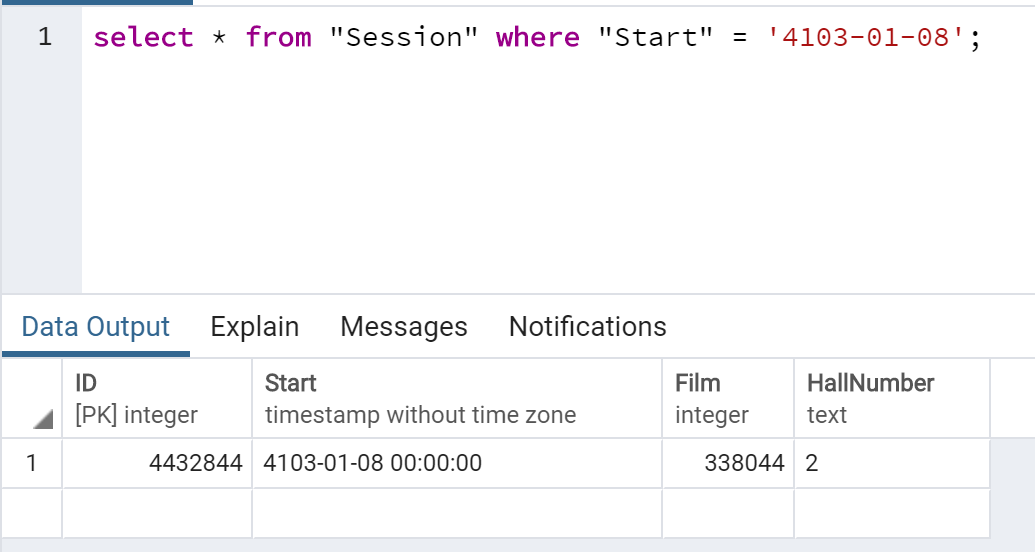
**Робота з індексами**

**BRIN** – це Block Range Index,головна концепція якого не знаходження необхідного значення, а уникнення перегляду свідомо непотрібних.

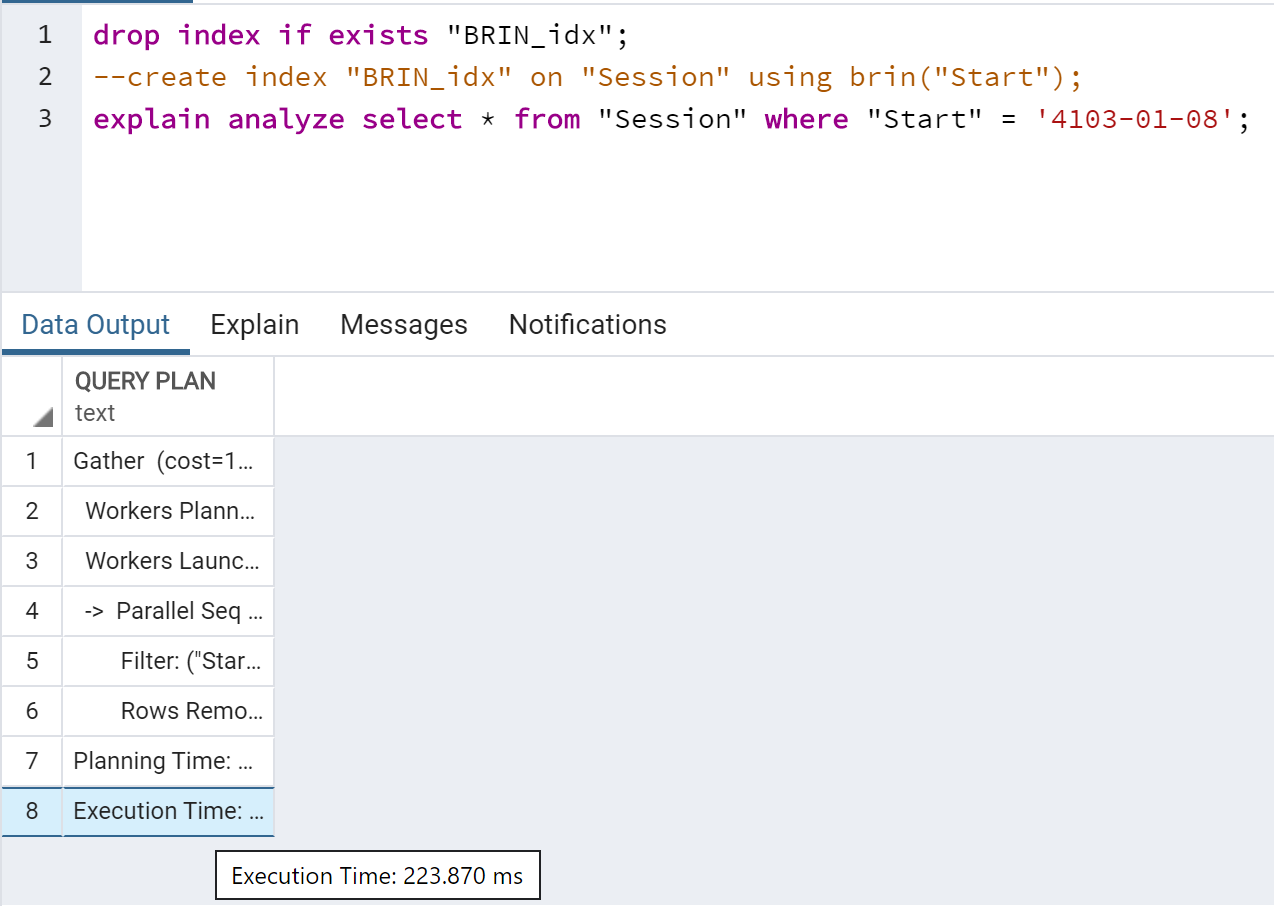
Він працює добре для тих стовпчиків, де значення корелюють із їх фізичним положенням в таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання.

Перевіряємо роботу індексу у таблиці Session на стовпчику Start з датою і часом початку Сеансу, значення якого при рандомізації впорядковувалися заздалегідь для кращої роботи індексу.

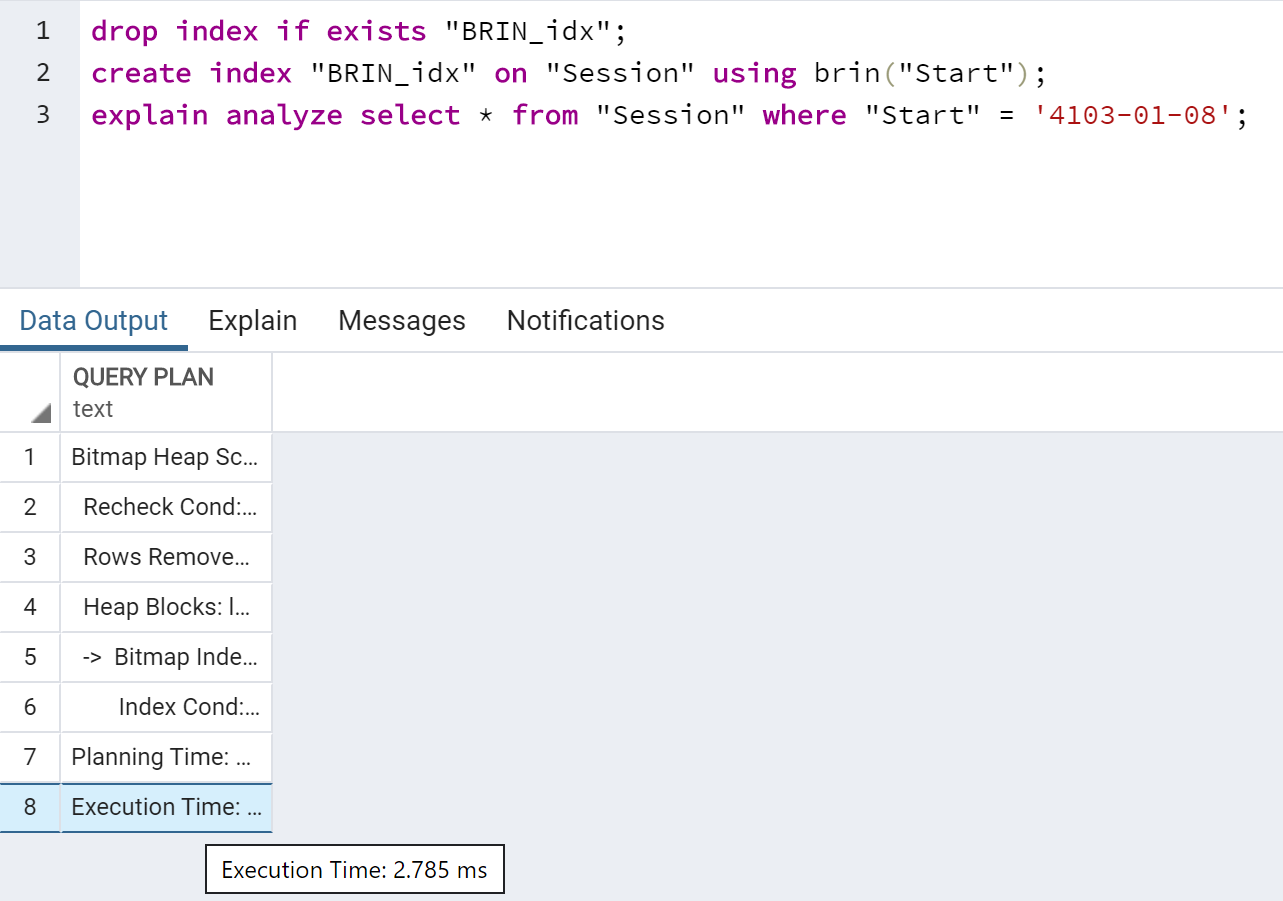
Знаходження рядків з певною датою і часом початку сеансу(якщо час не вказується,то за замовчання 00:00:00) без індексу



