

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 3 курсу

ФПМ групи КВ-82

Ященко Іван Васильович

Перевірив: Павловський В.І.

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, 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.

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

Варіант 27:

|  |  |  |
| --- | --- | --- |
| ***27*** | ***GIN, BRIN*** | ***after update, insert*** |

**Меню для навігації**

Оглавление

[Завдання 1 6](#_Toc57893607)

[Завдання 2 10](#_Toc57893608)

[Завдання 3 20](#_Toc57893609)

**Модель бази даних**

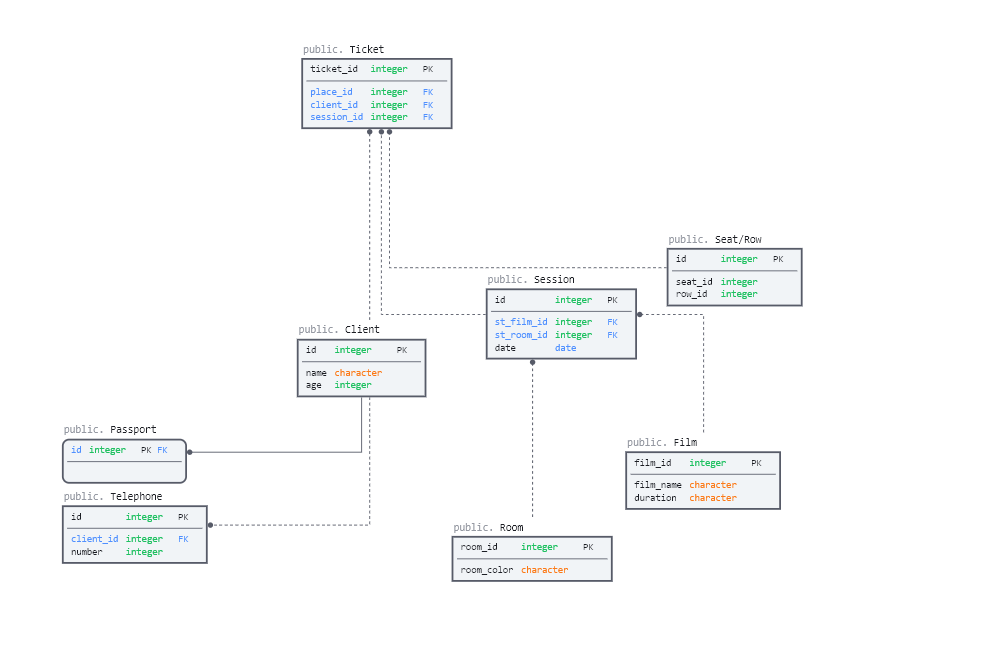
На Рисунку 1 наведено структуру бази даних яка використовується в даній роботі. Суттєвих змін, в порівнянні з 1та 2 Лабораторною роботами не відбулося. 

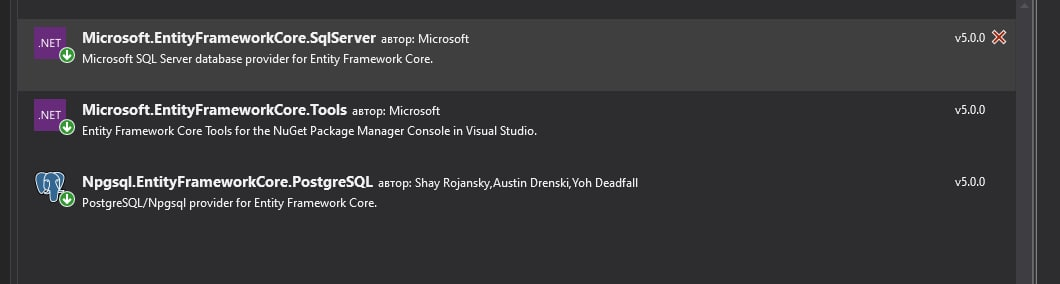
Рисунок 1 – Логічна модель (Структура) БД “ Сервіс продажу квитків кіно ” (засобами SqlDMB)

# Завдання 1

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувалась мова програмування C# та IDE Visual Studio 2019. Для підключення до серверу бази даних PostgreSQL використовувався пакет Npgsql. Разом з ORM Microsoft EntityFrameworkCore спеціально для пакету баз даних PostgreSQL, задля реалізації оптимізованих звернень до бази даних.

