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: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав: студент 3 курсу

ФПМ групи КВ-82

Громко О.О

Перевірив: Павловський В.І.

Київ – 2020

**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).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

**Варіант 10**

У другому завданні проаналізувати індекси *Hash, BRIN*

Умова для тригера – *after delete, insert*

**Зміст**

1. [Завдання 1](#Завдання1)
2. [Функція для вставки у таблицю](#Функціядлявставки)
3. [Функція для модифікації даних](#Функціядлямодифікації)
4. [Функція для видалення даних](#Функціядлявидалення)
5. [Завдання 2](#Завдання2)
6. [тестовий запит на пошук записів за значення поля user\_id](#з21)
7. [Запит, з використанням предикату над індексом](#з22)
8. [Запит, з використанням сортування, яке потребує багато порівнянь об’єктів](#з23)
9. [Запит пошуку користувачів з однаковим полем user\_id](#з24)
10. [Створення та аналіз індекса BRIN](#з25)
11. [Запит, з використанням предикату над індексом](#з26)
12. [Запит, з використанням сортування, яке потребує багато порівнянь об’єктів](#з27)
13. [Запит пошуку користувачів з однаковим полем user\_id](#з28)
14. [Завдання 3](#з30)
15. [Команда створення тригеру](#з31)
16. [Виконаємо запит видалення з таблиці](#з32)
17. [Виконаємо запит додавання даних](#з33)

**Завдання 1**

Логічна модель БД «Онлайн кінотеатр»(Рис.1)

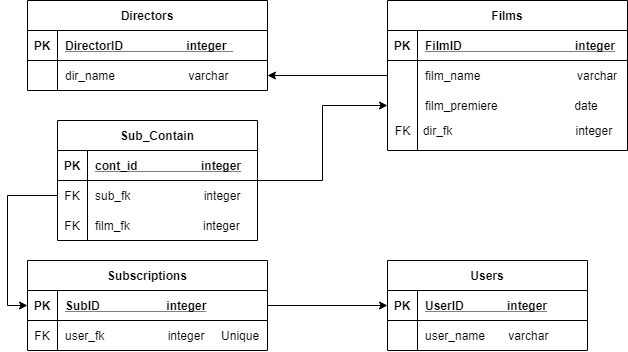
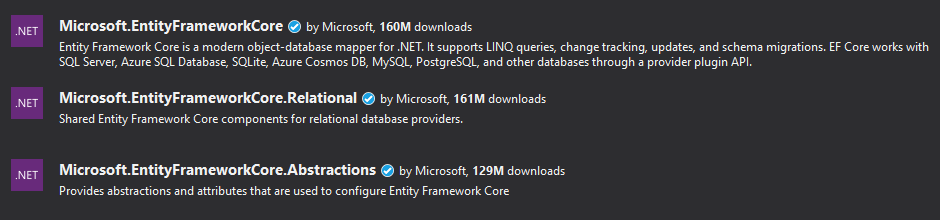
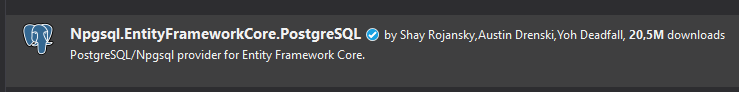
****

Рис.1 - Логічна модель БД «Онлайн кінотеатр»

Для перетворення модулю “Model”, створеної в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі використовуємо бібліотеку Entity Framework, так як робота початково була розроблена на мові C#. Для роботи були встановлені наступні пакети з репозиторію NuGet:



А також його розширення для роботи з PostgreSQL:



Ця бібліотека є дуже зручною в використані, бібліотеку для вирішення задач об’єктно-реляційного відображення. Вона інтегрується з іншими функціональними розширеннями мови, що робить роботу з даними ще більш зручним.