Аналіз виконання запиту без індексу



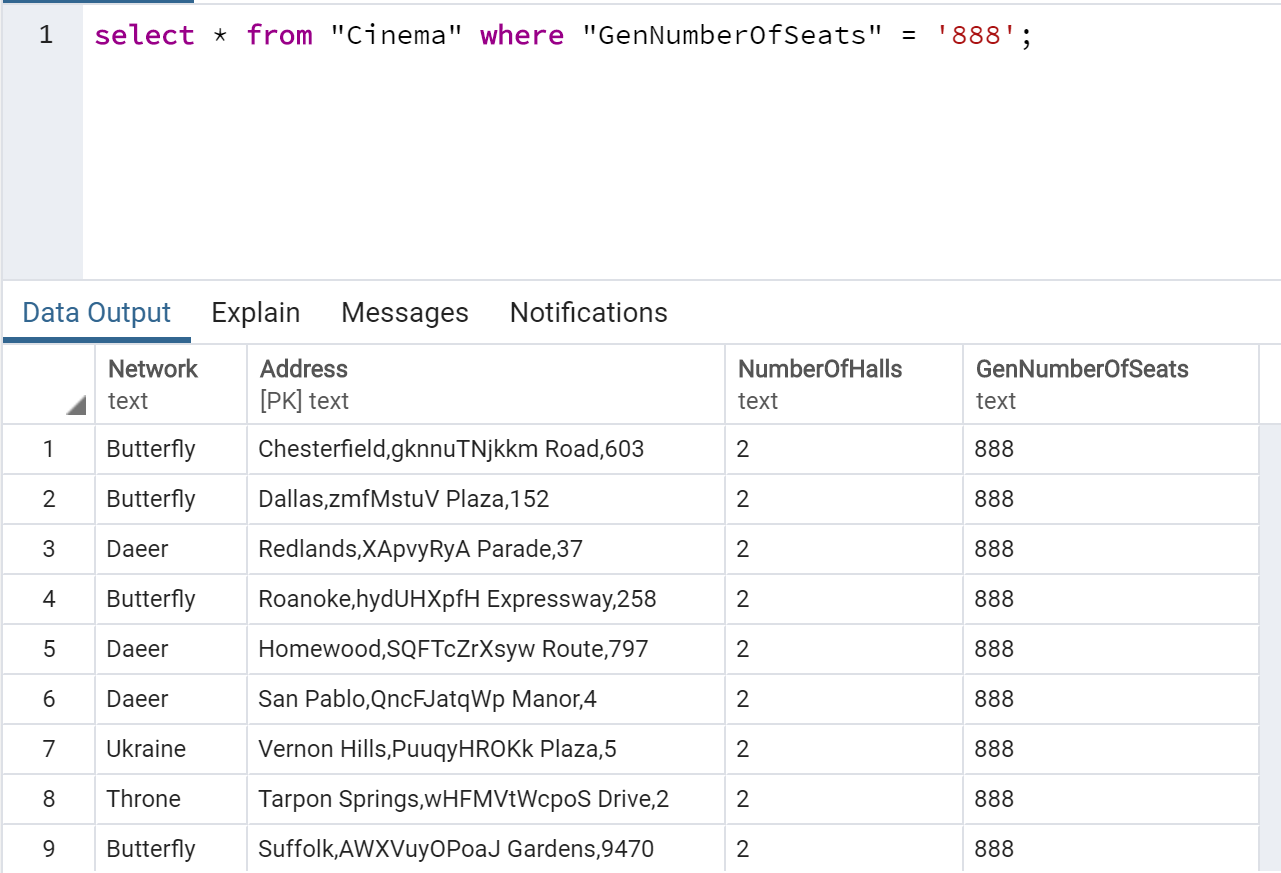
Аналіз виконання запиту з індексом BRIN



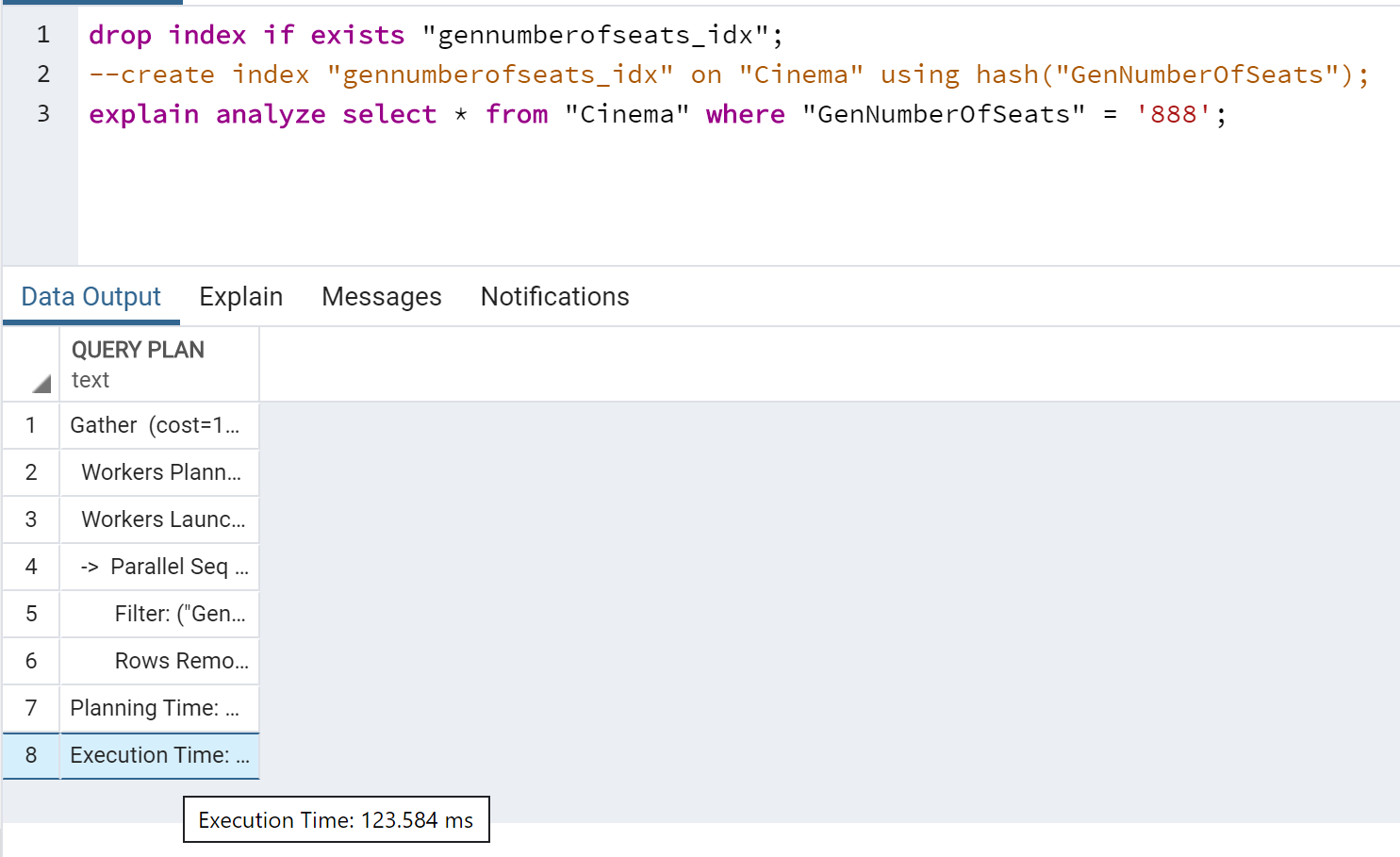
**Hash-індекси** були запропоновані Артуром Фуллером, і припускають зберігання не самих значень, а їх хеш, завдяки чому зменшується розмір (а, відповідно, і збільшується швидкість їх обробки) індексів з великих полів. Таким чином, при запитах з використанням HASH-індексів, порівнюватися будуть не шукане зі значення поля, а хеш від шуканого значення з хешамі полів.

Через нелінейнойсті хеш-функцій даний індекс не можна сортувати за значенням, що призводить до неможливості використання в порівняннях більше / менше і «is null». Крім того, так як хеші не унікальні, то для співпадаючих хеш застосовуються методи вирішення колізій.

