

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

ФПМ групи КВ-84

Котлярський Алекс Олександрович

Перевірив: Петрашенко А.В.

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.

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 відлагодження тригера при різних вхідних даних, навівши 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](https://www.sqlalchemy.org/) або інша з подібною функціональністю.
2. Середовище для відлагодження SQL-запитів до бази даних– pgAdmin 4.
3. СУБД - PostgreSQL 11-12.

***Варіант:***



**Меню для навігації**

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

**Завдання 1**

На рисунку 1 наведено логічну модель бази даних «Футбольний Клуб»

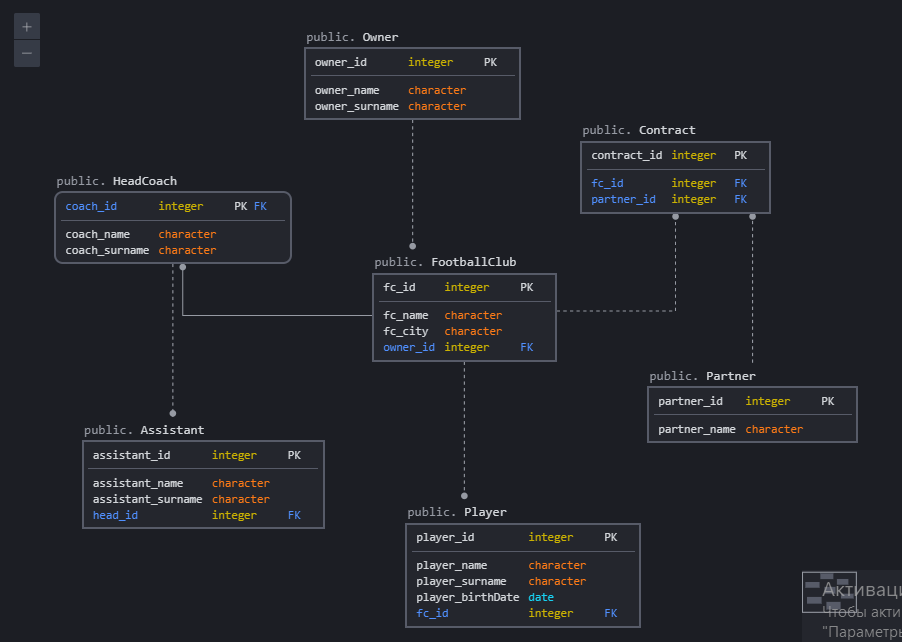


Рисунок 1 – Логічна модель бази даних «Футбольний Клуб»

Програма створена за допомогою мови програмування Python в середовищі розробки PyCharm Community Edition 2020.2.3

Для вирішення задачі перетворення моделі з MVC у вигляд об’єктно-реляційної проекції (ORM) використовується бібліотека SQLAlchemy для мови програмування Python, яка є однією з найпопулярніших для вирішення задач об’єктно-реляційного відображення.

Представлення таблиць у класах полягає у представленні полів класів як колон таблиці. Для цього потрібно врахувати відповідність типів даних між сервером PostgreSQL і відповідною мовою програмування. Наприклад, типам даних *date* та *time without time zone* PostgreSQL відповідає один тип даних – *DateTime*.

Для реалізації зовнішнього ключа у класі, який відповідає таблиці із зовнішнім ключем з відношенням до колонки таблиці А, потрібно додати поле типу класу, який відповідає таблиці А. Також потрібно написати відповідну команду для визначення цього поля як зовнішній ключ.