Концептуальна модель «Онлайн кінотеатру»(Рис.2)

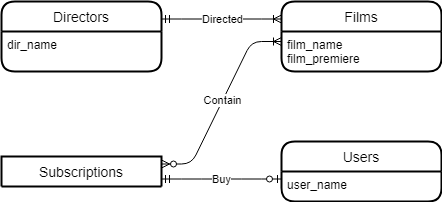
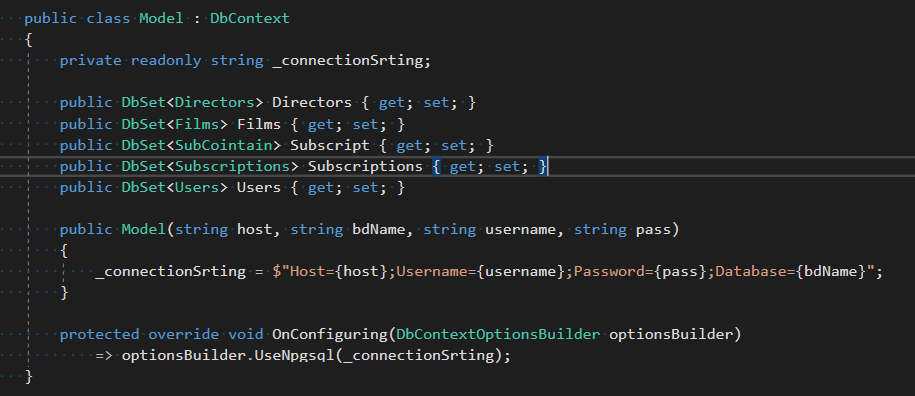
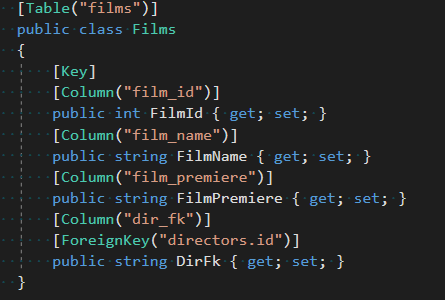
****

Рис.2 - Концептуальна модель «Онлайн кінотеатру»

Для роботи з базою даних, необхідно створити її контекст – це клас, який зберігає інформацію про всі її таблиці та дані, необхідні для підключення до СКБД:

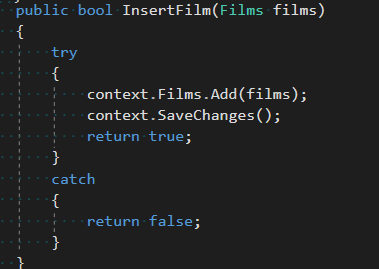


Кожна таблиця задається, як клас, у якому за допомогою атрибутів задаються зв’язки з БД, та інші необхідні параметри (продемонструємо код лише для однієї таблиці – Films) :

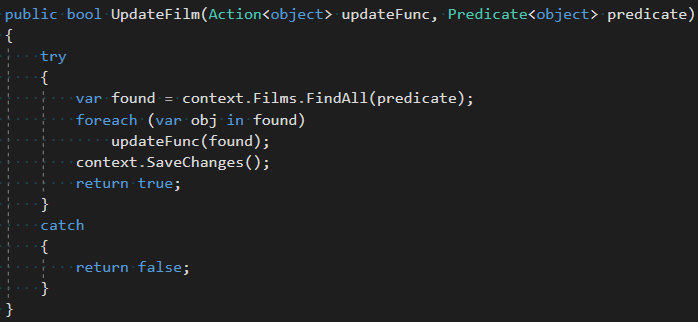


Далі, усі базові функції виконуються дуже просто – модифікацією полів контексту, та збереженням змін об’єкту. Бібліотека самостійно розуміє, що було змінено у таблиці та синхронізує зміни з віддаленим сервером.