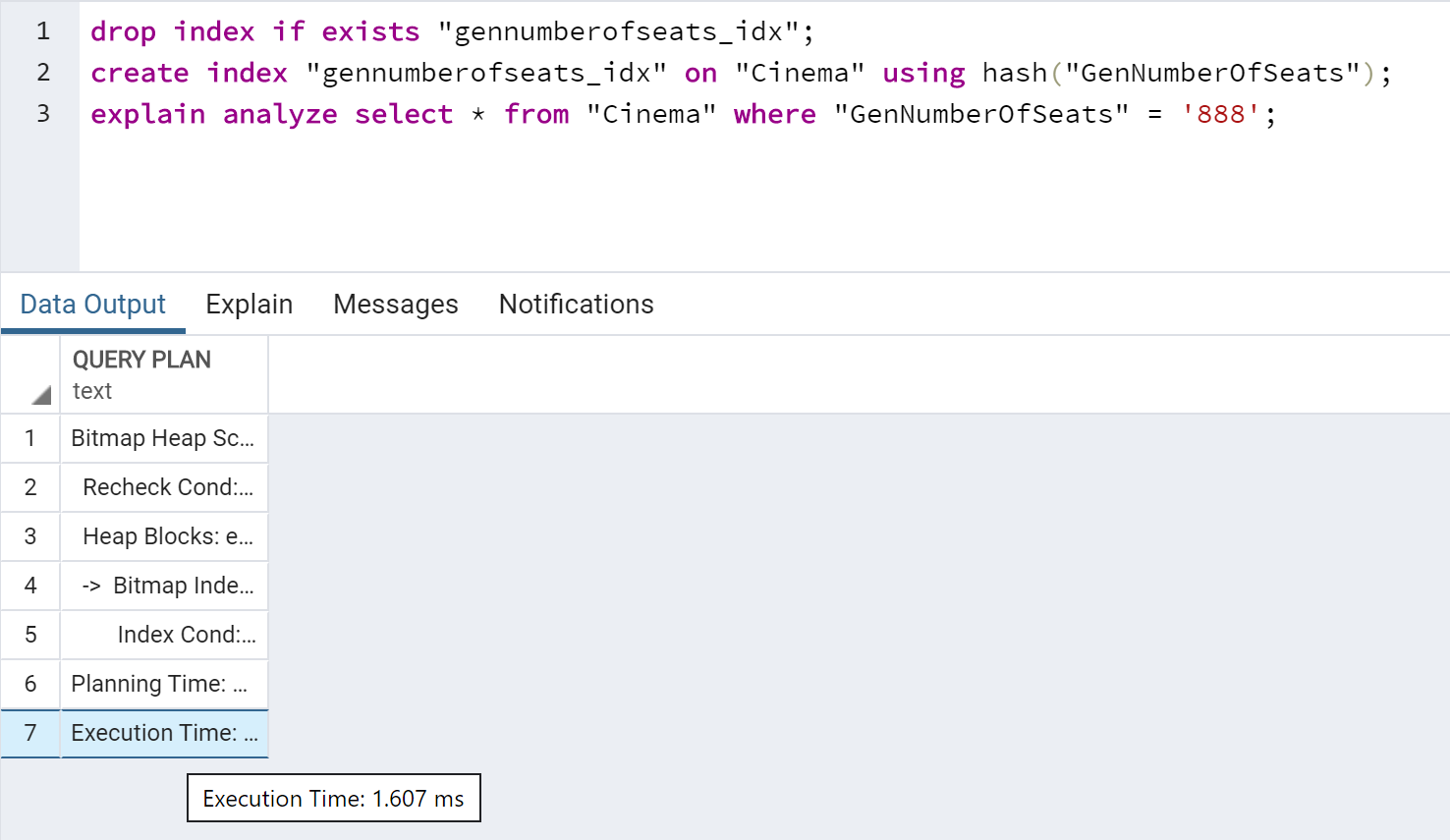
Знаходження рядків з певною загальною кількістю місць



Аналіз виконнання запиту без індексу



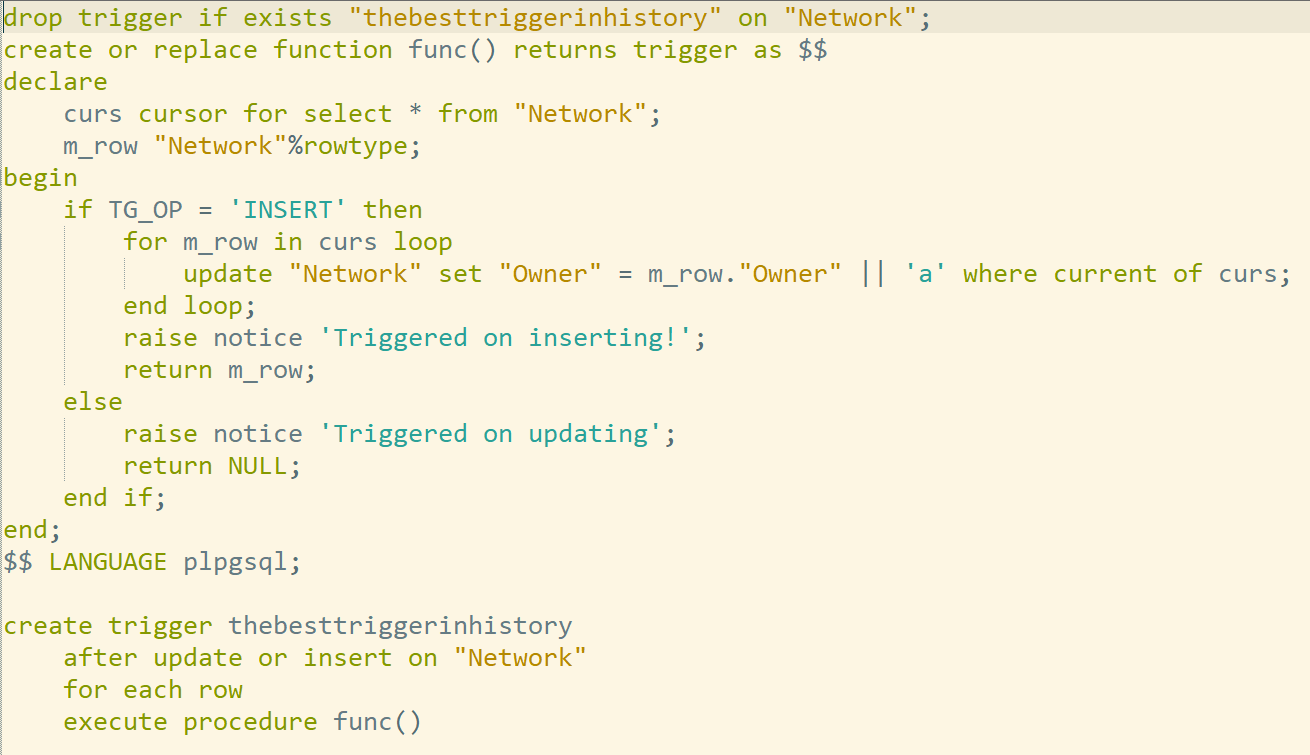
Аналіз виконання запиту з індексом Hash



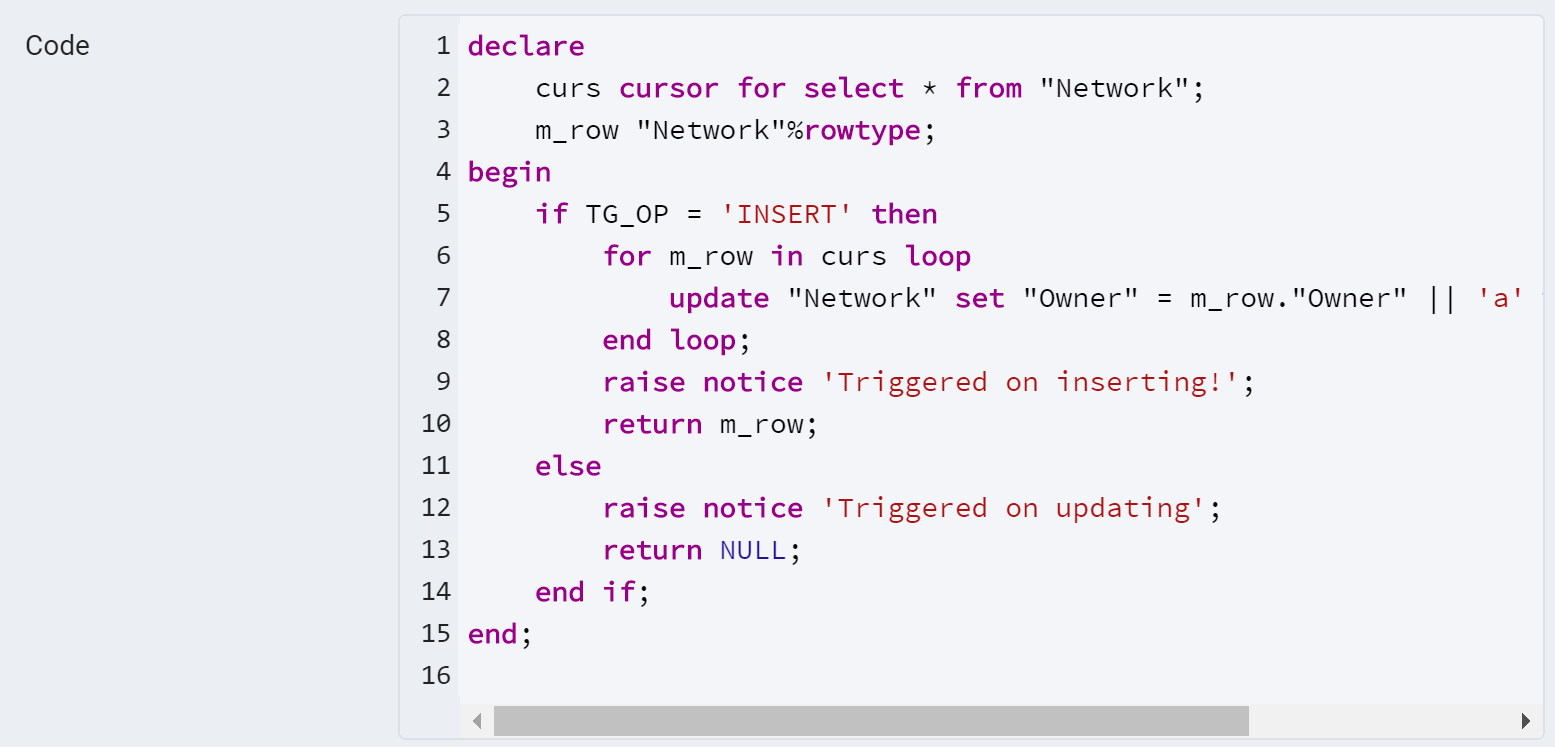
**Робота з тригерами**

Створений тригер thebesttriggerinthehistory стає активним після виконання вставки нового рядка,а також після оновлення вже існуючого рядка у таблиці Network.Якщо була виконана вставка нового рядка,то тригер проходиться по всім рядкам таблиці і додає до кінця поля Owner літеру ’а’, і після того, як він зробив дану дію над всіма рядками,то видає повідомлення- “Triggered on inserting” та повертає зміну з останнім рядком.Якщо ж було виконане оновлення рядка таблиці,то тригер видає повідомлення “Triggered on updating” та повертає NULL.