На рисунку 2 зображена структура класів програми, що відповідають таблицям БД «Футбольний Клуб»

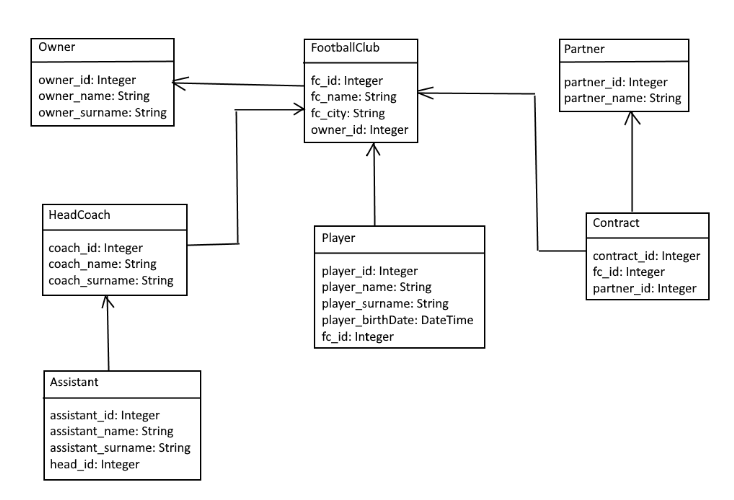
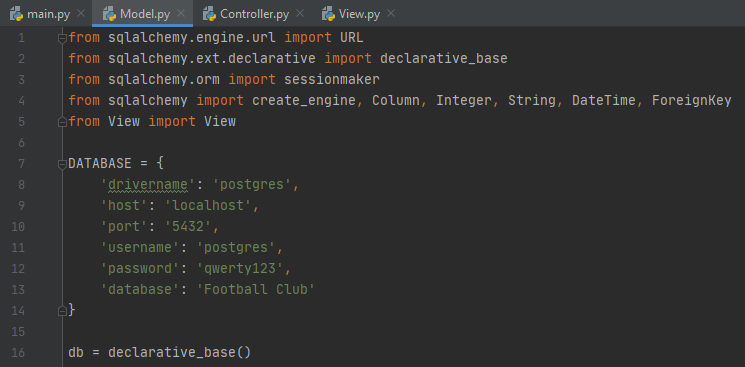
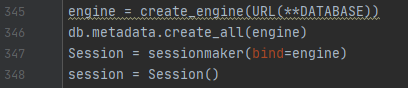


Рисунок 2 – Структура класів програми, що відповідають таблицям БД

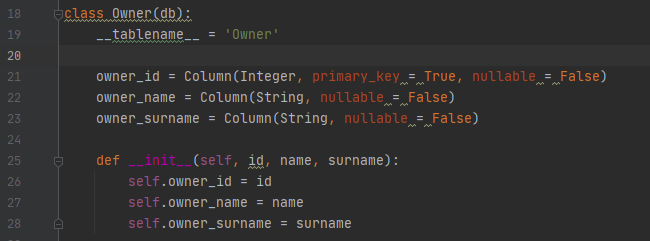
Підключення SQLAlchemy до БД:



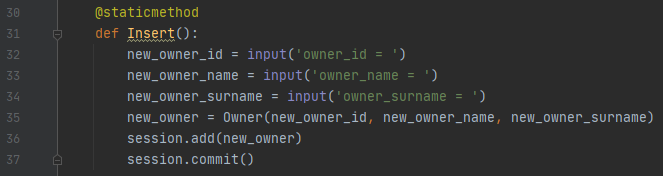


Для прикладу реалізації перетворення моделі з MVC у вигляд об’єктно-реляційної проекції (ORM) розглянемо код для таблиці «Owner».

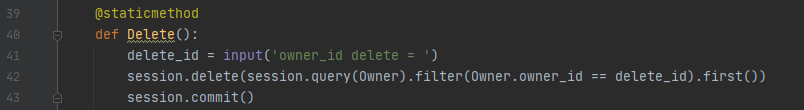
Клас Owner для співвідношення даних класу з таблицею «Owner» в БД:



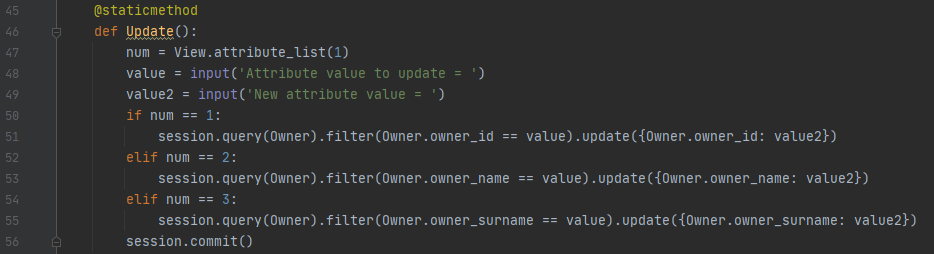
Метод додавання даних до таблиці «Owner»:



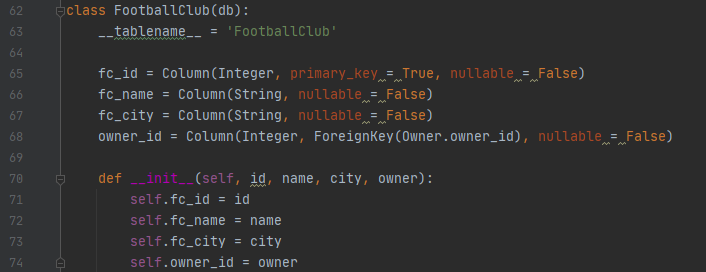
Метод видалення даних за полем owner\_id з таблиці «Owner»:



Метод редагування даних за будь-яким атрибутом таблиці «Owner»:



На прикладі таблиці «FootballClub» розглянемо, як визначається зовнішній ключ (поле owner\_id):



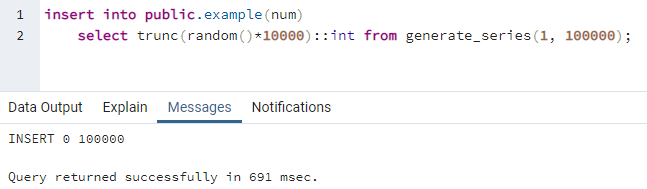
Інші класи та функції реалізовано аналогічно.

**Завдання 2**

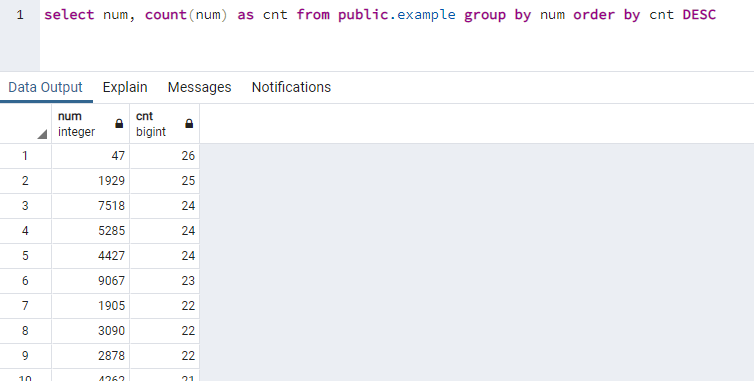
**Створення та аналіз індексу Hash**

HASH-індекси працюють тільки з простими умовами рівності. Хеш-індекс може бути використаний лише за умови, що стовпець, який індексується, бере участь у порівнянні з оператором «=». Для дослідження використаємо таблицю example з єдиною колонкою num типу integer.

До таблиці було додано 100 000 рандомізованих чисел:

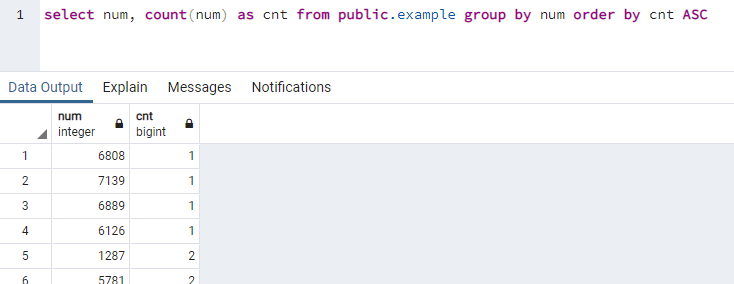


Дізнаємося дані, що зустрічаються найчастіше:



Оберемо num = 47.