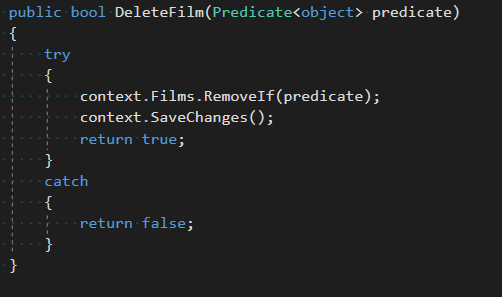
*Функція для вставки у таблицю:*



*Функція для модифікації даних:*



*Функція для видалення даних:*



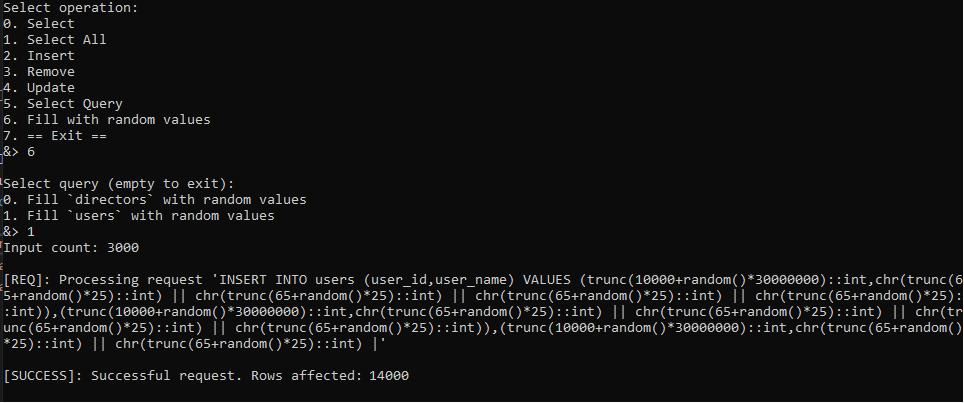
Інші класи та функції реалізовані аналогічно.

**Завдання 2**

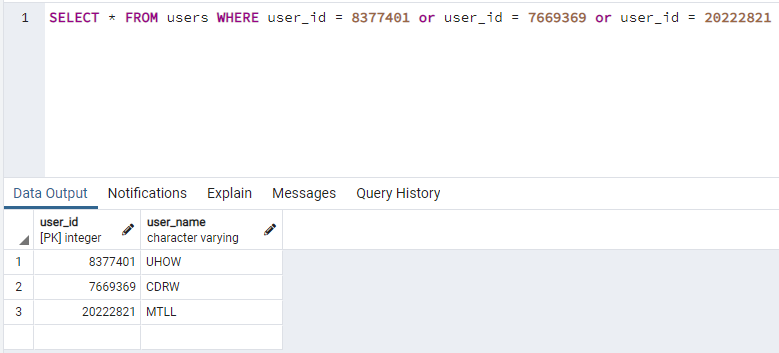
**Створення та аналіз індексу HASH**

**HASH** – це режим індексу, який автоматично застосовує хеш-функцію, до даних індексу. Хоча хешування і займає додатковий час, при великої кількості даних, це може значно прискорити виконання запитів, через те, що час доступу хеш-таблиці менший ніж у звичайних колекцій.

Для початку, додаємо до таблиці Users велику кількість випадкових даних, щоб запити не виконувалися миттєво, та щоб ми могли порівнювати різні режими роботи індексів. Зробимо це за допомогою нашої програми:



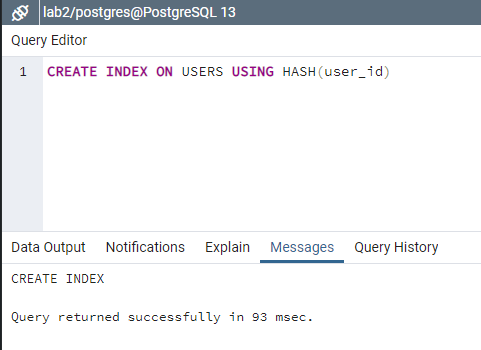
Виконаємо тестовий запит на пошук записів за значення поля user\_id.