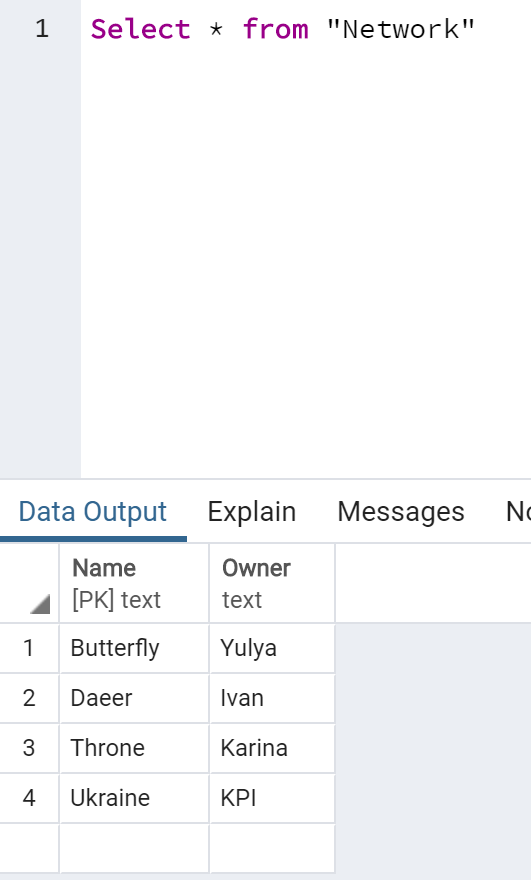
Тригерна функція та створення тригера



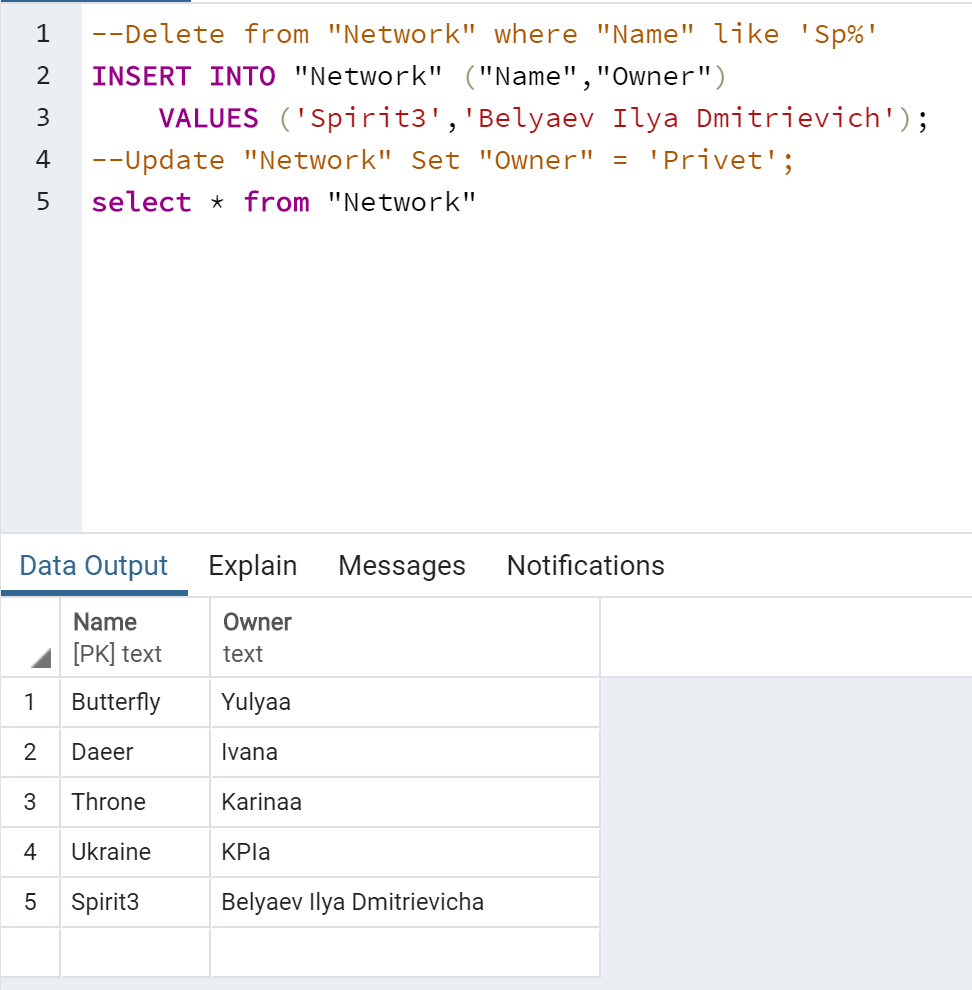
Безпосередньо код створеного тригера

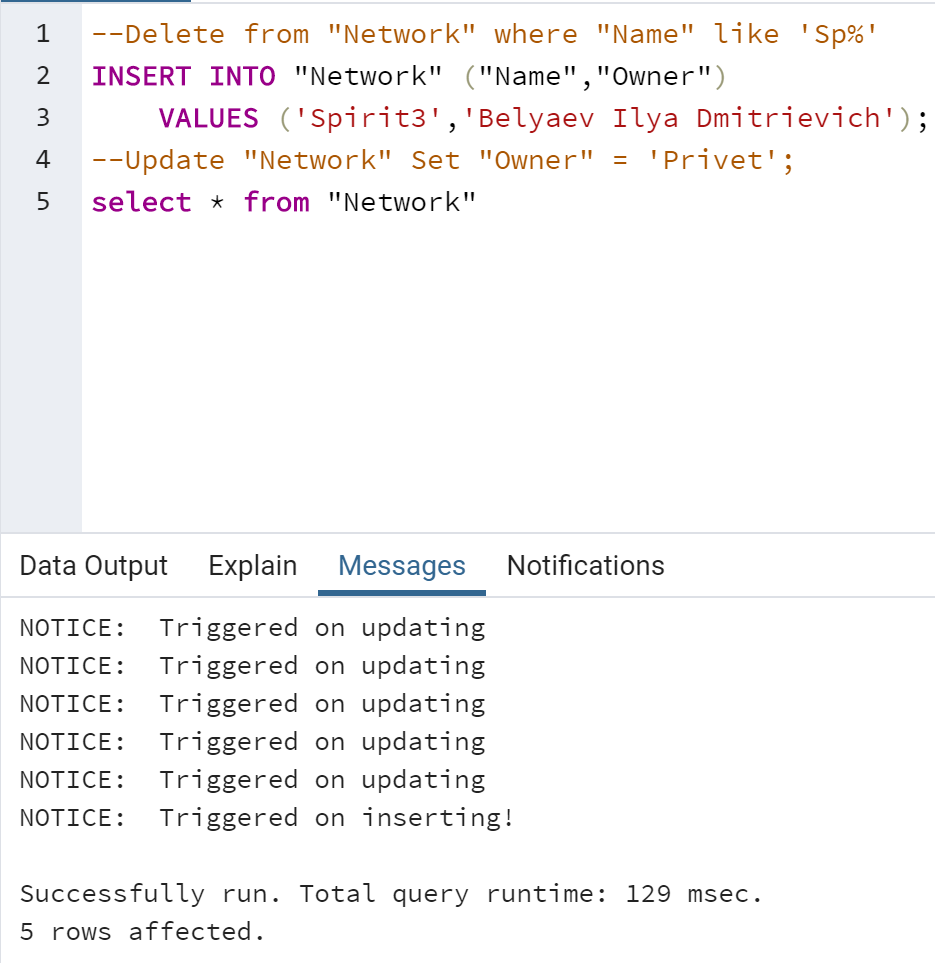


Початковий стан рядків таблиці Network

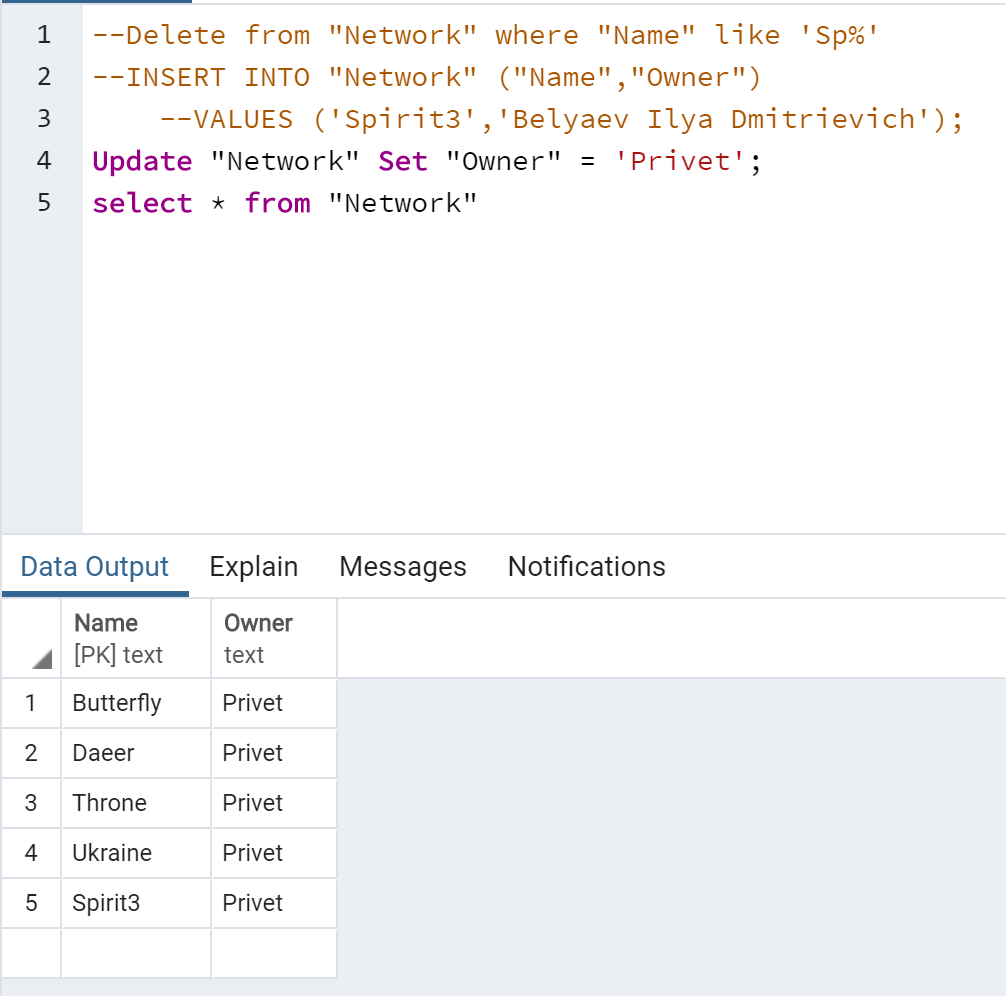


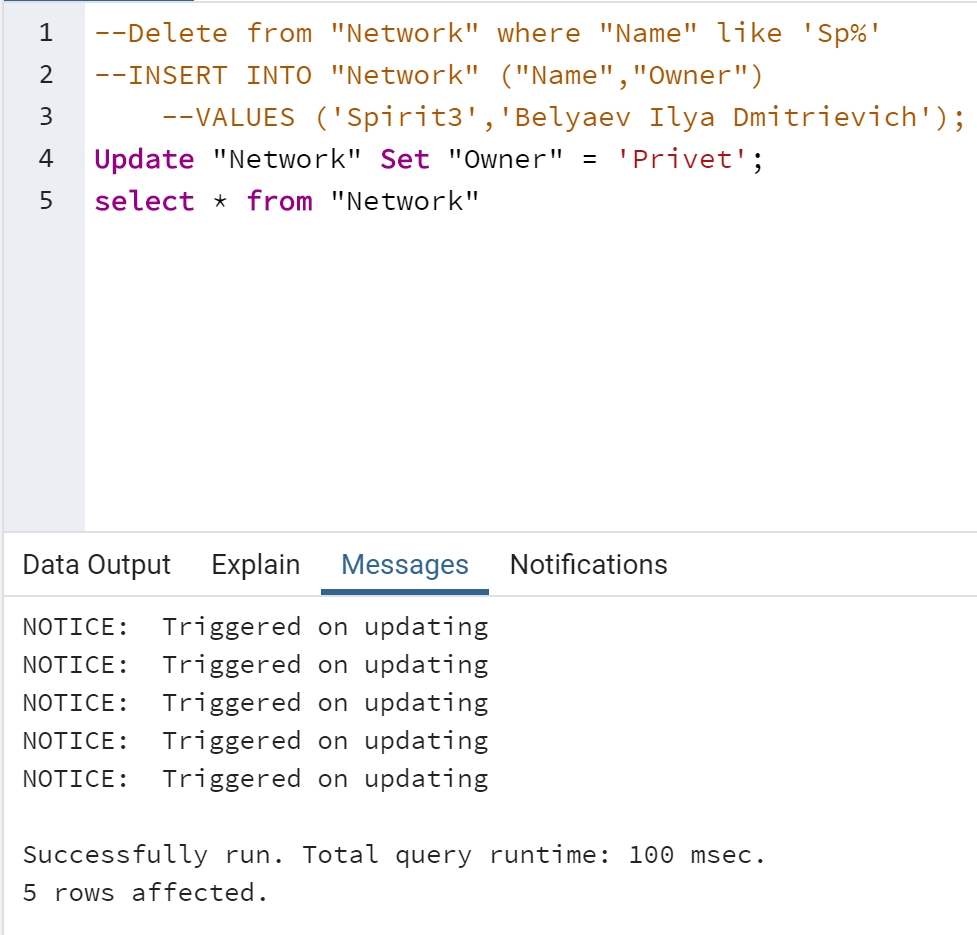
