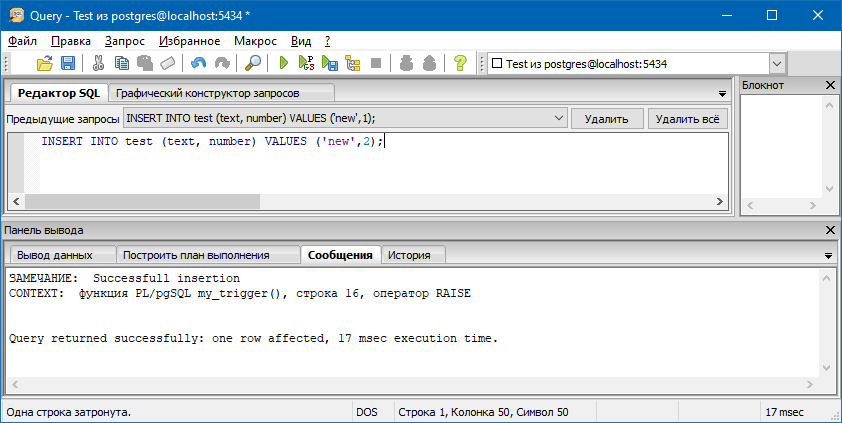
Якщо маємо операцію видалення, то просто додаємо про це інформацію у таблицю logs.