Час, що зайняло виконання запиту:



Cтворимо індекс на полі user\_id, використовуючи хеш-функцію для індексації даних:



Виконаємо попередній запит ще раз, та порівняємо затрачений час:

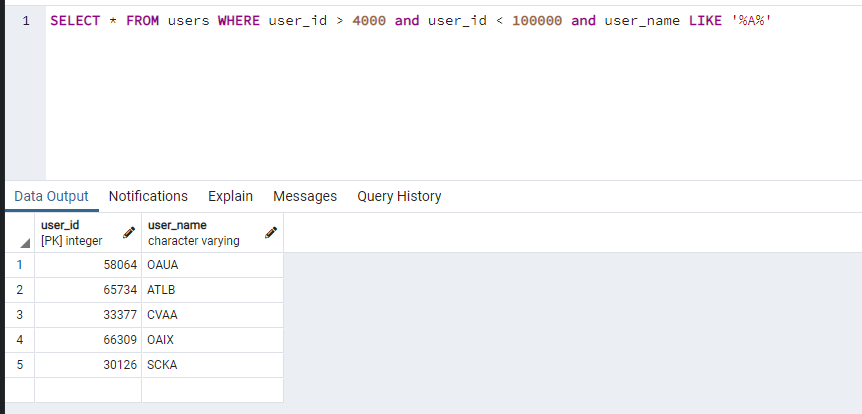


Повторне виконання запитів дає невелику похибку у 4-10 мілісекунд (у роботі будуть наведені лише по одному часу виконання на тест, проте, для того щоб упевнитися, що результат коректний, запит виконувався декілька разів).

Отриманий результат показує, що у даному випадку застосування хешування пришвидшило виконання запиту приблизно у два рази. Все через те, що пошук – найсильніша сторона цього методу індексування, через метод роботи хештаблиці.

**Наведемо ще декілька прикладів.**

*Запит, з використанням предикату над індексом:*



Час виконання без хешування:

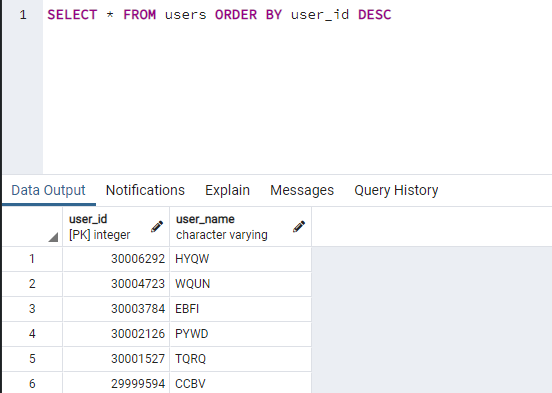


З використанням хешування:

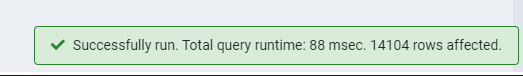


Значного прискорення у цьому випадку немає.

*Запит, з використанням сортування, яке потребує багато порівнянь об’єктів:*



Час виконання без хешування:

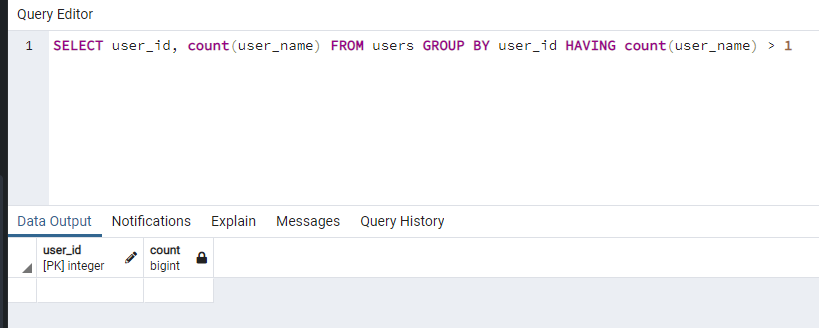


З використанням хешування:



Бачимо, що результат однаковий у межах похибки. Через те, що порівняння хешів, виконується однаково швидко, з порівнянням чисел індексів. Якби полем був рядок, ми б мали незначний приріст швидкодії.

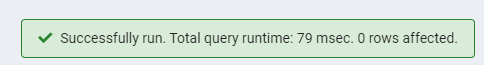
*Запит пошуку користувачів з однаковим полем user\_id:*



Час виконання без хешування:



З використанням хешування:



Бачимо приріст більш, ніж у два рази, через те, що тут виконується операція схожа на пошук. Тому хеш-таблиця дозволяє значно пришвидшити процес групування даних.