Перевірка роботи тригера після вставки нового рядка



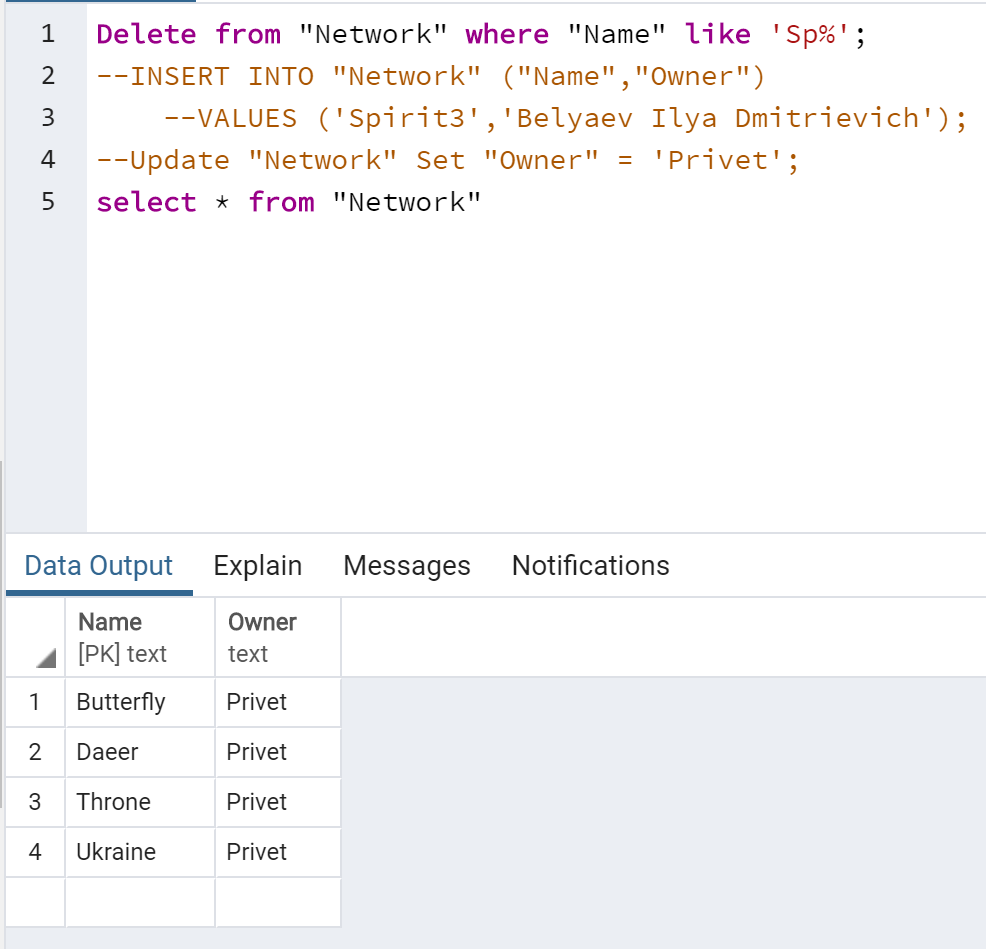


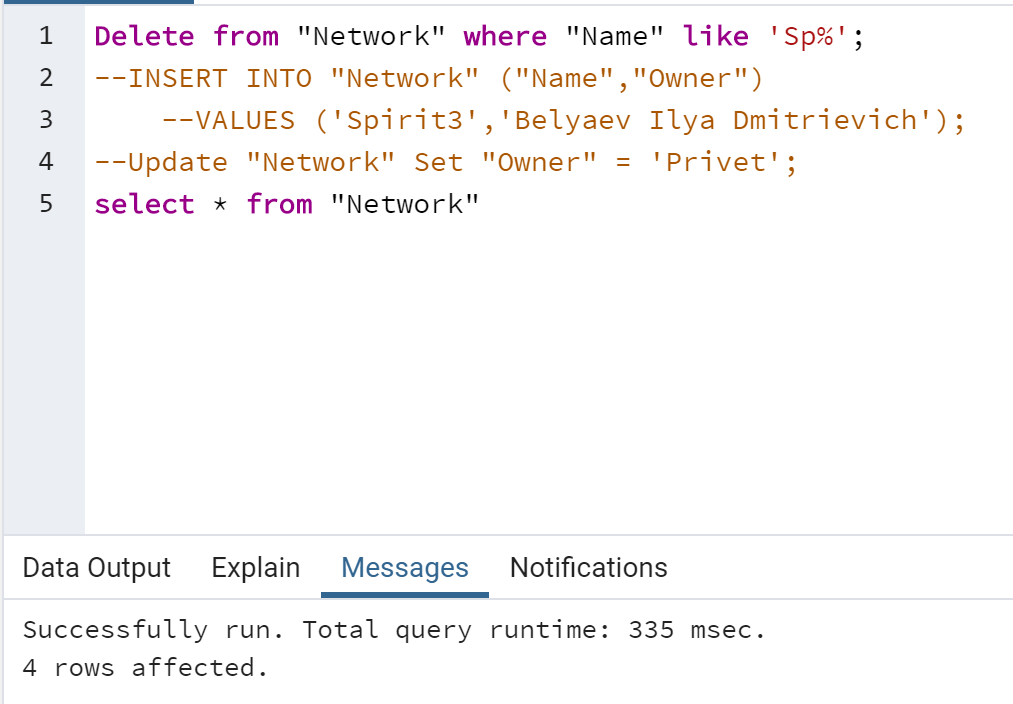
Перевірка роботи тригера після оновлення значень всіх рядків таблиці





Перевірка роботи тригера на дію відмінну від оновлення та вставки (в даному випадку візьмемо видалення).