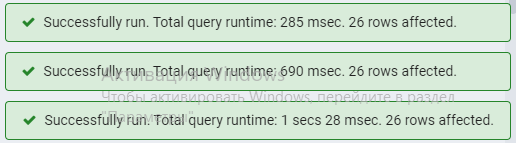
Дізнаємося дані, що зустрічаються найрідше:



Оберемо num = 6808.

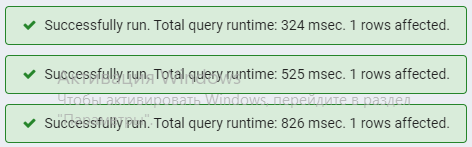
Виконаємо пошук записів за цими даними без використання індексу:





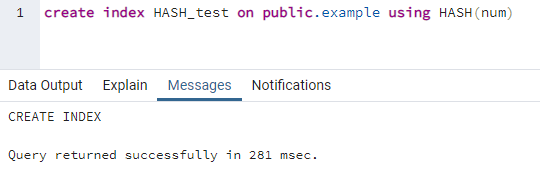
Середній час виконання запиту: 668 msec.



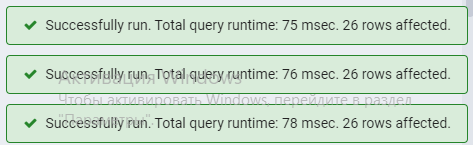


Середній час виконання запиту: 559 msec.

Тепер створимо індекс Hash на колонці num та виконаємо запити ще раз:

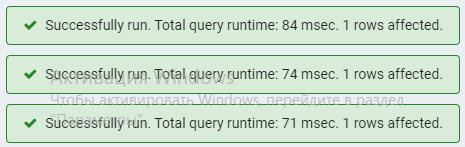






Середній час виконання запиту: 76 msec.





Середній час виконання запиту: 77 msec.

Як бачимо, час виконання запитів для даних, які найчастіше зустрічаються, зменшився приблизно в 8 разів. Для даних, які зустрічаються найрідше – приблизно в стільки ж.

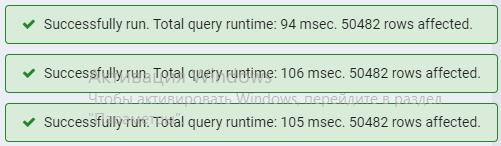
Отже, індексування HASH значно пришвидшило пошук і за значеннями, які часто зустрічається, і за значеннями, які зустрічаються найрідше. Використання цього індексу є доречним, якщо нам потрібно знайти дані за конкретним (=) значенням поля типу integer. Для інших знаків рівності (>, <, …) використання даного індексу буде не краще, ніж з використанням індексу BTree.

**Створення та аналіз індексу BRIN**

Для дослідження індексу BRIN використаємо таблицю example з єдиною колонкою num типу integer, використану для дослідження індексу Hash.

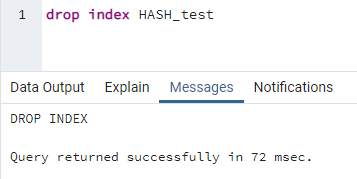
Для даного індексу є доречним пошук значної кількості даних. Виконаємо, наприклад, запит без використання індексу:



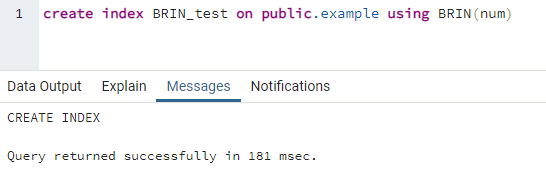


Середній час виконання запиту: 102 msec.