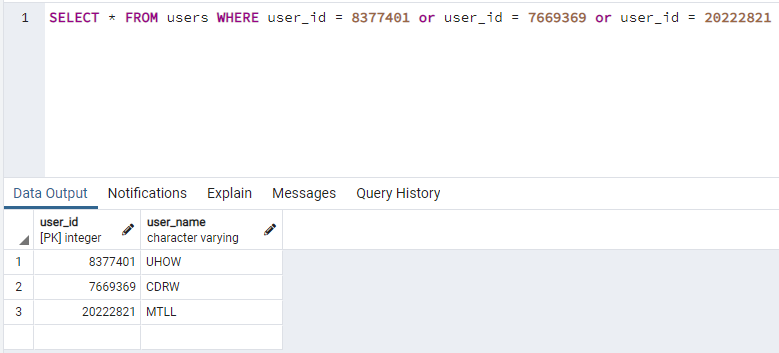
Наведені приклади кажуть, що хешування індексу ефективне, для виконання сортування, групування або пошуку даних (випадки, де використовується багато порівнянь поля), проте воно неефективне, для прямої роботи з даними.

Також, хешування займає значний час, тому цей спосіб буде ефективний, лише при великій кількості даних, та за умови, що індекси оновлюються не часто.

**Створення та аналіз індексу BRIN**

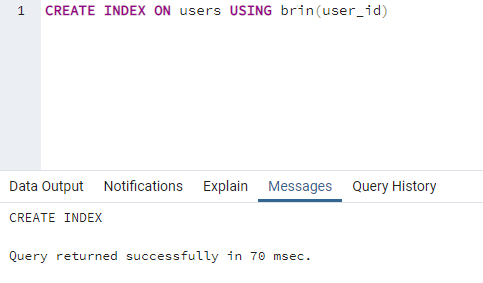
BRIN – це Block Range Index. Він працює добре для тих стовпчиків, де значення корелюють із їх фізичним положенням в таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання.

Тому будемо тестувати індекс BRIN на таблиці Users, як і для попереднього індексу, але по колонці email. Для великих обсягів даних, сортування рядків має прискорити пошук у декілька разів. Протестуємо це простим SELECT запитом з попередніх тестувань





Тепер створимо індекс.

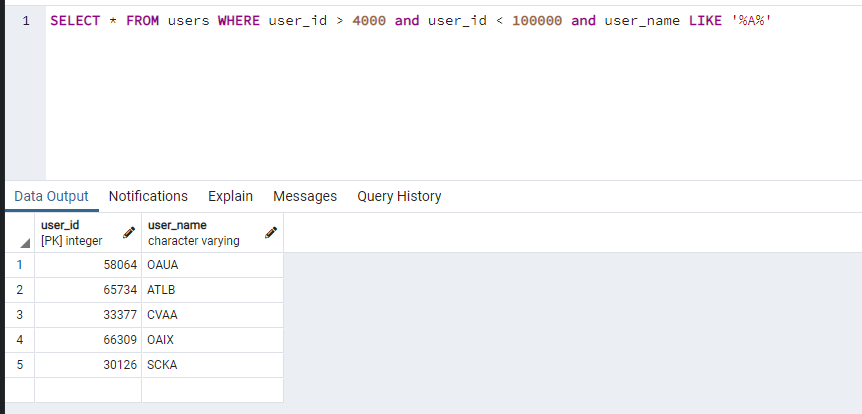


Результат виконання попереднього запиту з використанням індексу:



Бачимо значний приріст швидкодії, за рахунок впорядкування чисел у внутрішньому представленні бази даних.

*Запит, з використанням предикату над індексом:*



Час виконання без хешування:

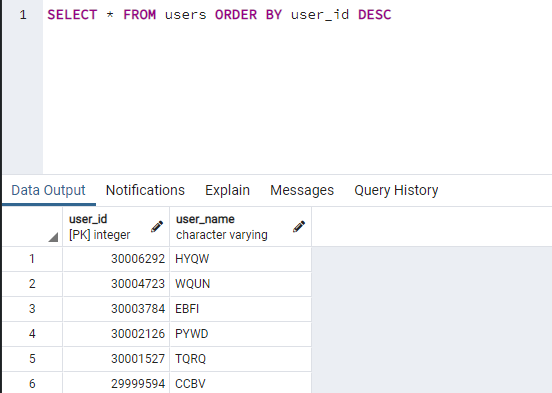


З використанням хешування:



Бачимо, що результат однаковий у межах похибки. Впорядкування індексів не може допомогти визначити положення елементу, що задовольняє предикатну функцію. Тому ми бачимо однаковий результат.

*Запит, з використанням сортування, яке потребує багато порівнянь об’єктів:*



Час виконання без хешування:

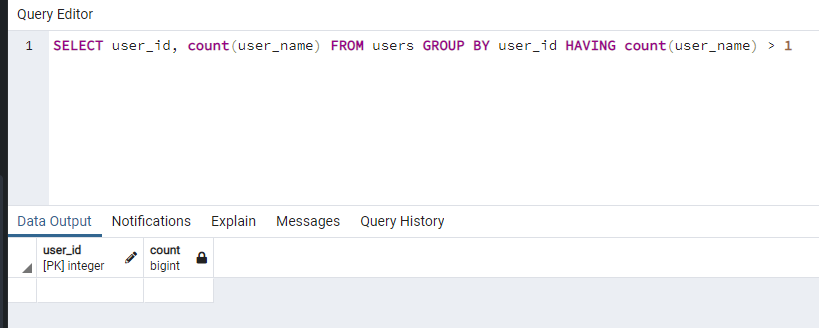


З використанням хешування:



.

*Запит пошуку користувачів з однаковим полем user\_id:*



Час виконання без хешування:



З використанням хешування:



У ситуаціях, коли необхідно пройти усі рядки таблиці (як і у тесті з предикатом), даний режим не може мати значного ефекту, через принцип роботи, тому і бачимо однаковий результат.

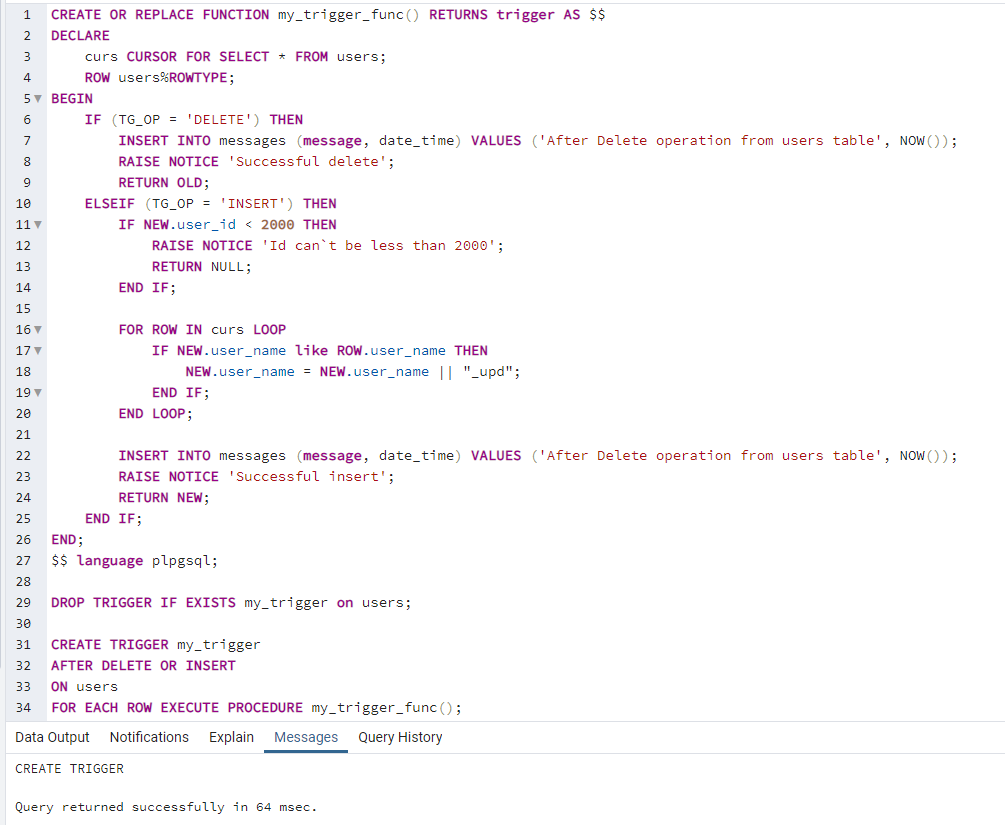
З наведених прикладів видно, що використання цього методу індексування може прискорити роботу деяких запитів пошуку, групування або сортування, проте на відміну, від хешування, цей спосіб не може уповільнювати запити, у деяких випадках використання (за виключенням операцій додавання операцій, при яких відбувається індексація даних).