**Додаткові пакети данних Visual Studio**

Рисунок 3 ‑ Фотографія екрану, де описано підключення до БД засобами Npgsql

**Підключення до БД**

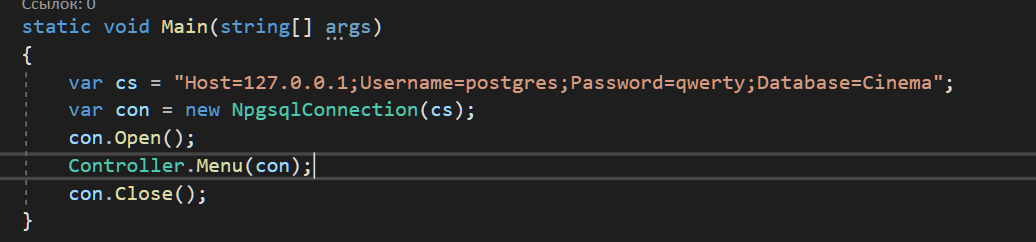


Рисунок 3 ‑ Фотографія екрану, де описано підключення до БД засобами Npgsql

**Структура програми**

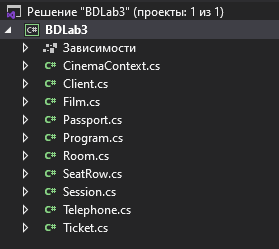


Рисунок 3 – Структура програми

**Опис структури програми:**

Програма містить 10 модулів. Кожна таблиця, що наявна в БД, була перероблена в окремий модуль-клас. Також тут ми маємо модуль CinemaContext, де описані взаємозв’язки БД, і модуль Program де реалізовані функції , додавання,вилучення та редагування даних, а також main функція програми.

Продемонструємо код лише для одного класу Client

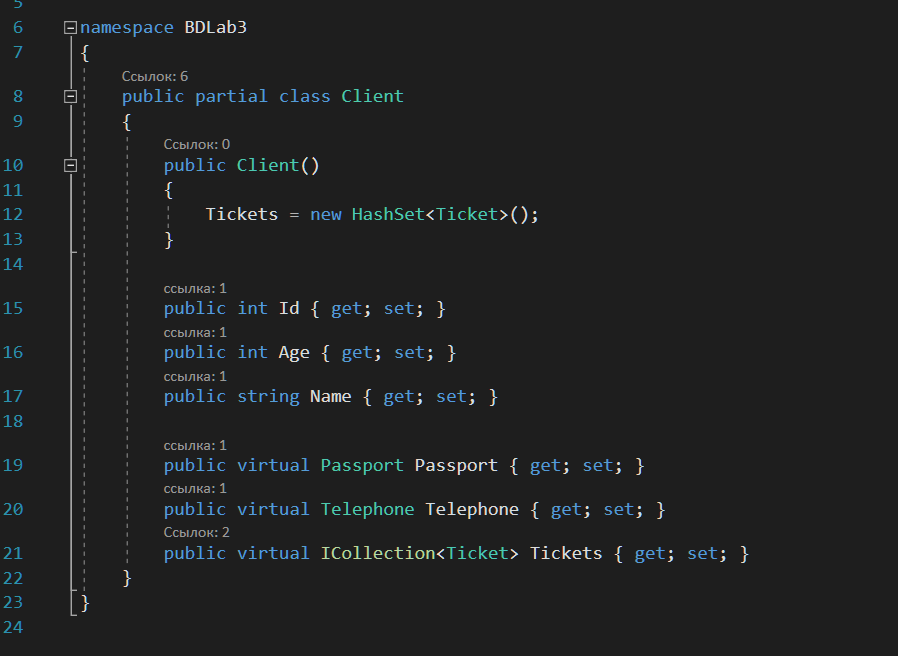
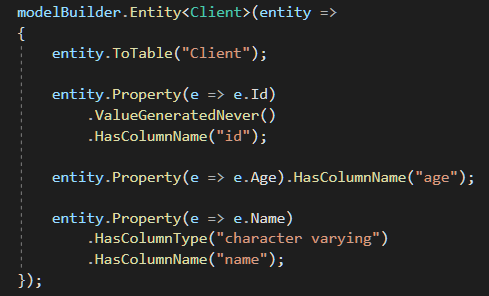
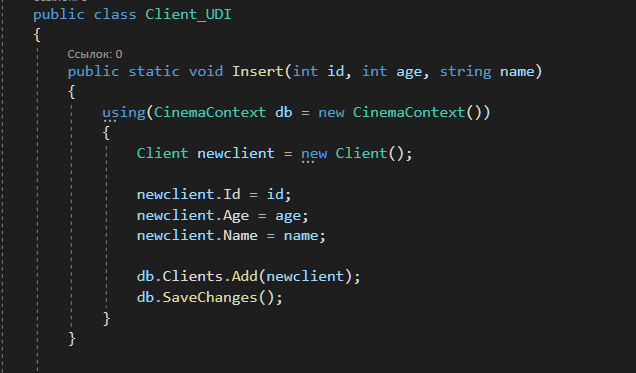