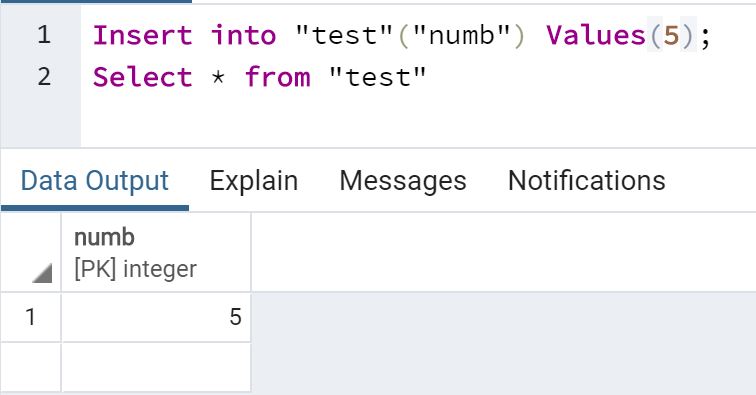


**Робота з транзакціями**

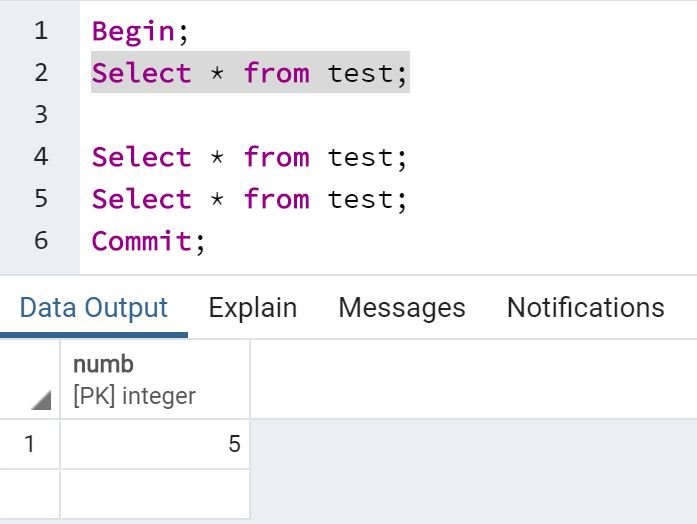
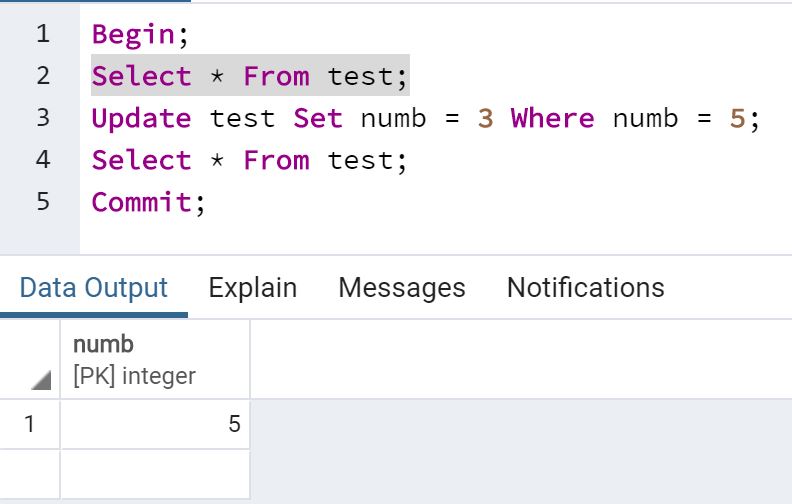
Будемо досліджувати рівні ізоляції транзакції на таблиці test у якого буде всього одне числове поле numb. Цього буде достатньо для експериментів.

Почнемо з Readcommited, який в PostgreSQL встановлений за замовченням.

Спочатку вставимо у таблицю перший рядок.

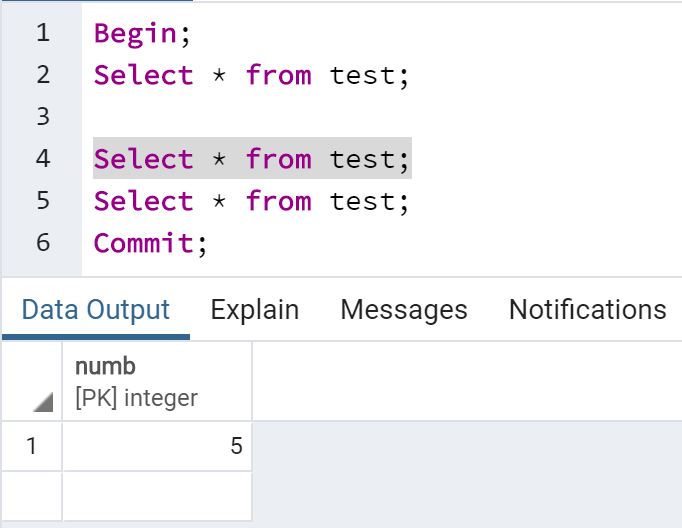


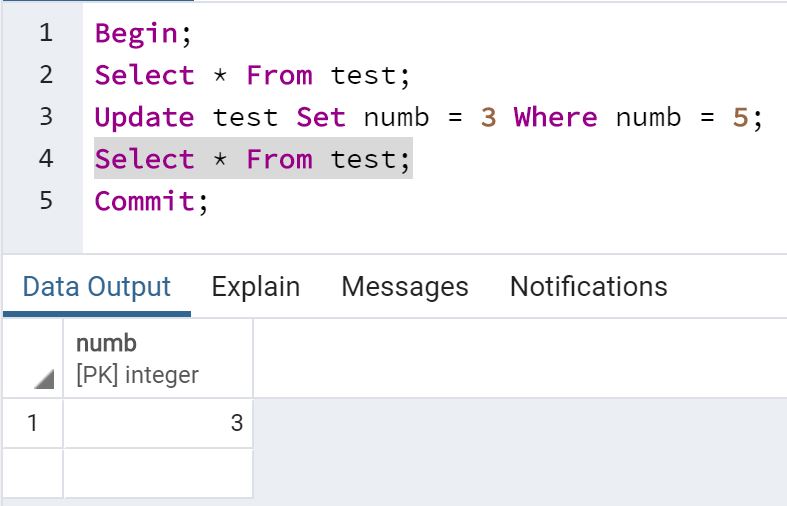
Тепер запустимо 2 транзакцій у двох вікнах та будемо послідовно виконувати команди



Бачимо, що дані в обох транзакціях поки однакові.

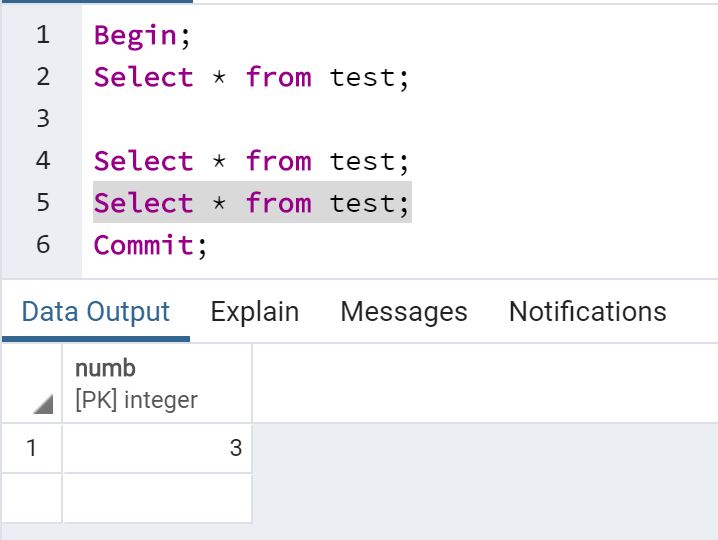
Тепер виконаємо update в першій транзакції та потім select в обох транзакціях.

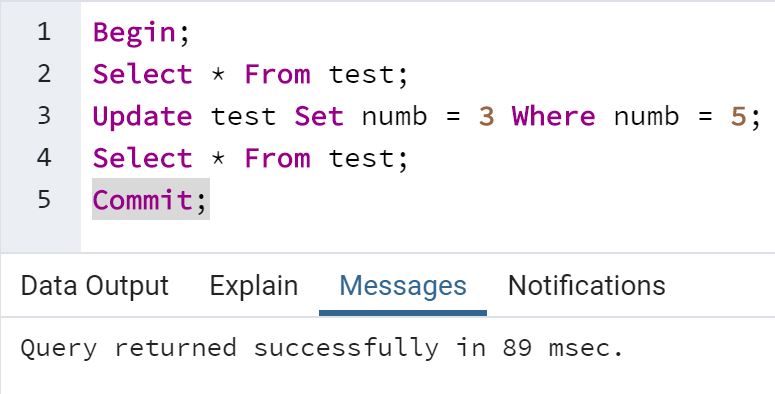
****



Бачимо, що в першій транзакції дані змінилися , а в другій залишилися ті самі.

Тепер виконаємо в першій транзакції commit та select у другій.

****

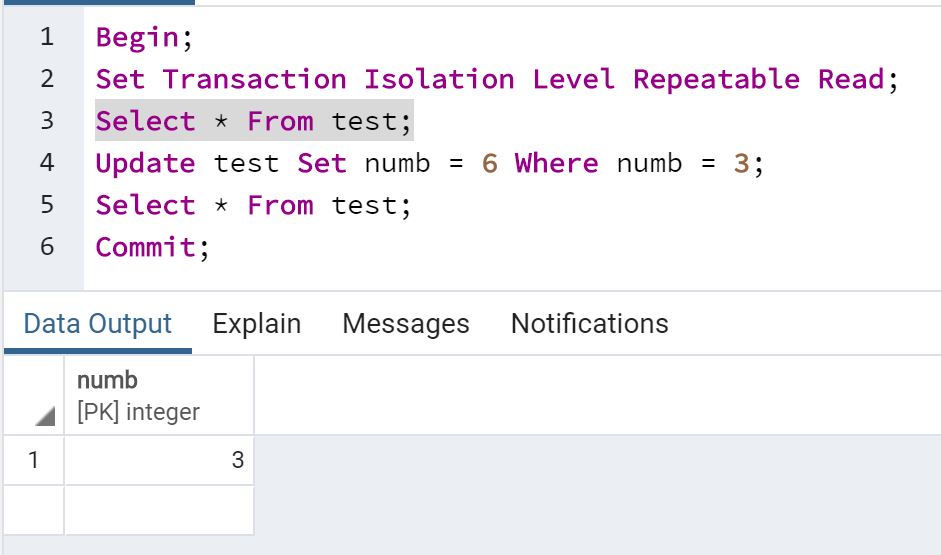


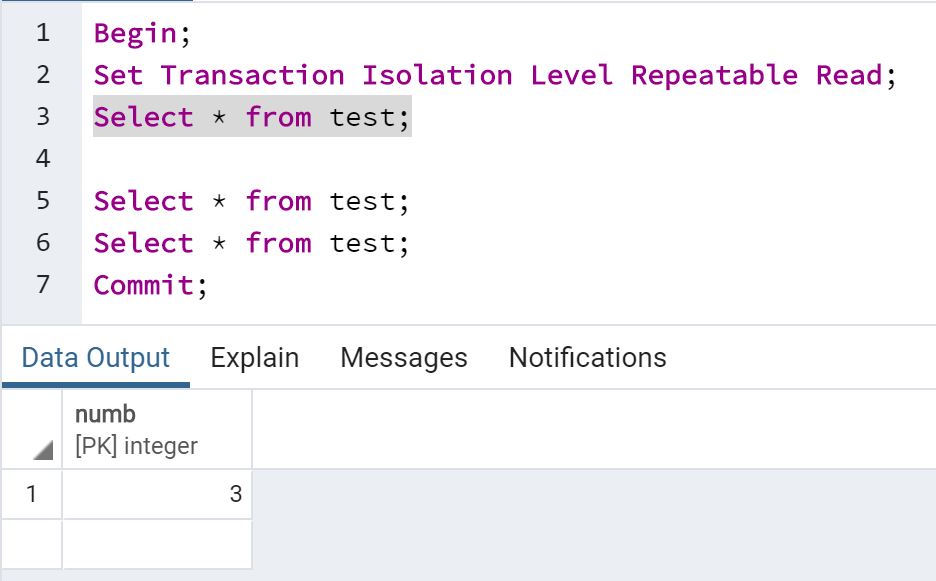
Ми маємо феномен під назвою “Unrepeatable read” в другій транзакції. Друга транзакція ще не закінчена, але ми бачимо вже закомічені дані з першої транзакції. Тому й має цей рівень ізоляції транзакцій назву “Read committed”, що означає, що в транзакціях будемо бачити при читанні дані, які в момент читання були закомічені в базу.

Для усунення цього ефекту треба перейти до рівня ізоляції транзакції “Repeatable read”.

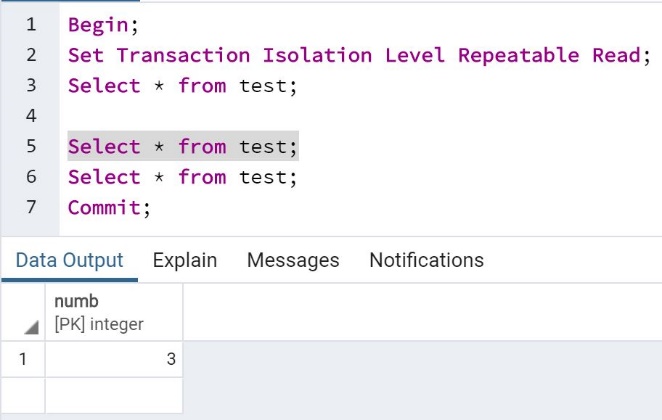
Запустимо знову 2 транзакції, але вже встановимо їм рівень ізоляції транзакції “Repeatable read”.

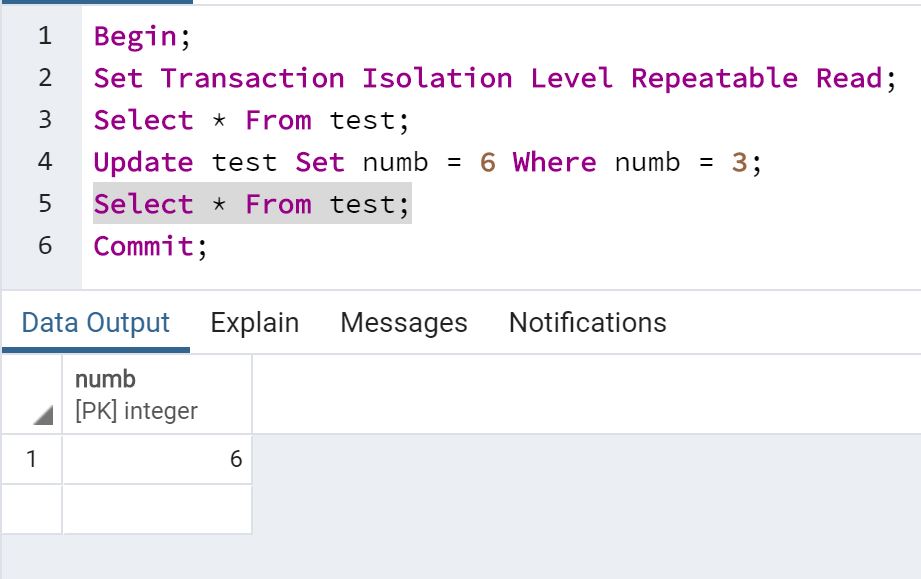
Бачимо, що на початку транзакцій дані в них відображуються однакові



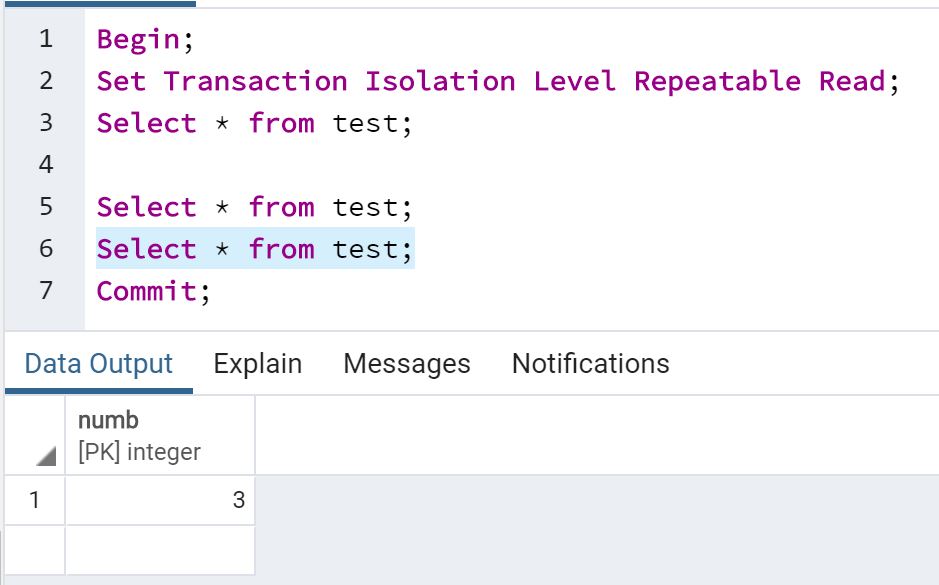


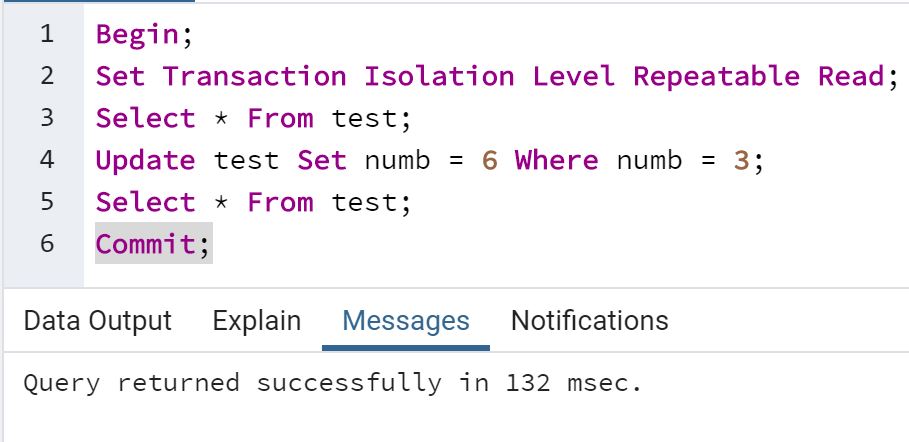
Виконаємо в першій транзакції update та потім у обох транзакціях select.

****



У першій транзакції бачимо оновлені дані , а в другій ще старі. Тепер зробимо commit першої транзакції та select в другій.

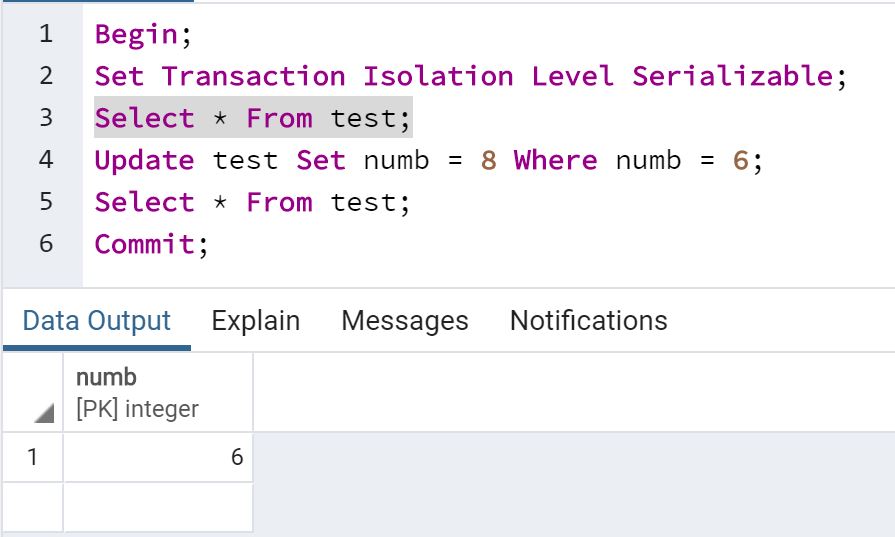
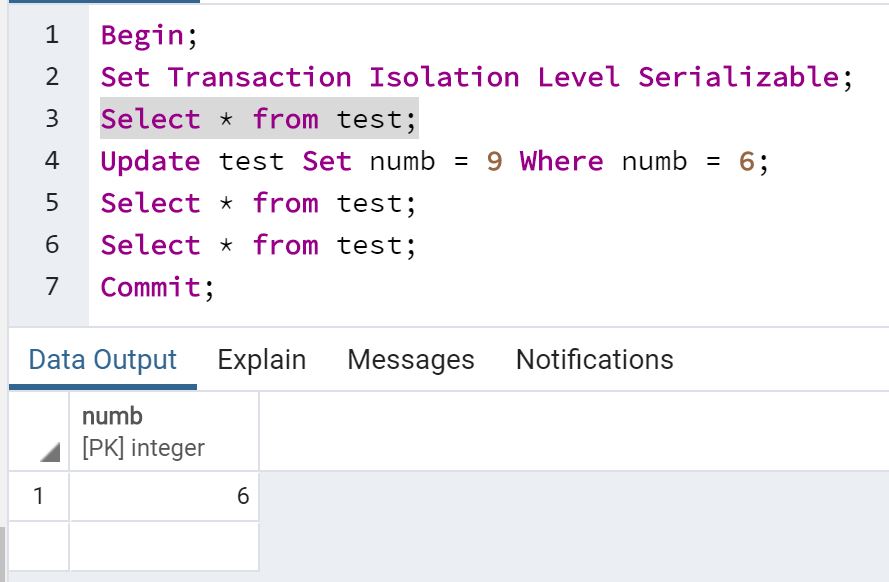
****



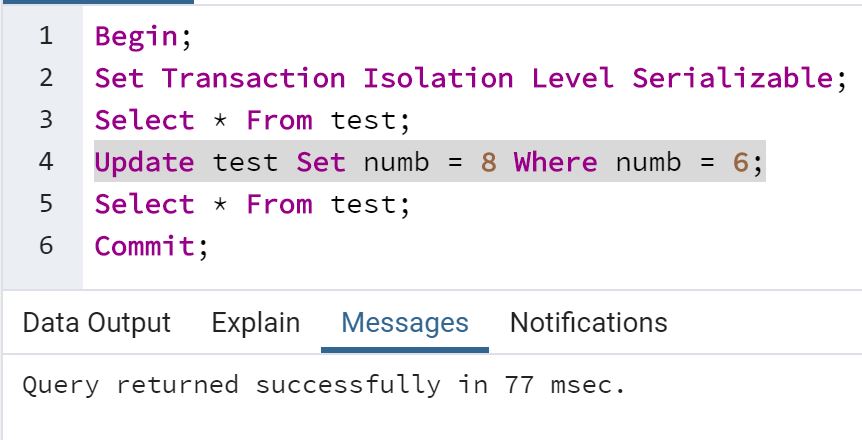
Вдалося позбутися феномену “Unrepeatable read”. Суть рівня ізоляції транзакцій “Repeatable read”, що в транзакції ми можемо бачити тільки зміни зроблені в даній транзакції.

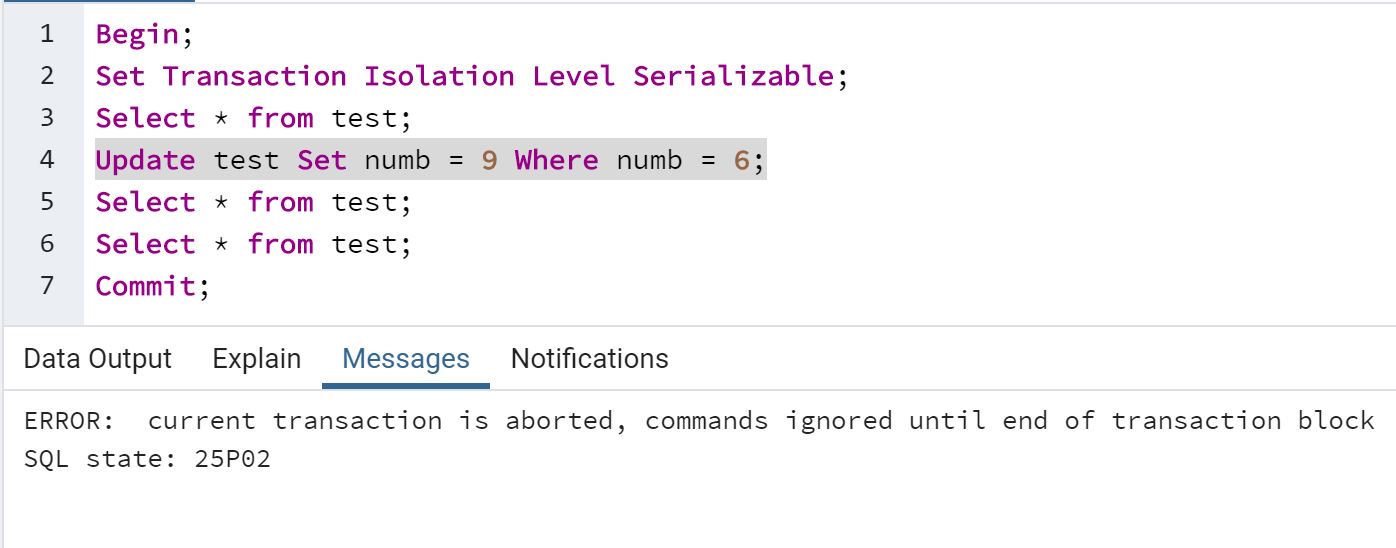
**Тепер дослідимо рівень транзакції “Serializable”**

Розпочнем 2 транзакції з рівнем ізоляції транзакції “Serializable” та виведемо через select дані таблиці test.



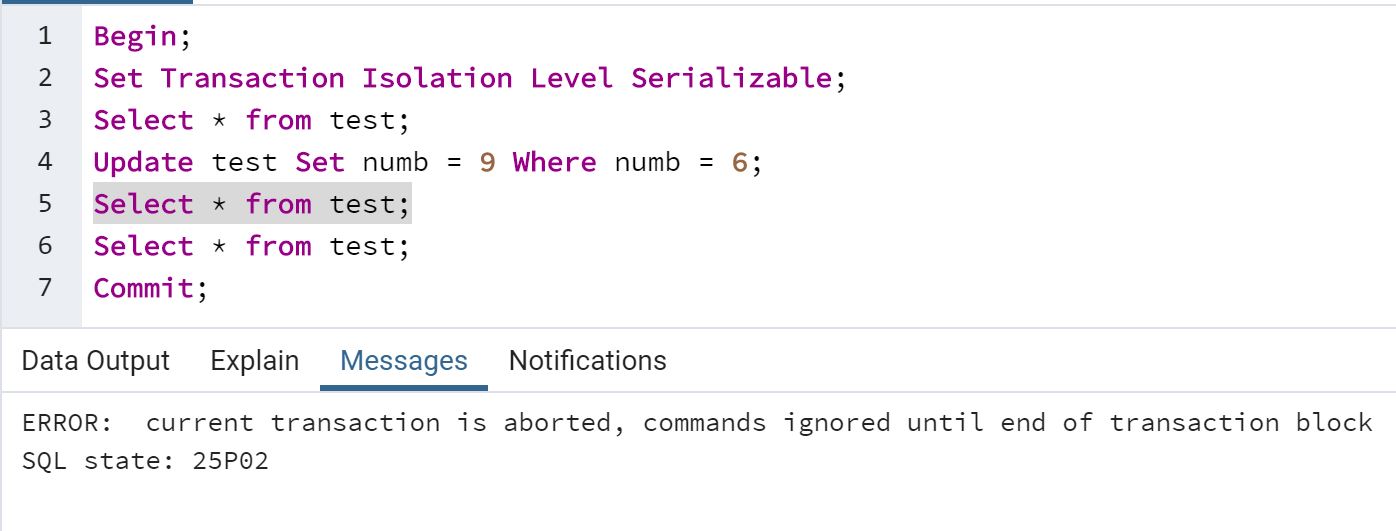
А тепер спробуємо зробити в обох транзакція update. Спочатку в першій, а потім в другій та побачимо, що буде.



****

У першій транзакції все пройшло добре, але не в другій. В другому вікні при спробі встановлення нового значення запит завис,і після того,як він був примусово достроково завершений,з другої спробою одразу з’явилось повідомлення про помилку.

Після спроби виконати інші команду із транзакції також отримуємо дане повідомлення.



По-факту, тепер можна зробити тільки commit, який буде виконаний як rollback. Це сталося по причині того, що в другій транзакції була спроба модифікувати рядок, який вже був змінений у першій транзакції, що й призвело до конфлікту, бо рівень ізоляції транзакції “Serializable” не дозволяє цього. Він тому й має назву “Serializable”, бо в паралельних транзакціях з цим рівнем ізоляції можуть виконуватися тільки ті команди з якими можна по-факту виконувати дані транзакції послідовно та в будь-якому порядку.