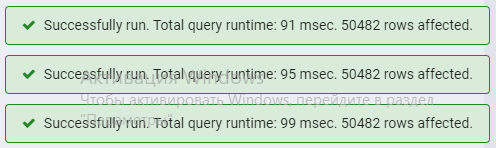
Спочатку видалимо індекс HASH, створений раніше:



Тепер створимо індекс BRIN на колонці num та повторимо запит:







Середній час виконання запиту: 95 msec.

Як бачимо, індекс несуттєво пришвидшив виконання запиту. Отже, в порівнянні з індексом BTree при пошуку значної кількості даних індекс BRIN проявив себе ненабагато краще в даному випадку та його використання є доречним.

**Завдання 3**

Опис тригера:

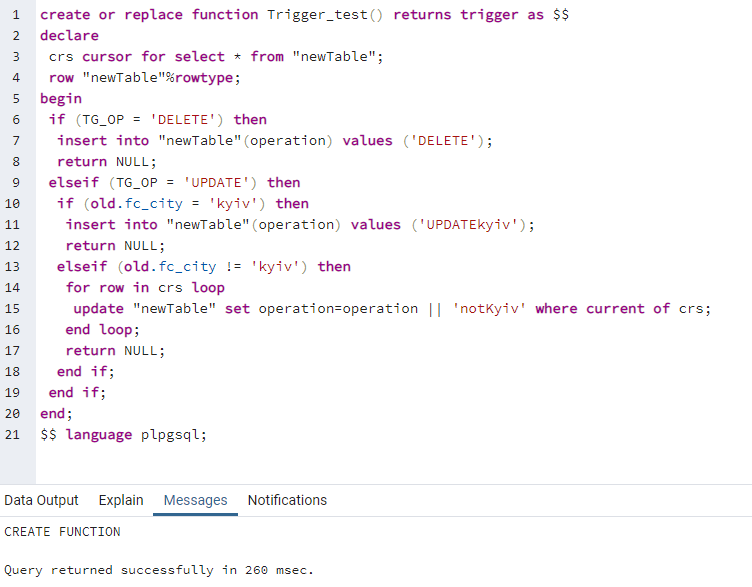
Тригер працює перед видаленням або оновленням таблиці «FootballClub».

Якщо була спроба видалити дані, то до нової таблиці «newTable», що має лише один атрибут operation типу character varying, заноситься назва операції, тобто «DELETE». При цьому дані з таблиці «FootballClub» видалені не будуть.

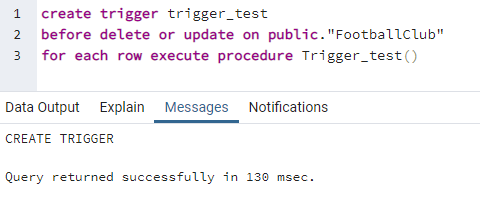
Якщо була спроба оновити дані, то тут маємо два варіанти:

1. Якщо поле fc\_city = ‘kyiv’, то до таблиці «newTable» буде додана назва операції з «kyiv» в кінці, тобто «UPDATEkyiv». Дані в таблиці «FootballClub» при цьому оновлено не буде;
2. Якщо поле fc\_city != ‘kyiv’, то до всіх даних поля operation таблиці «newTable» буде додано «notKyiv» в кінець. Дані в таблиці «FootballClub» при цьому оновлено не буде.

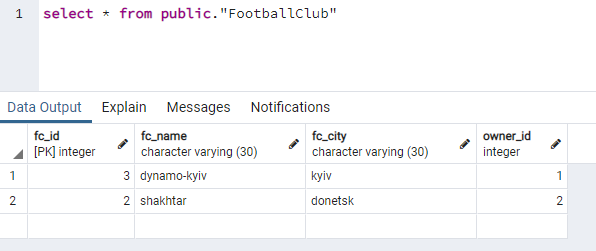
Створення тригеру:



Підключення тригеру до таблиці «FootballClub»:

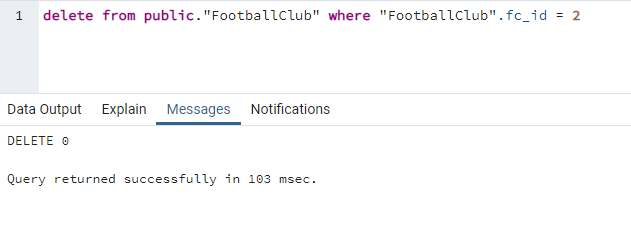


Вміст таблиці «FootballClub»:

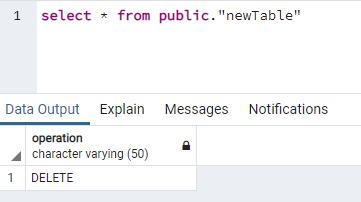


**Нотація:** Тригер не дозволяє видаляти та оновлювати дані. Тобто, в процесі дослідження роботи тригера вміст таблиці «FootballClub» змінюватися не буде.

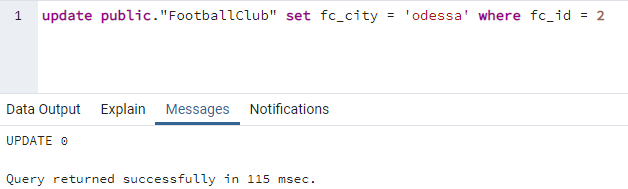
Спроба видалення даних з таблиці «FootballClub»:



Вміст таблиці «newTable»:

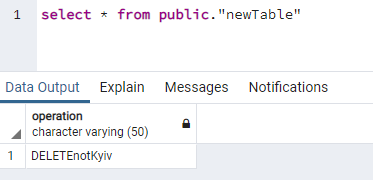


Спроба оновити дані таблиці «FootballClub», де поле fc\_city != ‘kyiv’:



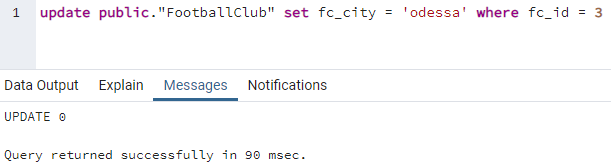
**Нотація:** За fc\_id = 2 поле fc\_city = ‘donetsk’, тобто != ‘kyiv’.

Вміст таблиці «newTable»:

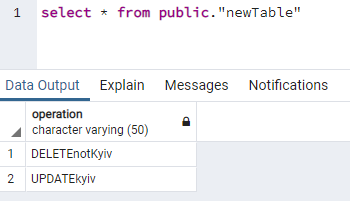


**Нотація:** Бачимо, що до всіх даних таблиці «newTable» в кінець колонки operation додалося «notKyiv».

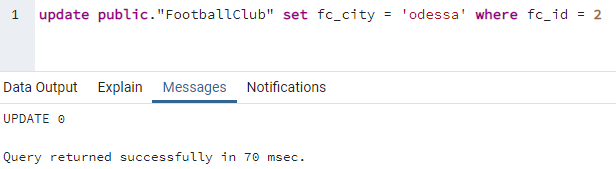
Спроба оновити дані таблиці «FootballClub», де поле fc\_city = ‘kyiv’ (в нашому випадку ми оновлюємо дані за fc\_id. Поле fc\_city = ‘kyiv’ при fc\_id = 3):



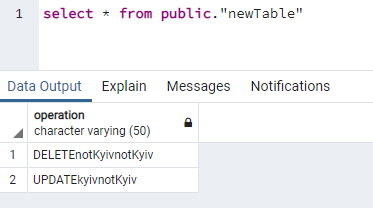
Вміст таблиці «newTable»:



Спробуємо ще раз оновити дані таблиці «FootballClub», де поле fc\_city != ‘kyiv’:



Вміст таблиці «newTable»:



Вміст таблиці «newTable» є коректним після проведення дослідження.

Отже, тригер працює відповідно до заданої логіки.