Рисунок 4 – Клас Client

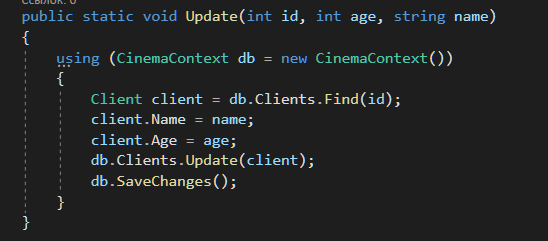
Виконуємо синхронізацію з даними із БД



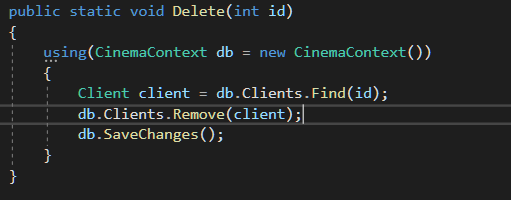
Функція додавання даних класу з таблицею в БД:



Функція оновлення даних класу з таблицею в БД:



Функція видалення даних класу з таблицею в БД:



По аналогії створені інші функції додавання,оновлення та видалення інших класів.

# 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. Створимо вибірку згенерованих тестових рядочків у таблиці inform типу tsvector, і згенеруємо 500000 рядочків.

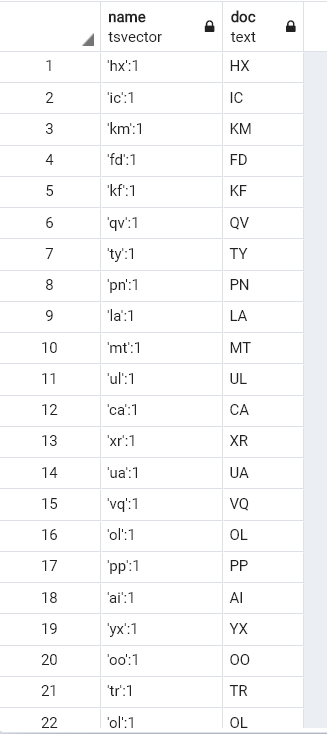


Рисунок 5 – Згенерована таблиця

Тепер проаналізуємо частоту повтору

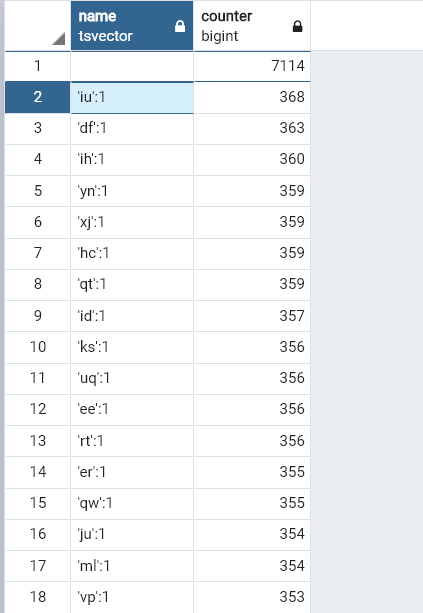
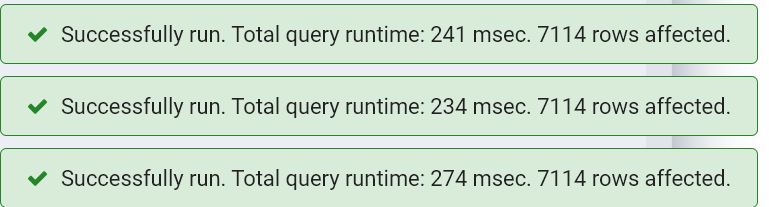


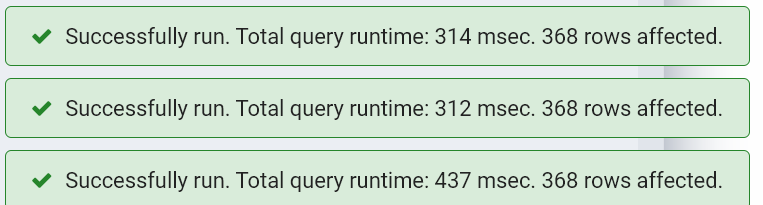
Рисунок 6 – Частота повтору даних згенерованої таблиці

Найчастіше тут зустрічається комбінація “”(пустий рядок), а рідше за все буде зустрічатись рядок name1, бо був заданий вручну.

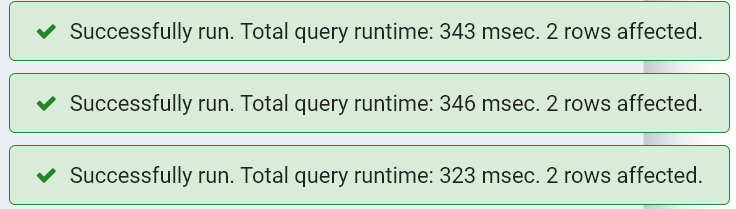
Тепер перевіримо пошук цих рядків без використання індексів:



Середній час =249 msec

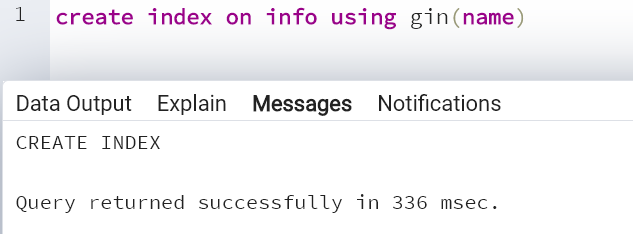


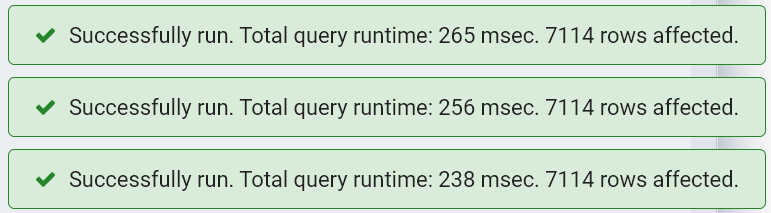
Середній час =354 msec

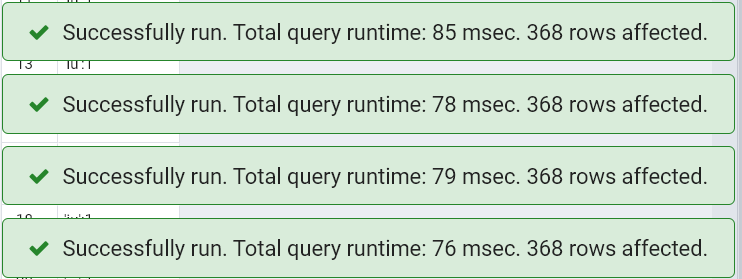


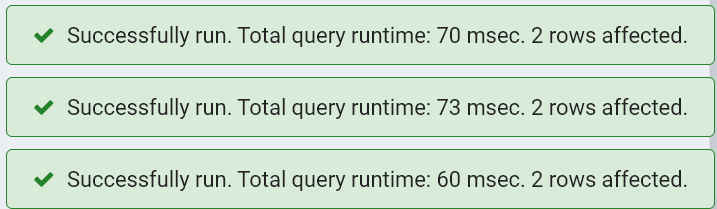
Середній час =337 msec

Тепер створимо GIN індекс:



Виконаємо ті ж самі запити:Середній час =253 msec

Середній час =79.5 msec

Середній час =68 msec

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 4-5 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, де зберігабються часто-змінні дані не рекомендовано використовувати індекс GIN, 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

Створимо таблицю numbers з полем date1 де згенеруємо 500000 міток часу.

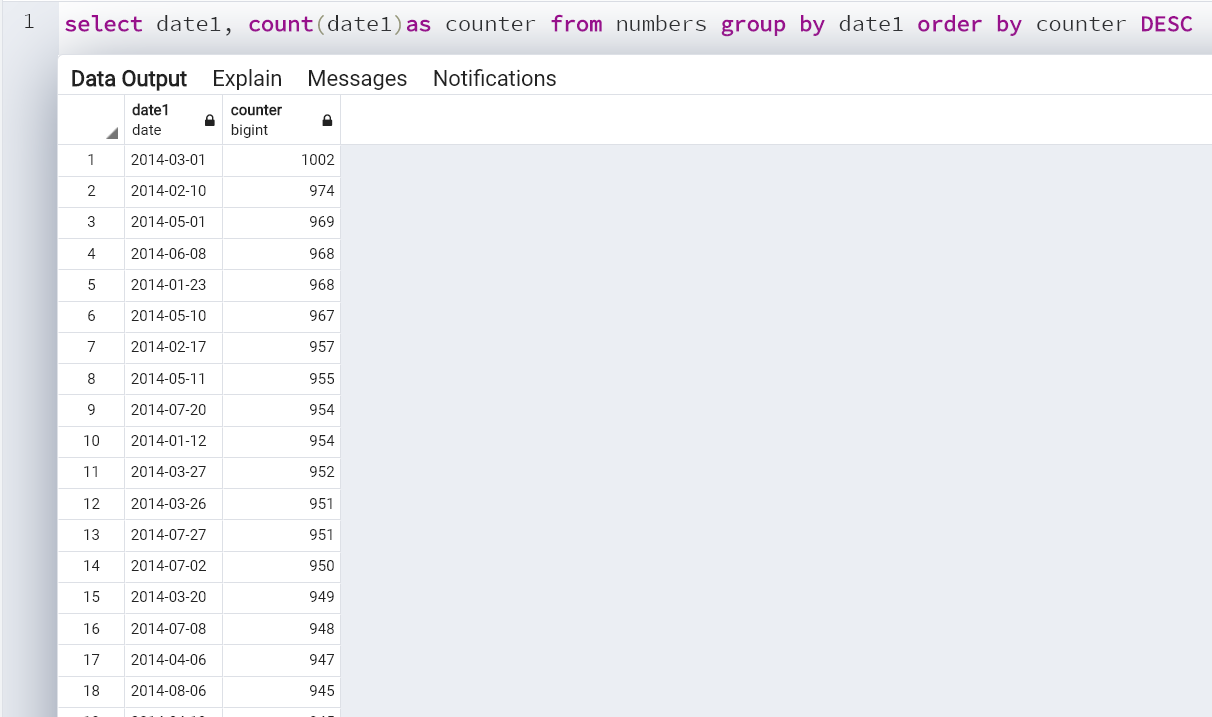
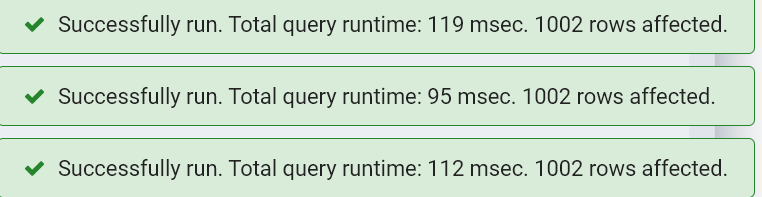


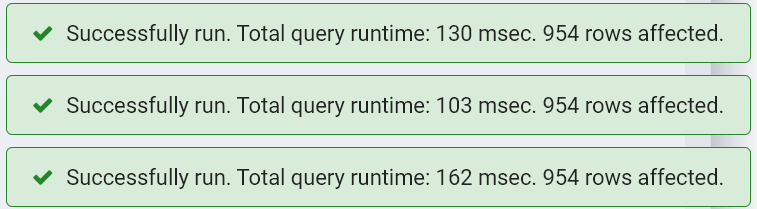
Рисунок 7– Частота повтору даних згенерованої таблиці

Дані у стовпчику date є не впорядкованими, тому робота індексу повинна бути не коректною, перевіримо це.

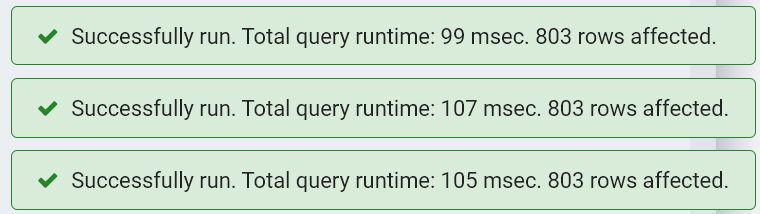
Тепер протестуємо дані без використання індексу:’



Середній час =109 msec

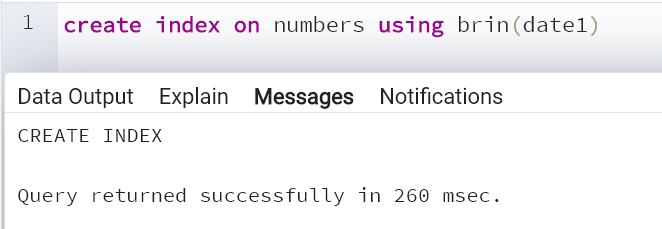


Середній час =131 msec

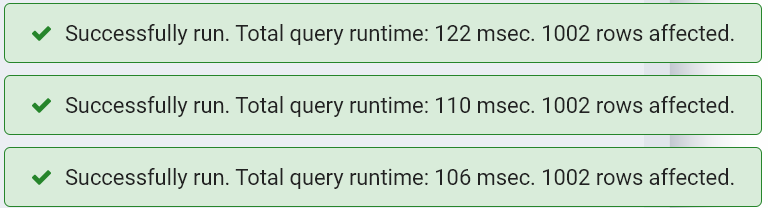


Середній час =104 msec

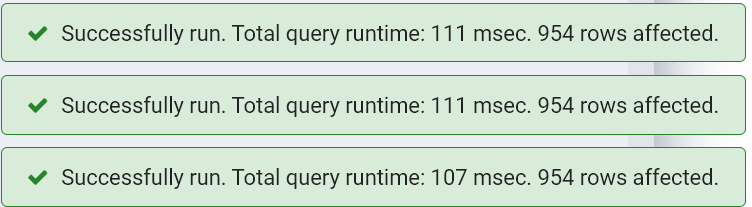
Тепер створимо індекс:



Виконаємо ті ж самі запити:



Середній час =113 msec



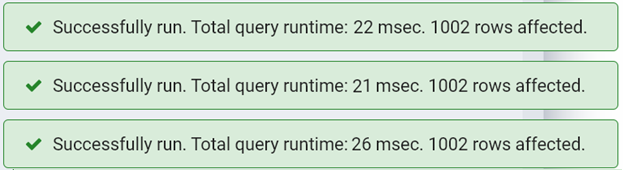
Середній час =109 msec



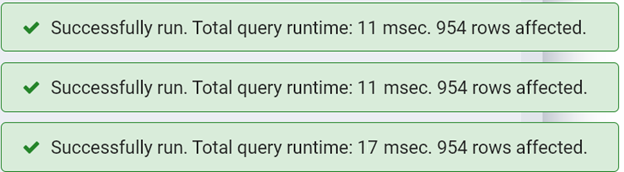
Середній час =109 msec

Як ми можемо побачити із-за невідсортованості даних маємо незмінний результат, з-за внутрішньої будови індекса.

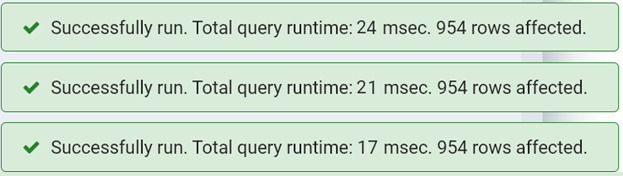
Тепер відсотуємо данні і перевіримо роботу індексу:



Середній час =23 msec



Середній час =13 msec



Середній час =20 msec

Як можемо побачити, робота індекса оптимізувалась і працює коректно.

Для таблиць без кореляції певних стовпчиків із фізичним положенням даний індекс не буде добре працювати по причині внутрішньої реалізації даного індексу. Як ми і побачили на попередньому прикладі.

# Завдання 3

Команда створення тригеру(функції тригера)

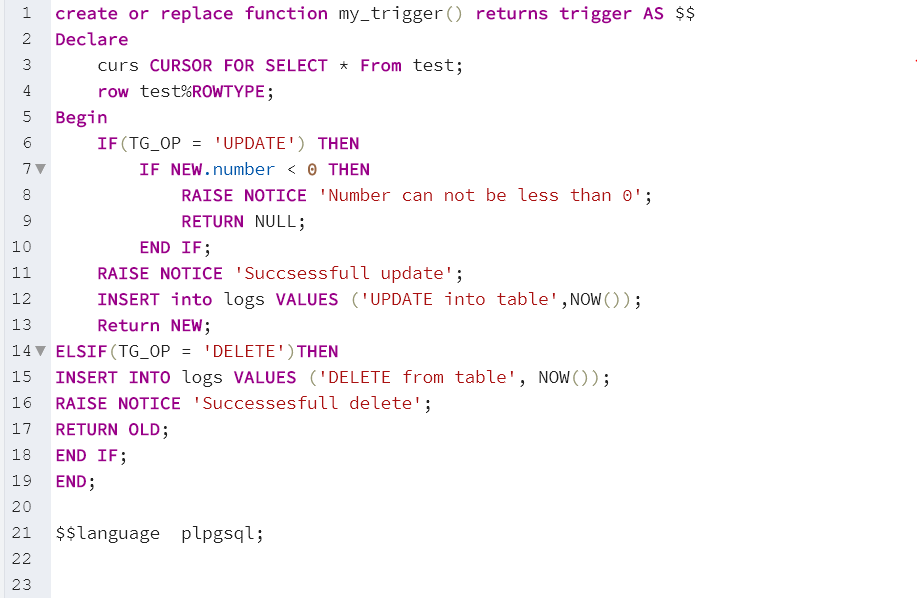
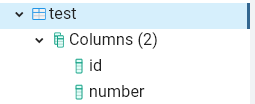


Рисунок 8– Функція тригеру

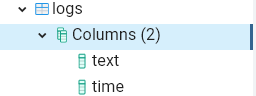
І підключимо тригер:



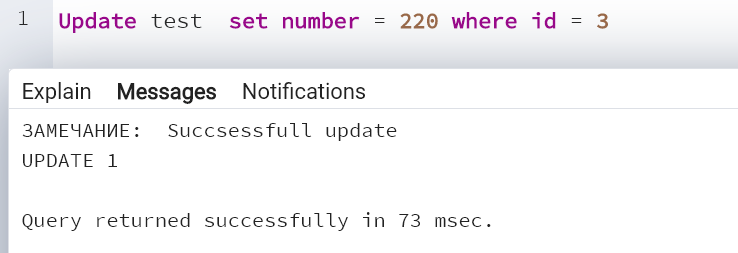
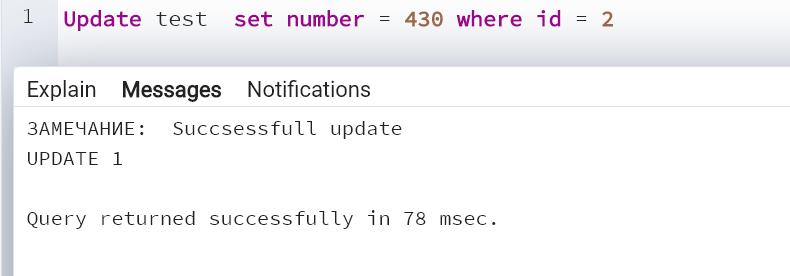
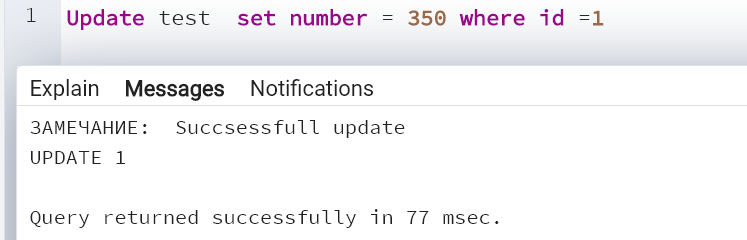
Тепер перевіримо працездатність тригера. Створимо таблицю test з полями id і number, куди додамо 3 рядочки .

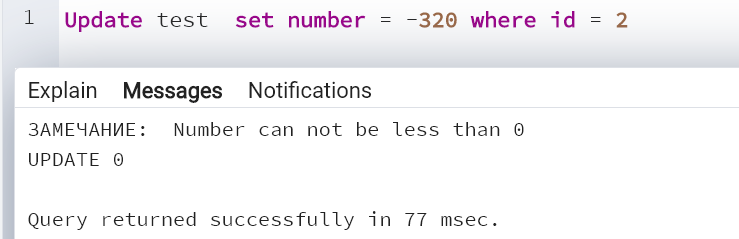
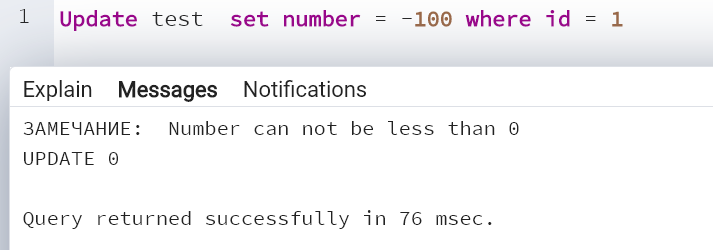


І створимо таблицю logs куди будуть записуватись усі успішно оброблені тригером записи.



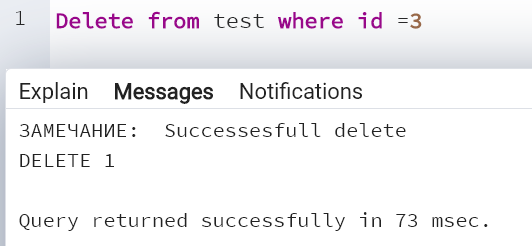
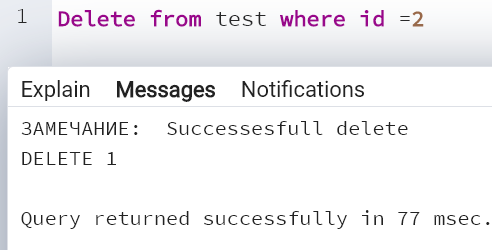
Тепер оновимо ці рядочки і перевіримо роботу тригера.



Як можемо побачити тригер спрацював на визовах UPDATE. Тепер спробуємо оновити рядочки значеннями num < 0, і перевіримо вставлену умову в тригері:

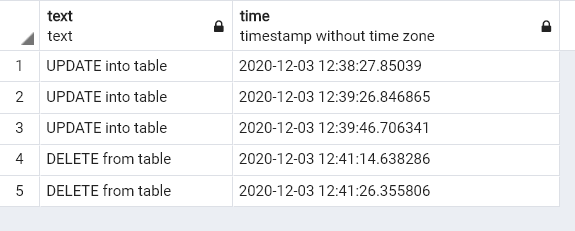
Як бачимо умова тригера спрацювала і рядочки не оновились, так як вони < 0.

Тепер спробуємо видалити рядочки:

Як бачимо тригер спрацював.

Тепер подивимся таблицю logs:



Отже тригер працює на командах UPDATE і DELETE, а також я додав умову, що якщо оновлене значення < 0 тоді тригер не оброблює цю операцію і повертає NULL, і виводить відповідне сповіщення користувачу.