**Завдання 3**

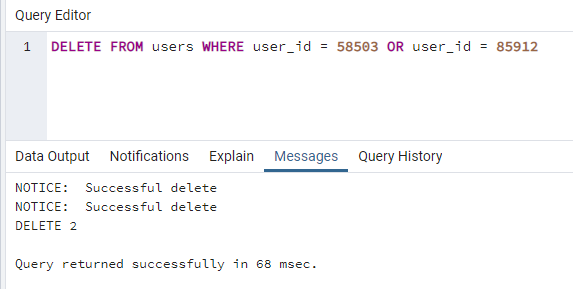
Для тестування тригерів створимо ще одну таблицю, для зберігання повідомлень від тригеру.



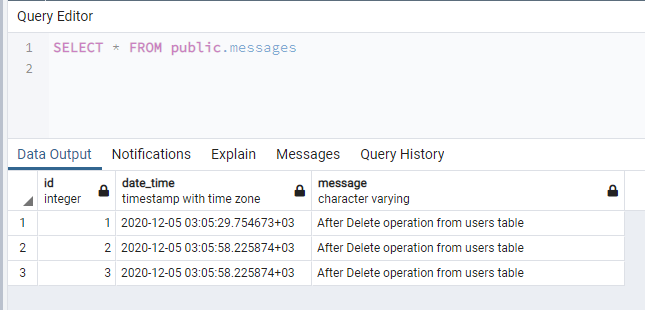
*Команда створення тригеру:*



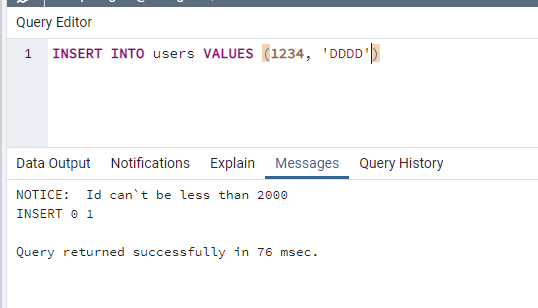
*Виконаємо запит видалення з таблиці:*



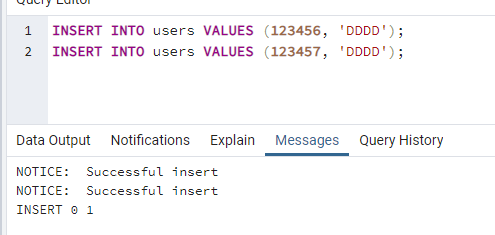
Бачимо повідомлення від тригеру. Зміст таблиці Messages також був змінений:



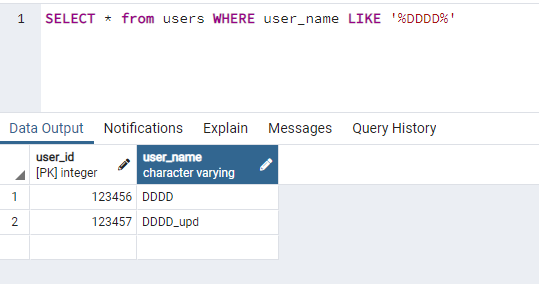
*Виконаємо запит додавання даних:*



Тригер повернув «Помилку» через те, що ми не виконали його умову додавання даних. Виконаємо запит, з збільшенням індексів:



Бачимо повідомлення від тригеру. Зміст таблиці users був змінений наступним чином:



Зміст таблиці Logs також був змінений:

