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 СКБД PostgreSQL»

Виконав: студент 3 курсу

ФПМ групи КВ-82

Іваненко Олександр

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.

*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.

Корисні посилання: [тут](https://www.learndatasci.com/tutorials/using-databases-python-postgres-sqlalchemy-and-alembic/) і [тут](https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/).

*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).

Корисні посилання: [Hash](https://habrahabr.ru/company/postgrespro/blog/328280/), [B-tree](https://habrahabr.ru/company/postgrespro/blog/330544/), [GIN](https://habrahabr.ru/company/postgrespro/blog/340978/), [BRIN](https://habrahabr.ru/company/postgrespro/blog/346460/).

*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 виключних ситуацій. Виконати лагодження тригера при різних вхідних даних та навести 2-3 приклади його використання.

Корисні посилання: [тут](https://www.enterprisedb.com/postgres-tutorials/everything-you-need-know-about-postgresql-triggers), [тут](https://www.postgresqltutorial.com/postgresql-triggers/).

*Вимоги до інструментарію*

1. Бібліотека для реалізації ORM - SQLAlchemy для Python або інша з подібною функціональністю;
2. Середовище для лагодження SQL-запитів до бази даних – pgAdmin 4;
3. СУБД - PostgreSQL 11-12.

**Варіант 12**

У другому завданні проаналізувати індекси *BTree, GIN*

Умова для тригера – *after update, insert*

**Меню для навігаціі**

1. [Завдання 1](#завд1)
2. [Завдання 2](#завд2)
3. [Завдання 3](#завд3)

**Завдання 1**

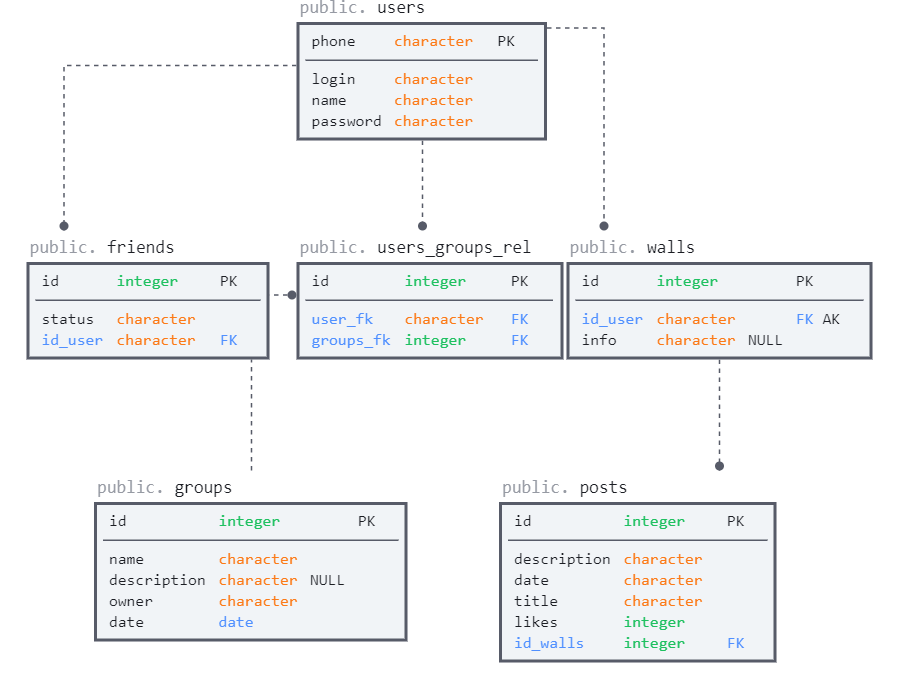
На рисунку 1 наведено логічну схему бази даних «Соціальна мережа».

Рисунок 1 – Логічна модель (Структура) БД “ Соціальна мережа ” (засобами SqlDMB)

Зобразимо на рисунку 2 сутнісні класи програми.

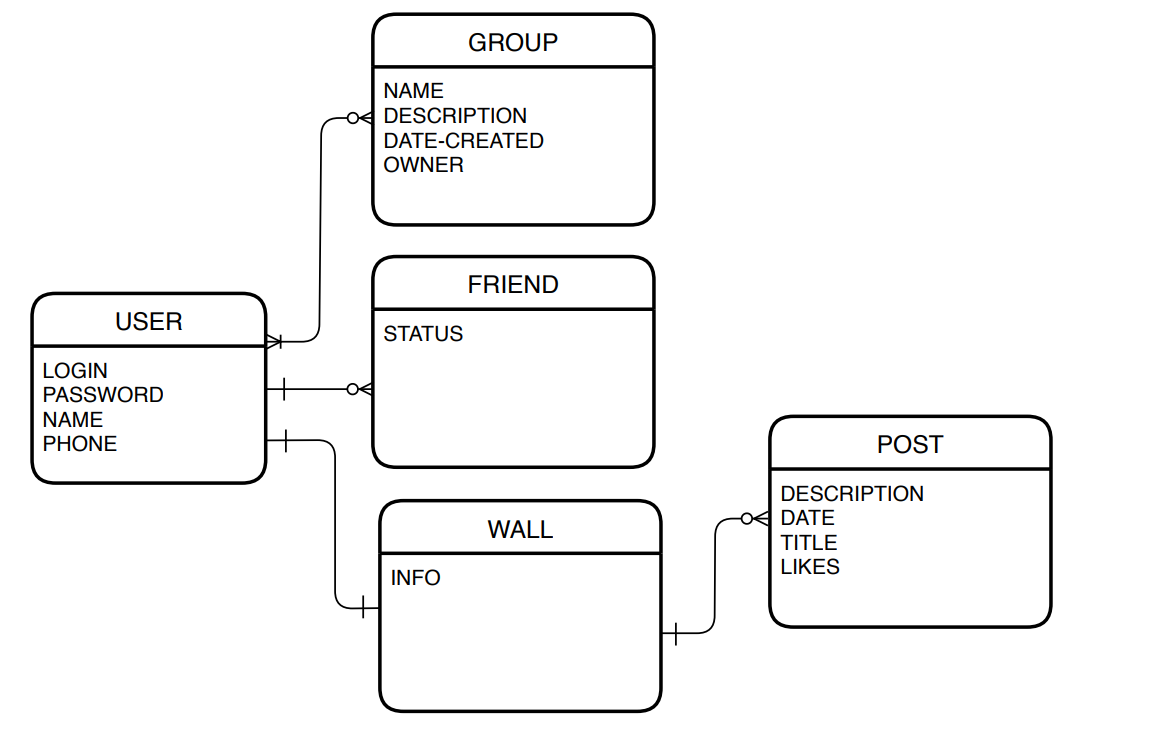
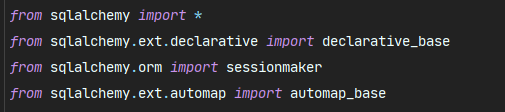
****

Рисунок 2 - Сутнісні класи програми.

Для перетворення модулю “Model” програми, створеної в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі використовую бібліотеку SQLAlchemy, яка є дуже зручною в використані, бібліотеку для вирішення задач об’єктно-реляційного відображення. Вона працювати з великою кількістю СКБД, у тому числі і PostgreSQL.

Для роботи з базою даних були підключені наступні залежності:

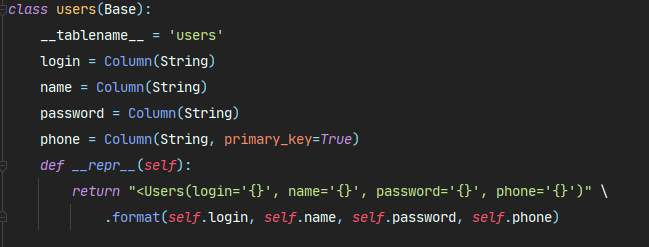


Для того щоб виконувати якісь дії над таблицями потрібно створити об’єкт Session, який необхідно відправляти для змін, щоб зберегти усі зміни, що були виконані в з минулого відправлення. Основний клас Database з внутрішнім об’єктом Session.



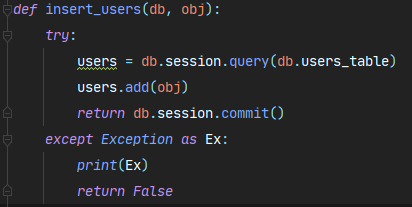
Представлення таблиць у класах полягає у представленні полів класів як колонок таблиці. Для цього потрібно враховувати відповідність типів в таблицях, та в полях класів.

Реалізація таблиці users:

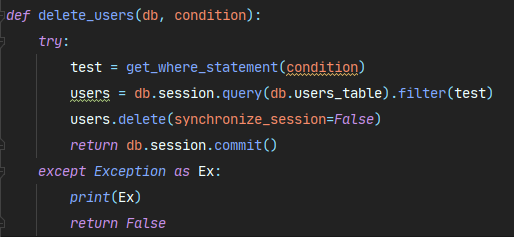


Інші класи таблиць реалізовані аналогічним образом.

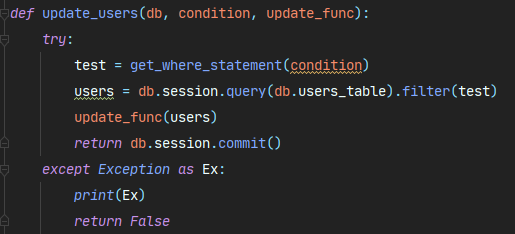
Функція додавання записів до таблиць:



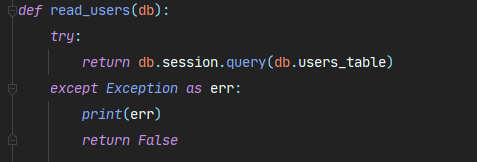
Функція видалення записів з таблиць:



Функція редагування записів в таблицях:



Функція читання з таблиць:



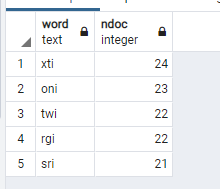
**Завдання 2**

**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.

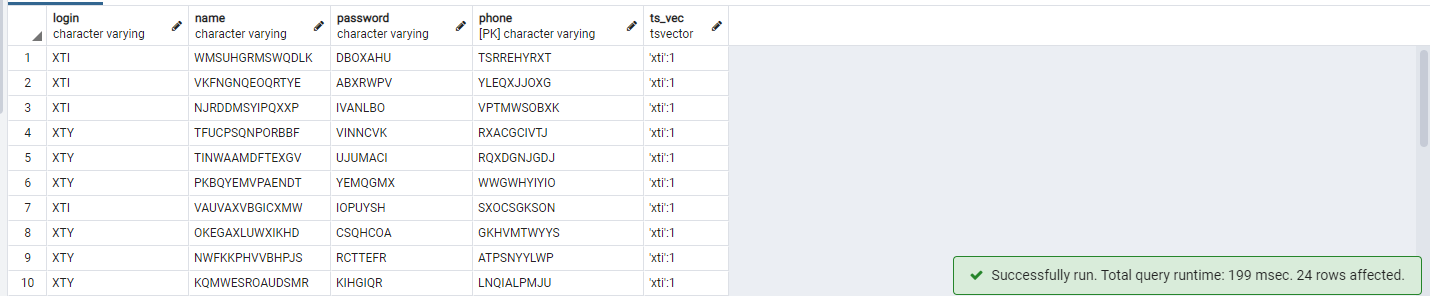
Додамо до таблиці users колонку ts\_vec с типом tsvector та заповнимо його за допомогою функції to\_tsvector, який буде приймати як аргумент значення колонки login . Саме на колонці ts\_vec будемо створювати індекс, але спочатку подивимось на швидкість повнотекстового пошуку без індекса.

Дізнаємось, які лексеми найчастіше зустрічаються в таблиці:



Виконаємо повнотекстовий пошук по слову ‘xti’ без індексу:



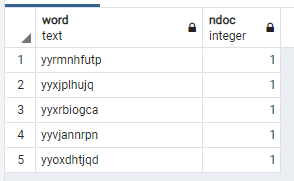




Бачимо що на пошук та відображення, середній час - 196 мс.