 Підключення тригеру до таблиці test

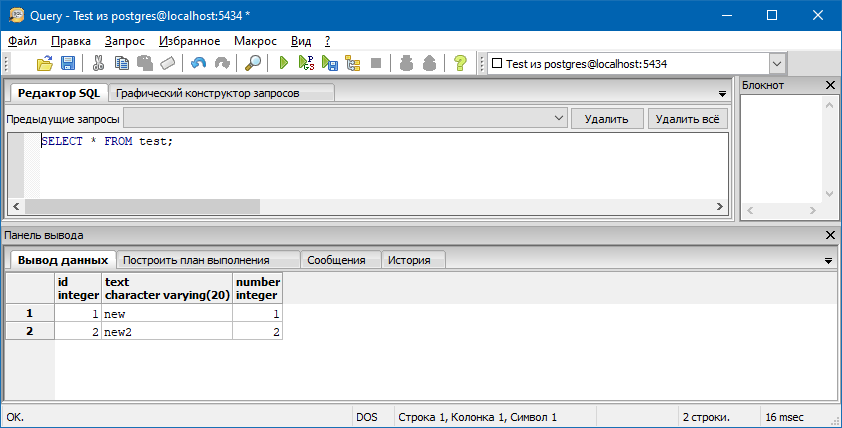
Таблиця test, до вставки



Вставка нового значення

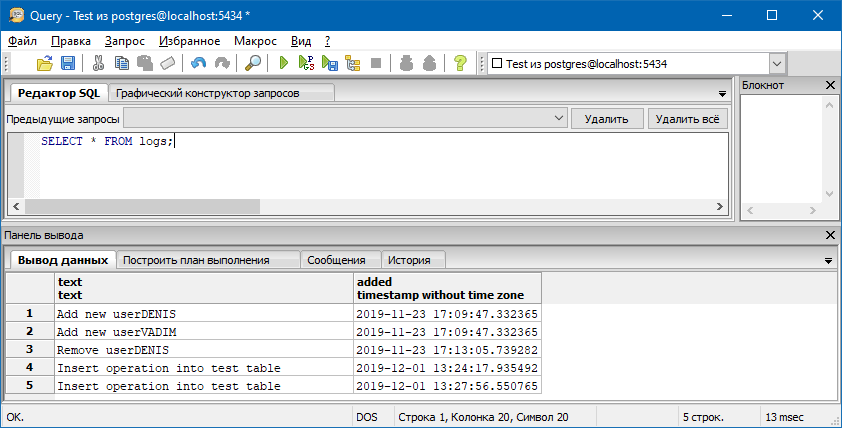


Результат



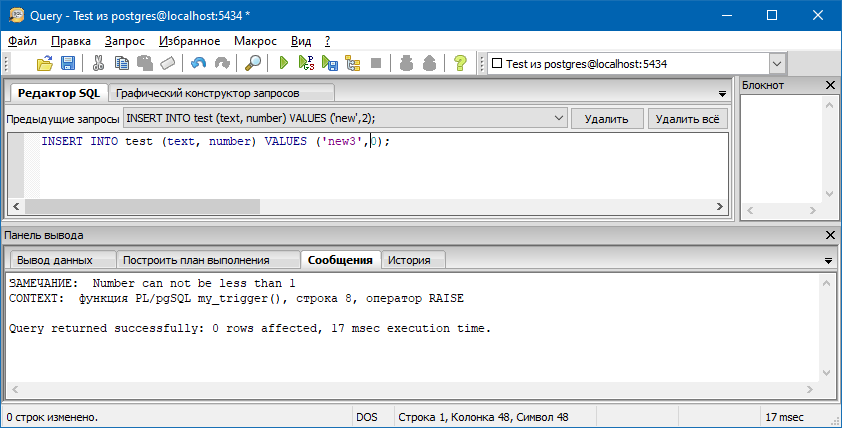
Як бачимо перед вставкою було виявлено, що значення поля text “new” вже було в таблиці та до нового значення була додана двійка

Подивимось таблицю логу



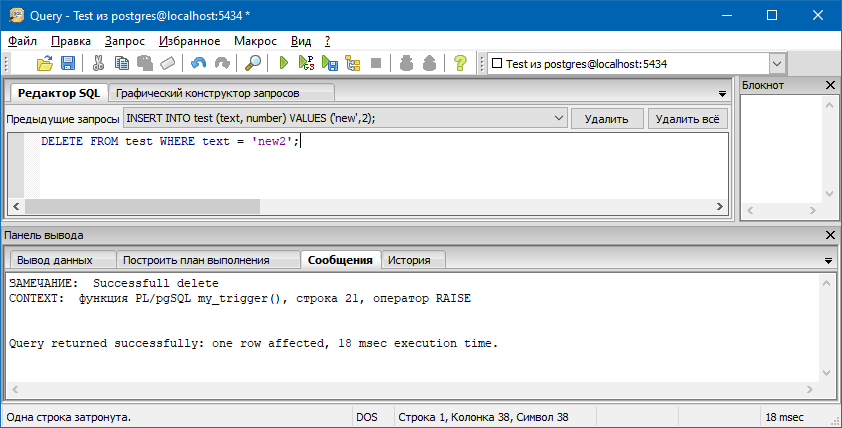
Бачимо, що лог був доданий

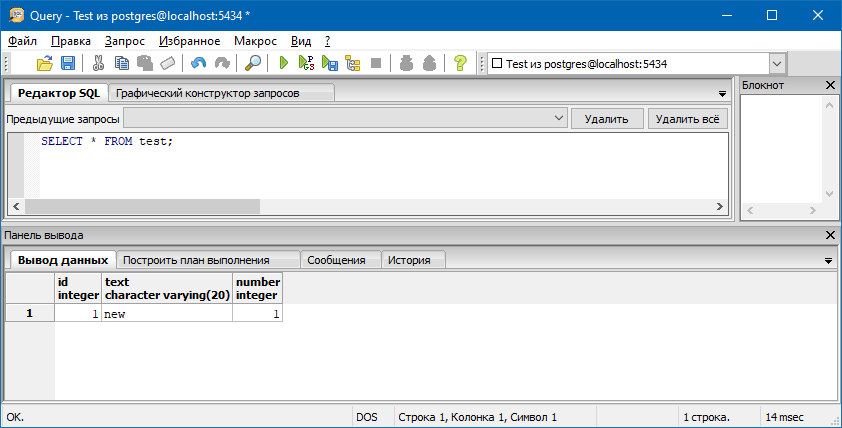
Тепер спробуємо ще раз додати нові дані



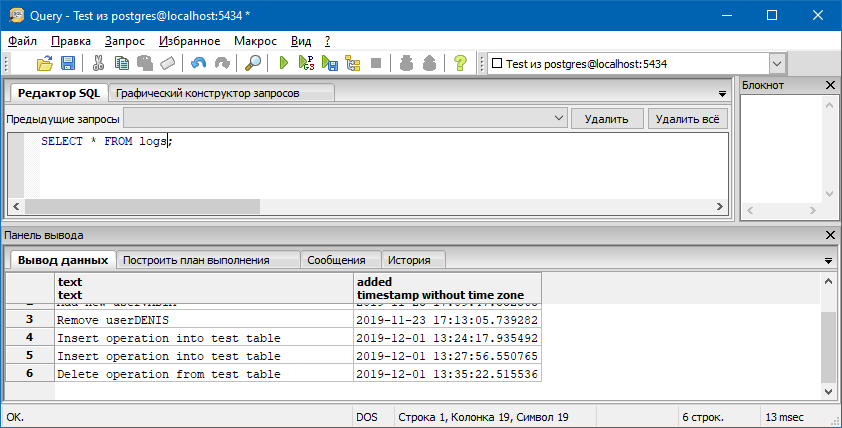
Бачимо, що вставки не було та виведено було відповідне повідомлення.

Тепер виконаємо операцію видалення.





Бачимо, що все добре. Подивимось знову таблицю logs.



Бачимо, що новий рядок був доданий.

Отже створений тригер працює правильно.

**ДОДАТКОВІ ПРИКЛАДИ**

**Дослідження індексів**

|  |
| --- |
| **Дослідження індексів 1** |
| *GIN*  Для дослідження індексу була створена таблиця, яка має дві колонки типу text. Вони проіндексовані як *GIN*. У таблицю було занесено 1000000 записів.  Виконуємо запити для текстового пошуку      У неіндексованій таблиці виконання запиту займає      У індексованій таблиці виконання запиту займає      *BRIN*  Для дослідження індексу була створена таблиця, яка має дві колонки: t\_data типу timestamp without time zone (дата та час (без часового поясу)) і t\_number типу integer. Колонка t\_data проіндексована як *BRIN.* У таблицю занесено 37843202.  Виконуємо запити для пошуку по інтервалу дати.    У неіндексованій таблиці виконання запиту займає.    У індексованій таблиці виконання запиту займає. |

|  |
| --- |
| **Дослідження індексів 2** |
| *BTree*  Для дослідження індексу була створена таблиця, яка має дві колонки: числову і текстову. Вони проіндексовані як *BTree*. У таблицю було занесено 1000000 записів.  Виконуємо запити для пошуку      У неіндексованій таблиці виконання запитів триває      У індексованій таблиці виконання запитів займає      *GIN*  Для дослідження індексу була створена таблиця, яка має дві колонки типу text. Вони проіндексовані як *GIN*. У таблицю було занесено 100000 записів.  Виконуємо запити для текстового пошуку      У неіндексованій таблиці виконання запитів триває      У індексованій таблиці виконання запитів займає |

**Створення тригерів**

|  |
| --- |
| **Тригер 1** |
| A screenshot of a social media post  Description automatically generated  *Принцип роботи*  Тригер спрацьовує при вставці чи оновлені у таблиці Chief. Якщо значення ім’я начальник, яке вставляється чи оновлюється, дорівнює test, то цей новий запис заноситься у таблицю логів Ch\_Log. Якщо значення ім’я не дорівнює test, то до кожного значення *l\_name* у таблиці Ch\_Log додається “AAA”.  *Результат роботи*  Вставляємо запис у Chief      Clog    Оновлюємо запис      Ch\_Log |

|  |
| --- |
| **Тригер 2** |
| *Принцип роботи*  Тригер спрацьовує після видалення з таблиці чи при оновлені у таблиці client. Якщо значення ідентифікатора запису, який видаляється або оновлюється, парне, то цей запис заноситься у додаткову таблицю client\_log. Також, з кожного значення name видаляються символи “х” на початку і кінці. Якщо значення ідентифікатора непарне, то до кожного значення *name* у таблиці client\_log додається “x” на початку і кінці.  *Результат роботи*  Видаляємо запис з client      client\_log    Оновлюємо запис      client\_log    Знову видаляємо запис      client\_log |