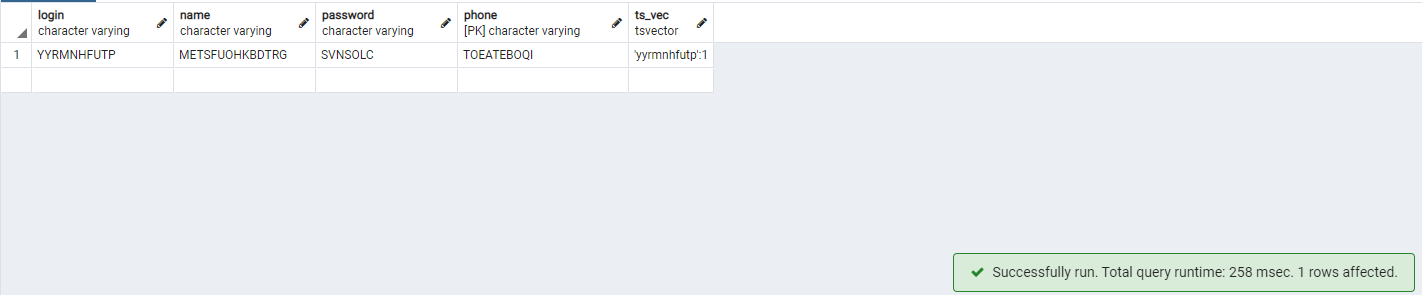
Дізнаємось які лексеми використовуються рідше:





Виконаємо повнотекстовий пошук по слову 'yyrmnhfutp’ без індексу:







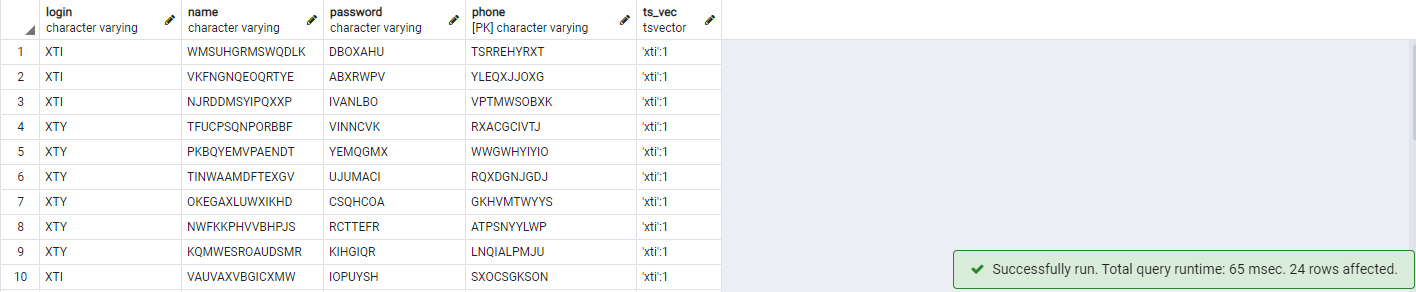


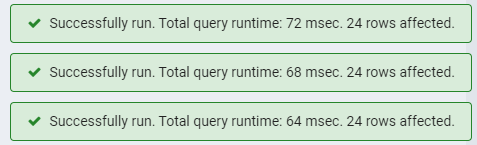
Бачимо що на пошук та відображення, середеній час - 317 мс.

Тепер створимо індекс GIN на колонці ts\_vec:



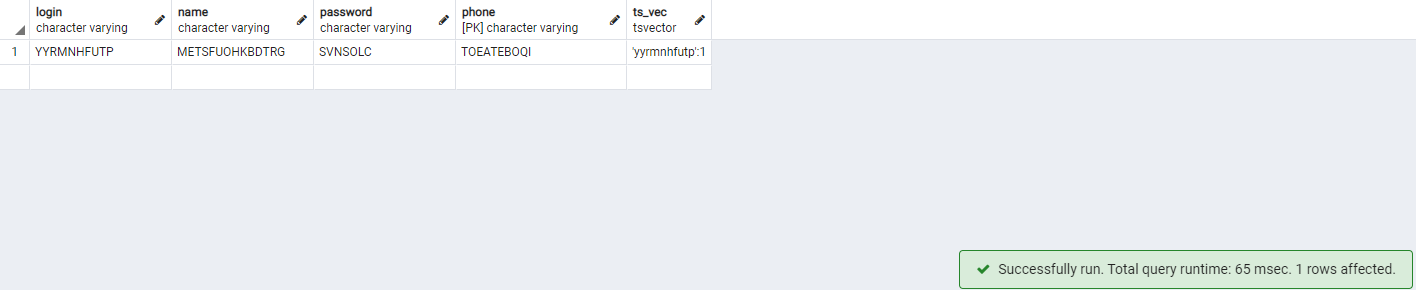
Виконаємо ті самі запити але тепер з індексом:

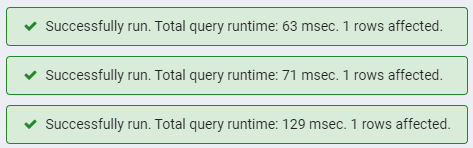




Бачимо що на пошук та відображення, середній час - 67 мс.







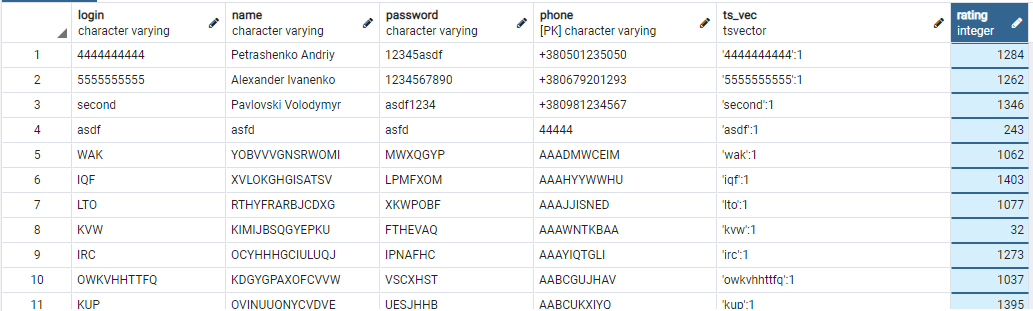
Бачимо що на пошук та відображення, середній час - 82 мс.

При порівнянні часу виконання з індексом та без індексу, індекс прискорив виконання запиту на частіших лексемах в 2.92 рази, а виконання запиту на рідкіших лексемах в 3.86 разів. Отже, індексування GIN значно прискорює повнотекстовий пошук та виконання запиту.

**Створення та аналіз індекса BTree**

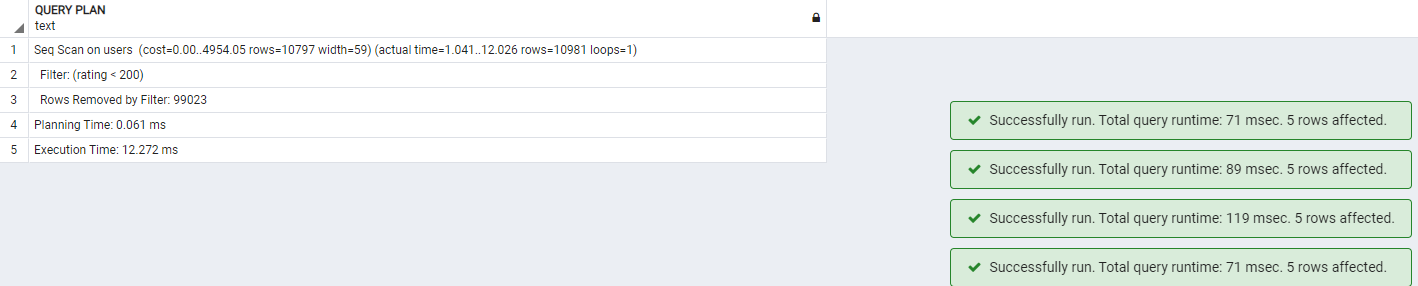
Індекс **BTree**, він же B-дерево, придатний для даних, які можна відсортувати. Іншими словами, для типу даних повинні бути визначені оператори «більше», «більше або дорівнює», «менше», «менше або дорівнює» і «дорівнює». Слід зазначити, що одні й ті ж дані іноді можна сортувати різними способами, що повертає нас до концепції сімейства операторів.

Для спрощення тестування індекса BTree додамо в таблицю “users” колонку rating з числовим типом (оскільки числа легше сортувати).



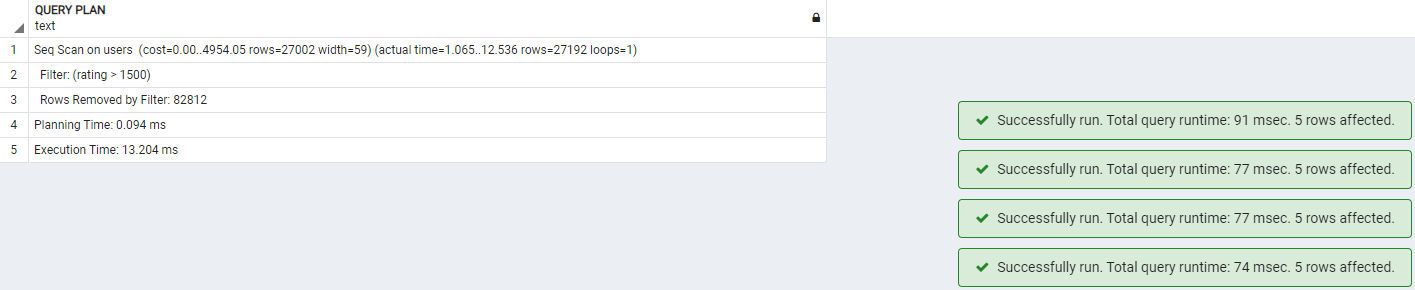
Виконаємо пошук по колонці rating з різними умовами без використання індексу BTree:





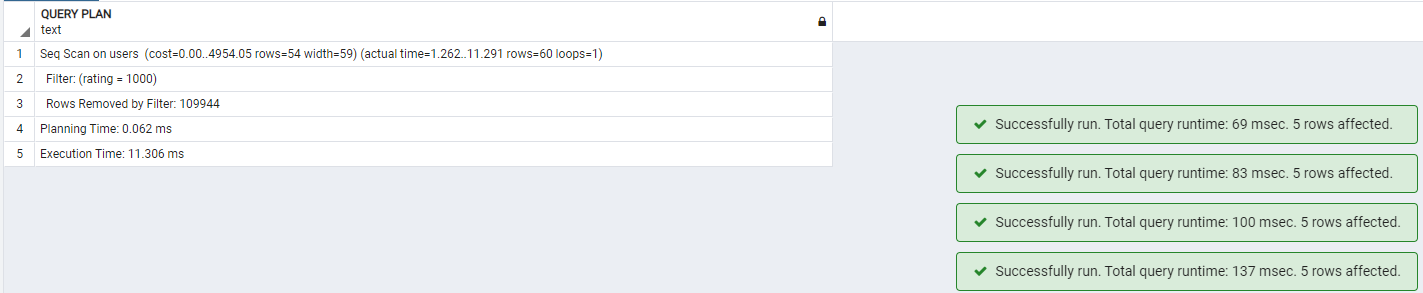
Як бачимо середній час виконання запиту – 85 мс.





Як бачимо середній час виконання запиту – 79 мс.





Як бачимо середній час виконання запиту – 96 мс.

Додамо індекс BTree



Виконаємо пошук по колонці rating з різними умовами з використанням індексу BTree:





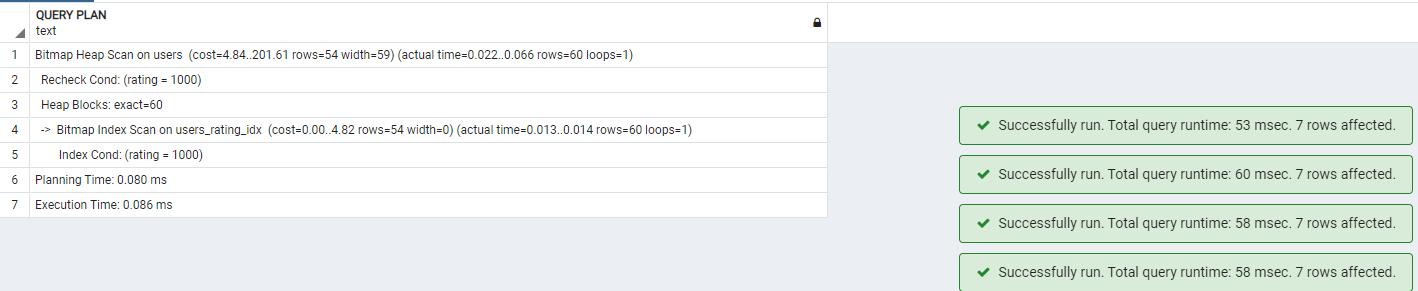
Як бачимо середній час виконання запиту – 70 мс.





Як бачимо середній час виконання запиту – 71 мс.

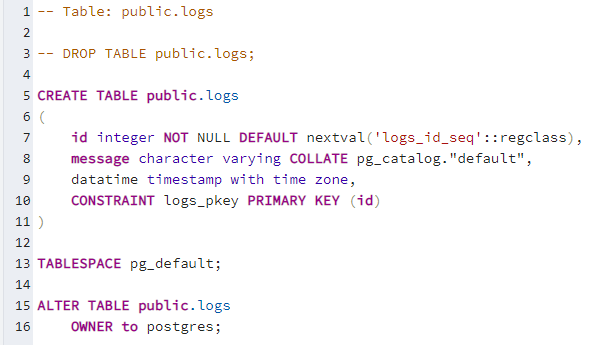


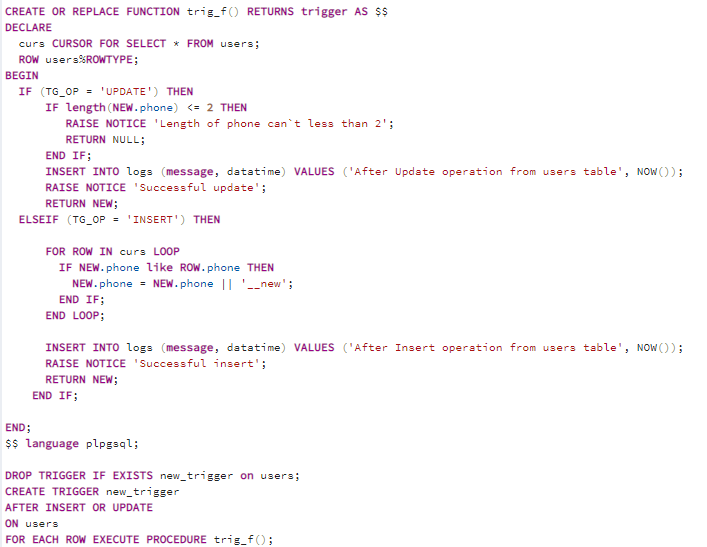


Як бачимо середній час виконання запиту – 56 мс.

Як видно, після додавання індексу BTree з’явилося невелике прискорення. Приблизно в 1.2 рази. Швидкість зростає через внутрішню структуру індексу, як збалансованого дерева.

**Завдання 3**

Для тестування тригерів створимо ще одну таблицю, для зберігання повідомлень від тригеру (logs)

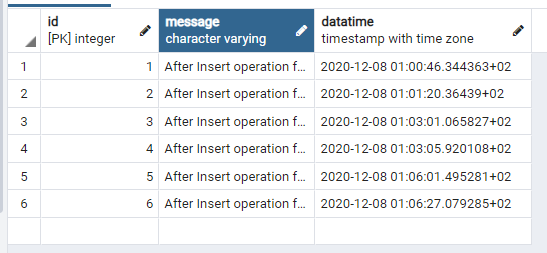
Створимо тригер:

Виконаємо запит вставки в таблицю users:



Є повідомлення від тригеру з успішною вставкою.

Також зміст таблиці logs змінився:



Виконаємо запит update:



Є повідомлення від тригеру з успішним оновленням.

Також зміст таблиці logs змінився:

