

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих комп’ютерних систем**

**Лабораторна робота №3**

з дисципліни

**“Бази даних та засоби управління ”**

*Тема****: «Засоби оптимізації роботи СУБД PostgreSQL»***

Виконав: студент ІII курсу

ФПМ групи КВ-93

Нікішин Є.О.

Перевірив: Павловський В.І.

**Завдання до лабораторної роботи**

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Загальне 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.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Завдання за варіантом**



**Логічна модель учбової предметної області “Аптека”**

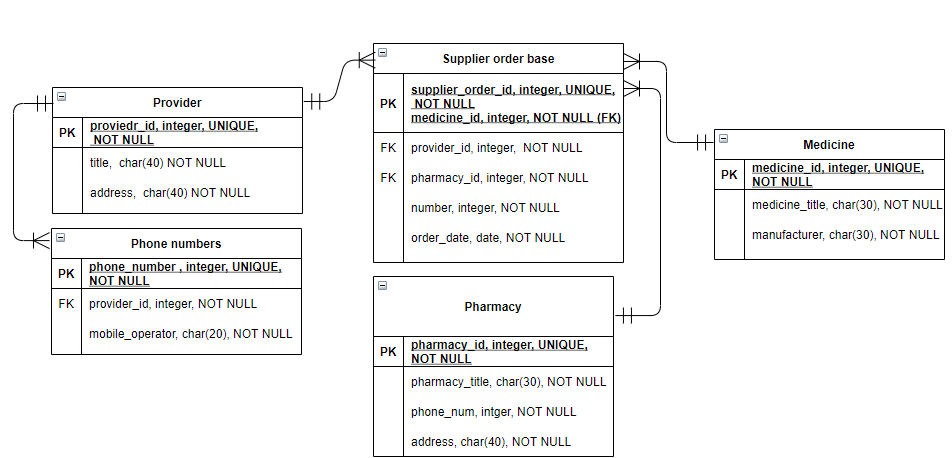


Рисунок 0 - Логічна модель учбової предметної області “Аптека”

**Середовище розробки**

Середовище розробки бази даних - PostgreSQL.

Середовище розробки програми - PyCharm Community Edition.

Мова програмування - Python(консольний застосунок).

Для зв’язку програми Python з базою даних PostgreSQL необхідно було імпортувати бібліотеку sqlAlchemy.

Середовище для відлагодження SQL-запитів до бази даних - pgAdmin4.

**Фрагменти програми внесення, редагування та вилучення даних у базі даних засобами ORM**

****

Рисунок 1 - Клас ORM для таблиці “Постачальник” реалізований у модулі Model

****

Рисунок 2 - Клас ORM для таблиці “Телефонні номери” реалізований у модулі Model



Рисунок 3 - Клас ORM для таблиці “Аптека” реалізований у модулі Model

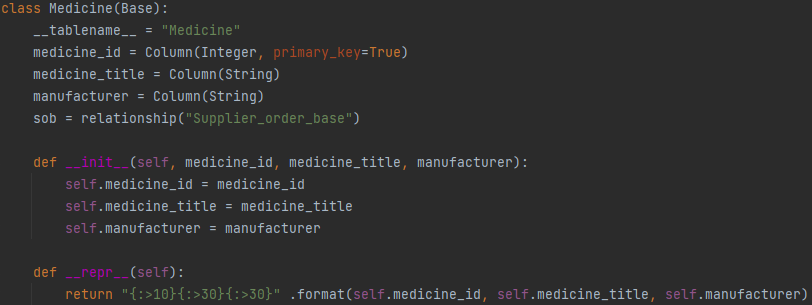
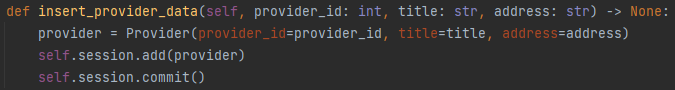
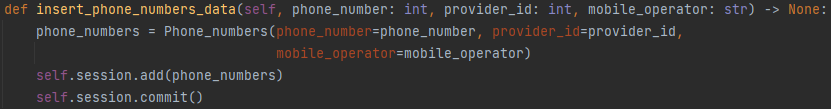
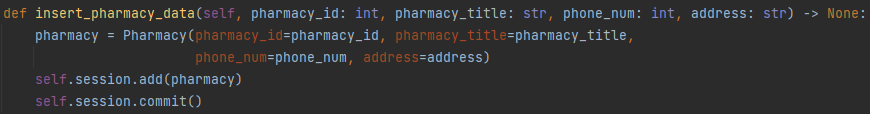


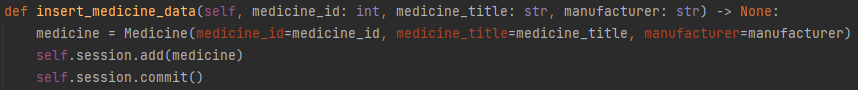
Рисунок 4 - Клас ORM для таблиці “Препарат” реалізований у модулі Model

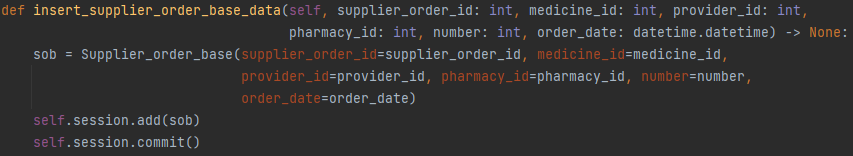
  
Рисунок 5 - Клас ORM для таблиці “Замовлення для постачальника” реалізований у модулі Model

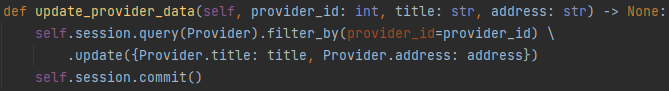
  
Рисунок 6 - Фрагмент програми для внесення даних у таблицю “Постачальник”

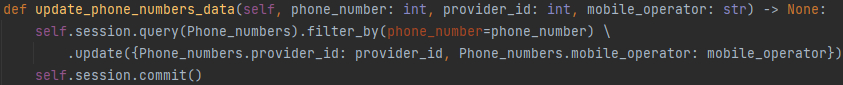
  
Рисунок 7 - Фрагмент програми для внесення даних у таблицю “Телефонні номери”

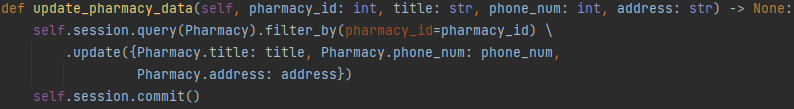
  
Рисунок 8 - Фрагмент програми для внесення даних у таблицю “Аптека”

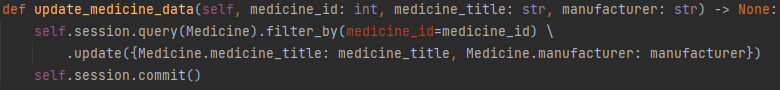
  
Рисунок 9 - Фрагмент програми для внесення даних у таблицю “Препарати”

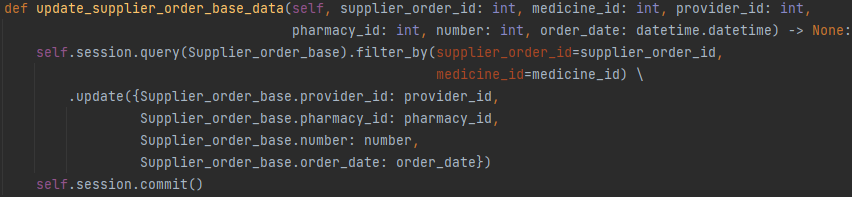
  
Рисунок 10 - Фрагмент програми для внесення даних у таблицю “Замовлення для постачальника”

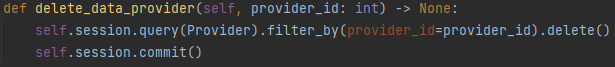
  
Рисунок 11 - Фрагмент програми для редагування даних у таблицю “Постачальник”

  
Рисунок 12 - Фрагмент програми для редагування даних у таблицю “Телефонні номери”

  
Рисунок 13 - Фрагмент програми для редагування даних у таблицю “Аптека”

  
Рисунок 14 - Фрагмент програми для редагування даних у таблицю “Препарати”

  
Рисунок 15 - Фрагмент програми для редагування даних у таблицю “Замовлення для постачальника”

  
Рисунок 16 - Фрагмент програми для вилучення даних у таблицю “Постачальник”

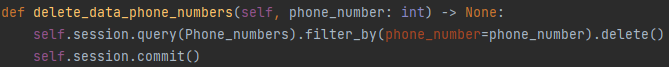
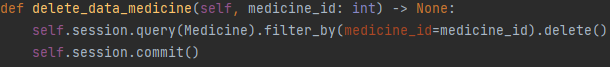
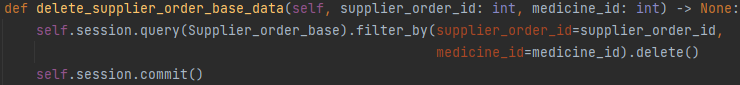
  
Рисунок 17 - Фрагмент програми для вилучення даних у таблицю “Телефонні номери”

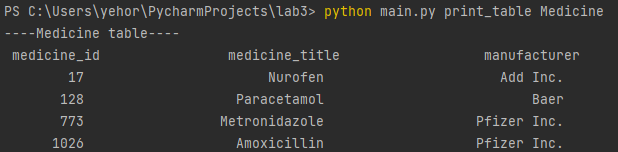
  
Рисунок 18 - Фрагмент програми для вилучення даних у таблицю “Аптека”

  
Рисунок 19 - Фрагмент програми для вилучення даних у таблицю “Препарати”

  
Рисунок 20 - Фрагмент програми для вилучення даних у таблицю “Замовлення для постачальника”

Наведені вище фрагменти програми відповідають за функціонал додавання, редагування та вилучення даних у базі даних. Але завдяки тому, що базові операції з базою даних були виконані за допомогою ORM, що суттєво зменшило об’єм коду у порівнянні з використанням SQL запитів.

**Результати виконання базових операцій з базою даних засобами ORM**

****Рисунок 21 - Таблиця “Препарати” до виконання директив внесення, вилучення та редагування

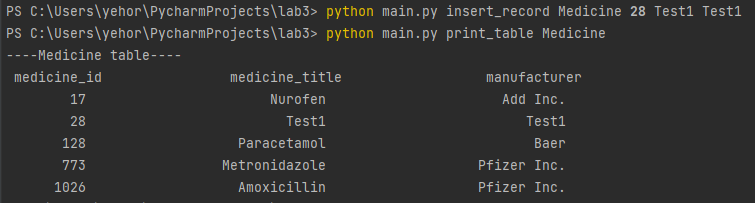
****

Рисунок 22 - Таблиця “Препарати” після виконання директиви внесення

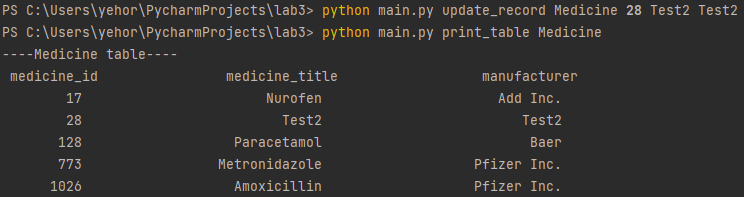
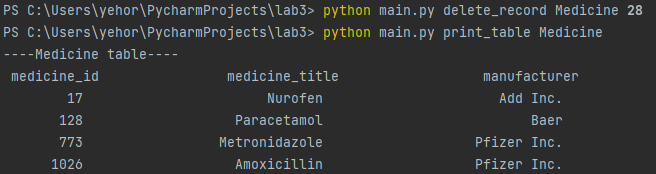


Рисунок 23 - Таблиця “Препарати” після виконання директиви редагування

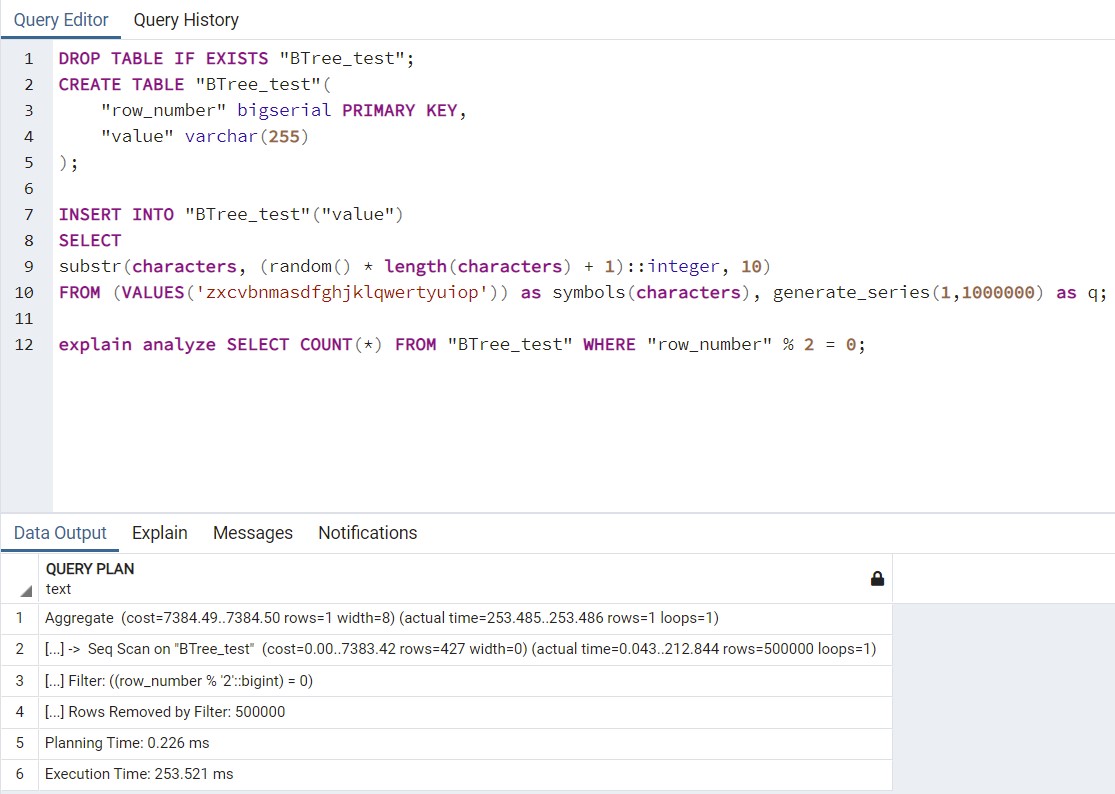
  
Рисунок 24 - Таблиця “Препарати” після виконання директиви вилучення

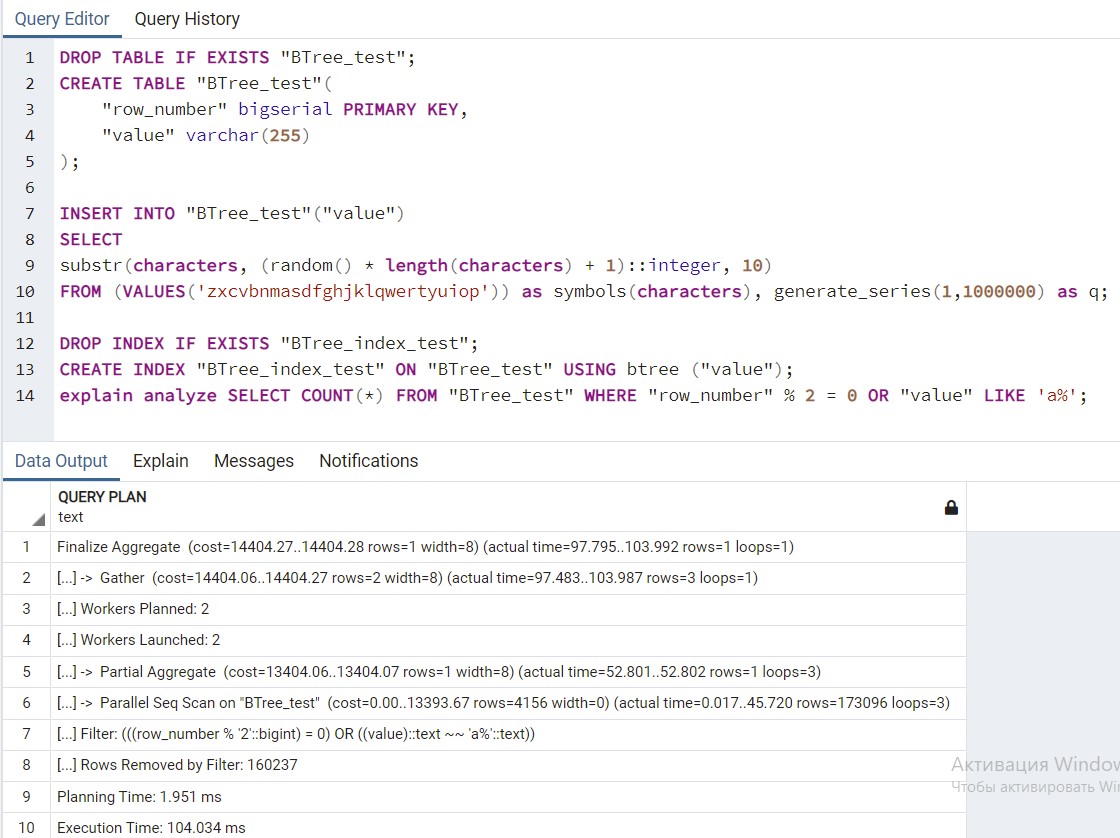
# **Завдання №2**

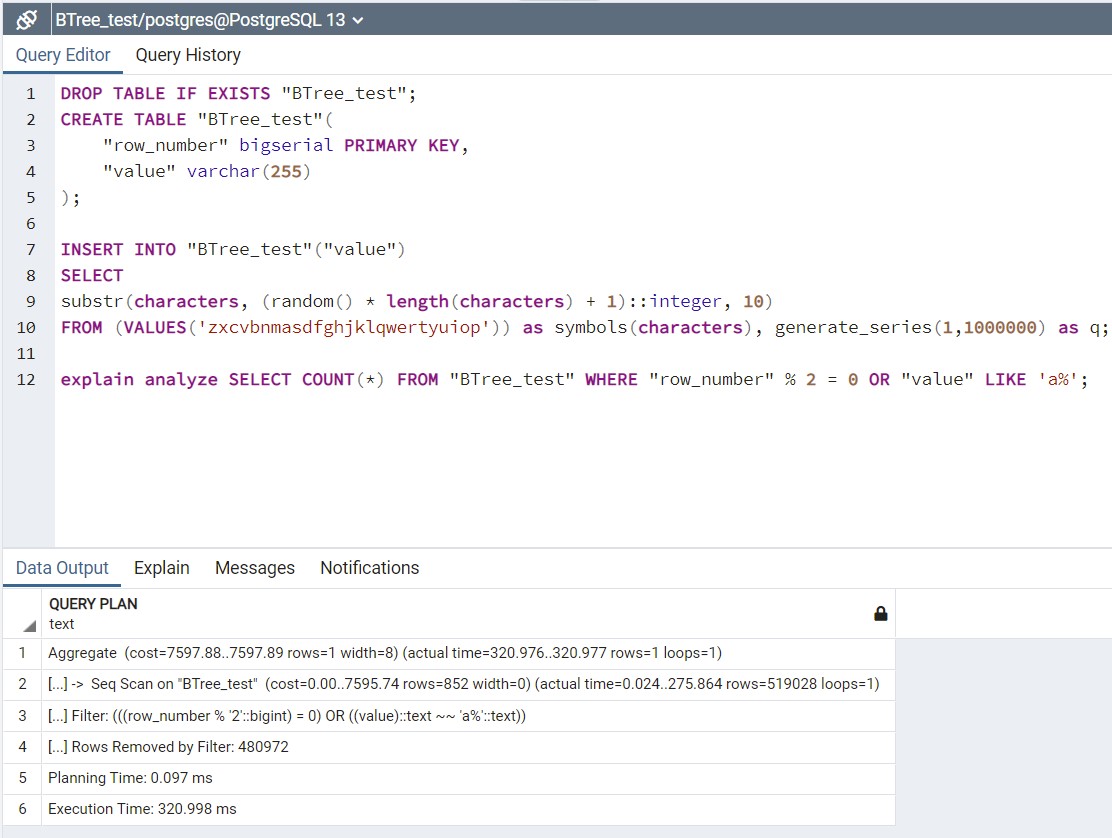
**Робота з індексами у PostgreSQL**

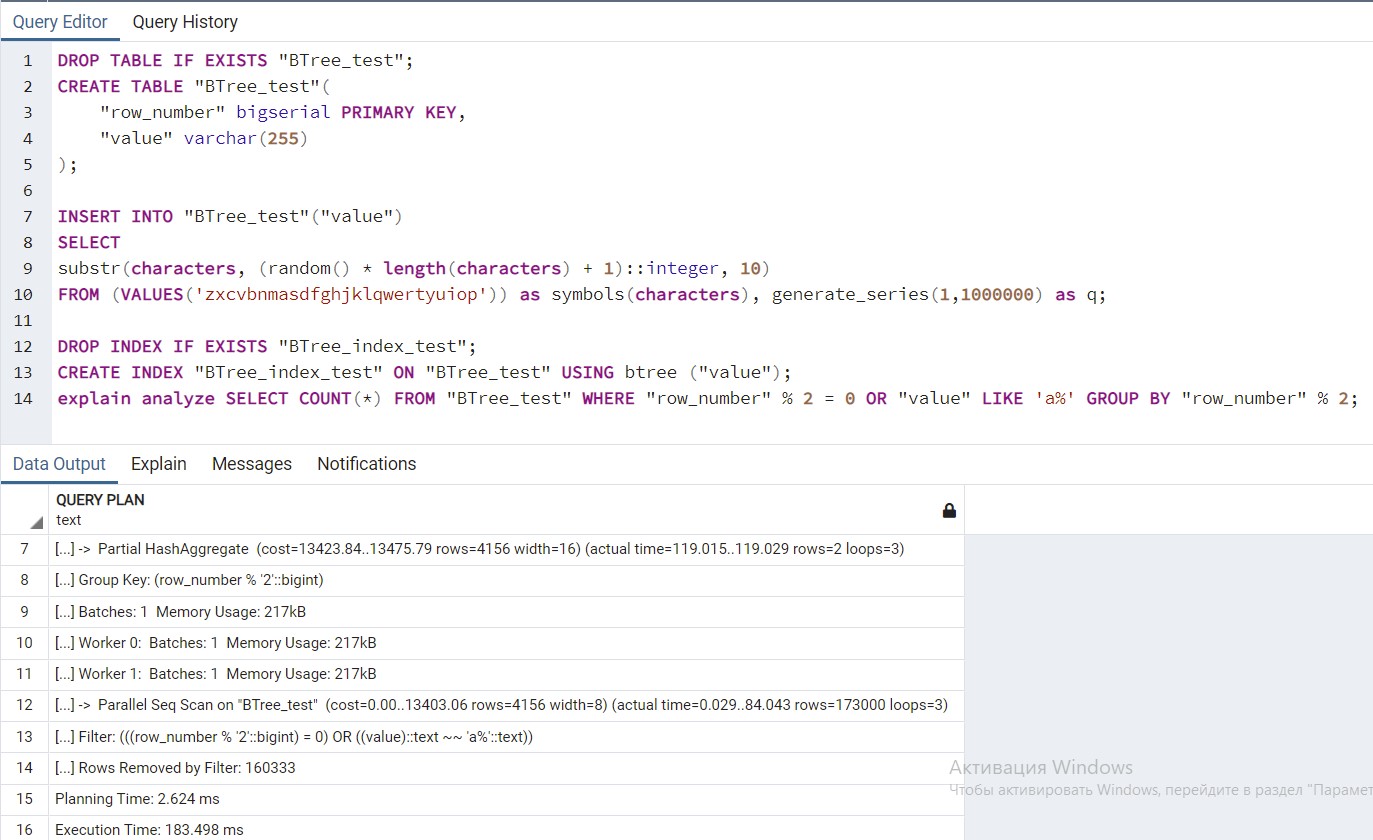
Індекс BTree, також називають B-дерево, придатний для даних, які можна відсортувати. Тобто для тих типів, для яких є визначенні наступні оператори: “більше”, “менше”, “більше чи одно”, “менше чи одно” та “рівно”. Одні й ті самі дані інколи можна сортувати різним чином, що повертає нас до концепції сімейства операторів. Індексні записи В-дерева упаковані у сторінки. У листкових сторінках ці записи містять індексовані дані (ключі) та посилання на рядки таблиці (TID-и); у внутрішніх сторінках кожен запис посилається на дочірню сторінку індексу та містить мінімальне значення ключа у цій сторінці.

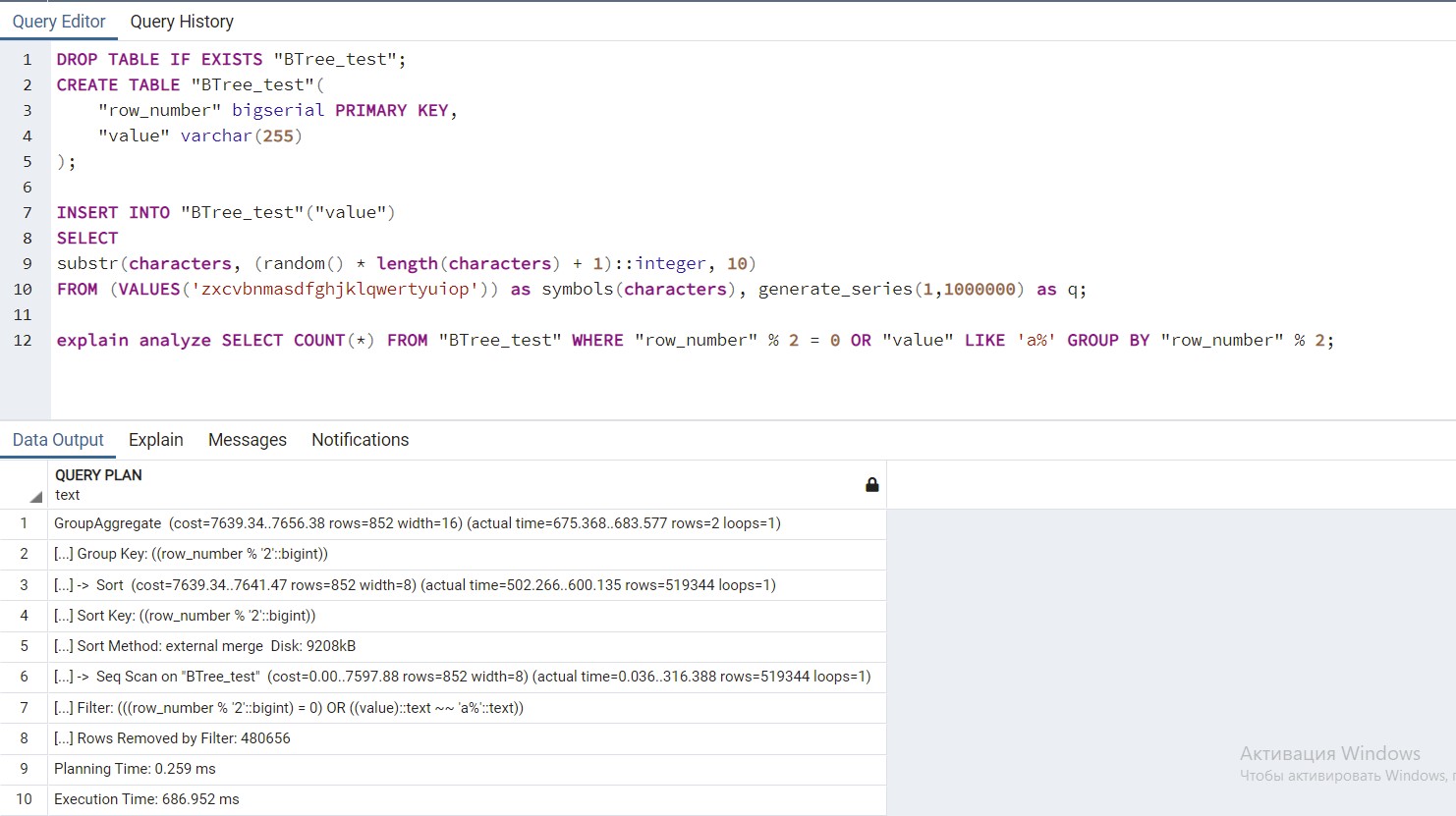
Для дослідження даного виду індексу було створено таблицю, яка має два стовпці: числову та текстову. Вони проіндексовані як BTree. Для таблиці було згенеровано 1 000 000 записів.

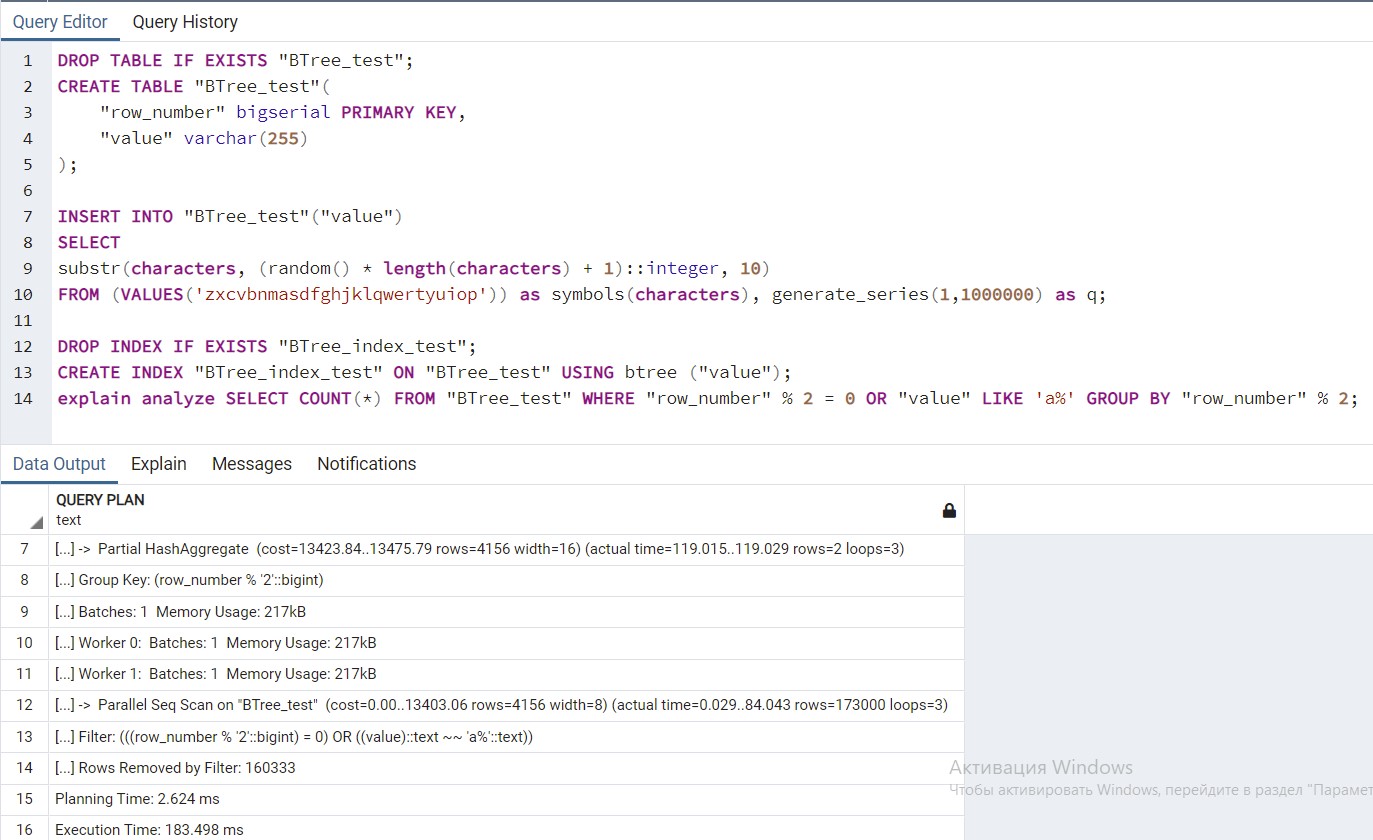
  
Рисунок 25 - Виконання першого запиту без використання індекса

  
Рисунок 26 - Виконання першого запиту з використанням BTree індекса

  
Рисунок 27 - Виконання другого запиту без використання індекса

  
Рисунок 28 - Виконання другого запиту з використанням BTree індекса

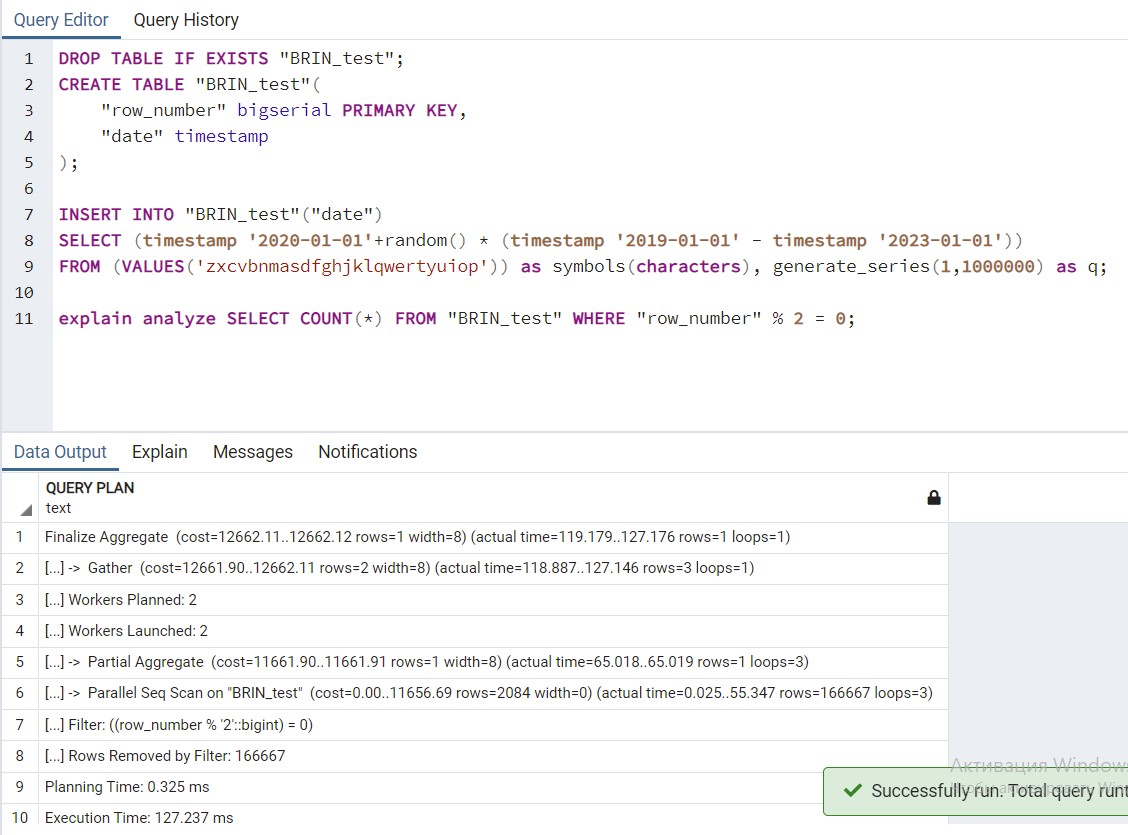
  
Рисунок 29 - Виконання третього запиту без використання індекса

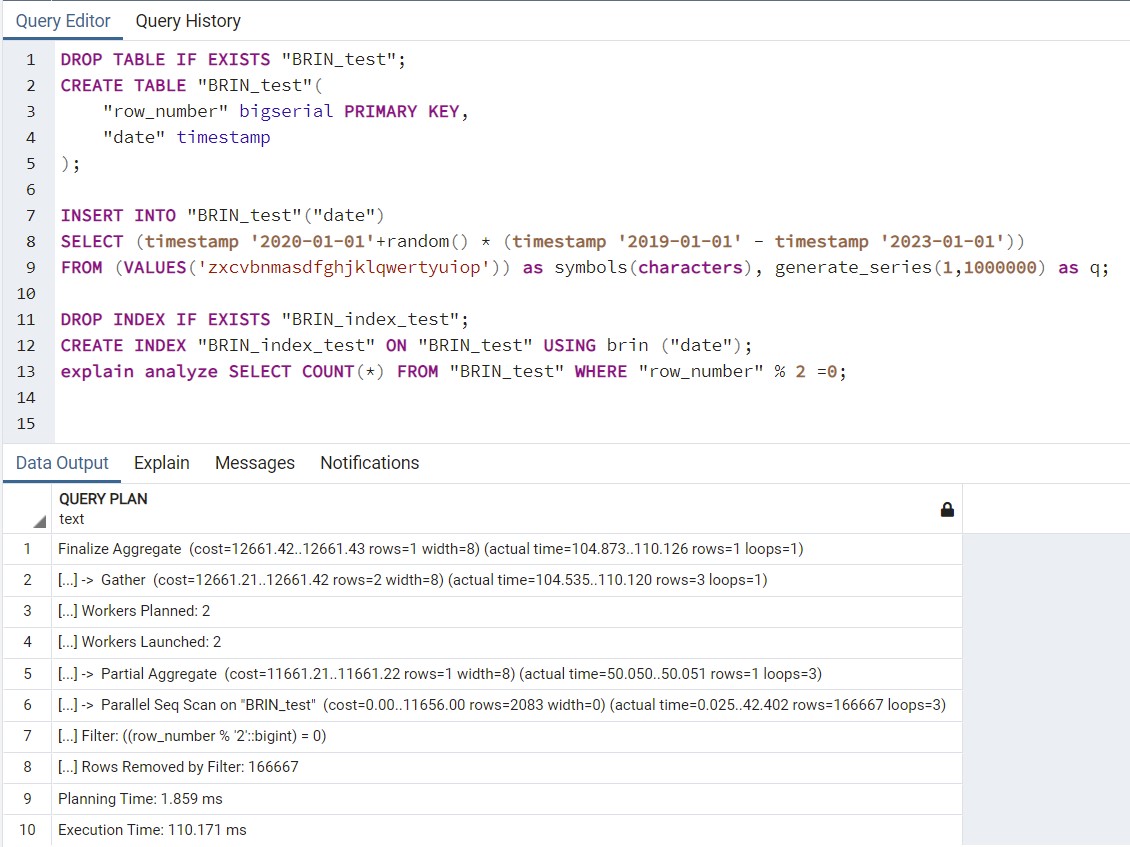
  
Рисунок 30 - Виконання третього запиту з використанням BTree індекса

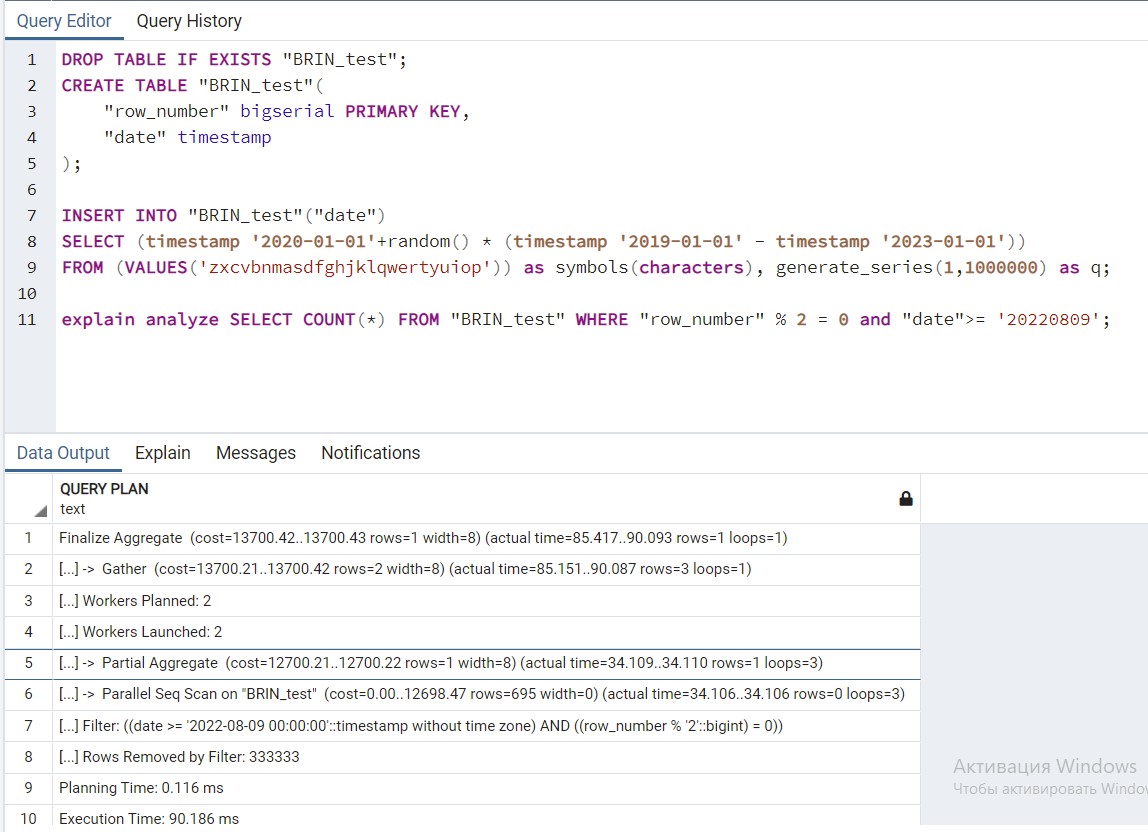
Індекс BRIN, або Block Range Index, головна ідея якого полягає у не знаходженні необхідного значення, а в уникненні перегляду свідомо непотрібних.

Даний вид добре працює для стовпців, де значення корелюються із їх фізичним положенням у таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання.

Для дослідження даного виду індексу було створено таблицю, яка має два стовпці: t\_data типу timestamp without time zone та t\_number типу integer. Для стовпця t\_data індексація проводилася як BRIN. Для таблиці було згенеровано 1 000 000 записів.

  
Рисунок 31 - Виконання першого запиту без використання індекса

  
Рисунок 32 - Виконання першого запиту з використанням BRIN індекса

  
Рисунок 33 - Виконання другого запиту без використання індекса

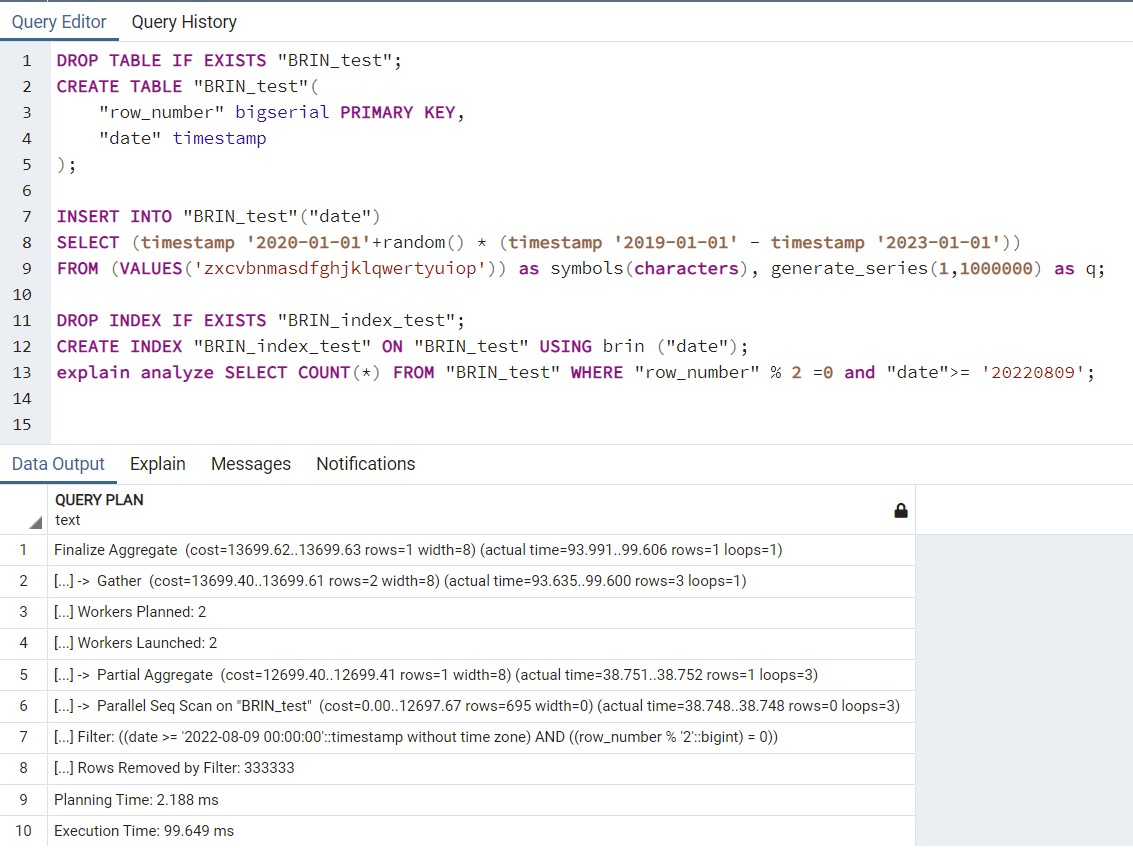
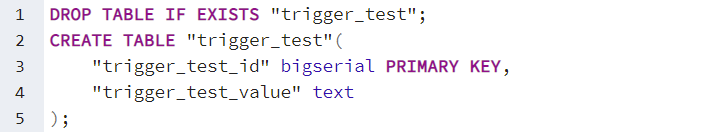
  
Рисунок 34 - Виконання другого запиту з використанням BRIN індекса

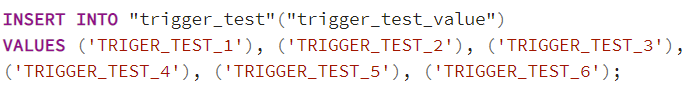
  
Рисунок 35 - Виконання третього запиту без використання індекса

  
Рисунок 36 - Виконання третього запиту з використанням BRIN індекса

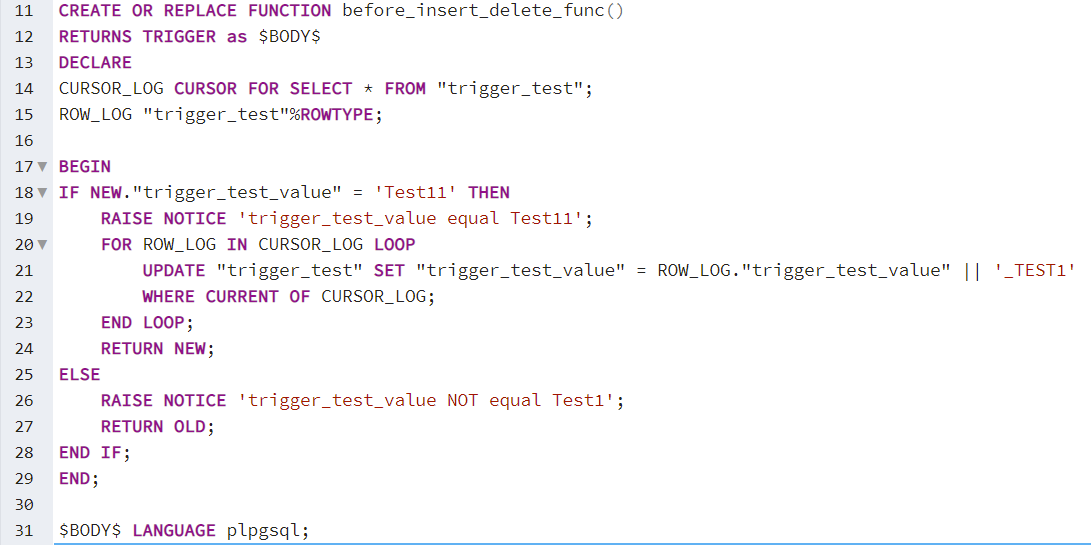
# **Завдання №3**

Створюємо таблицю на якій будемо проводити тестування тригера.

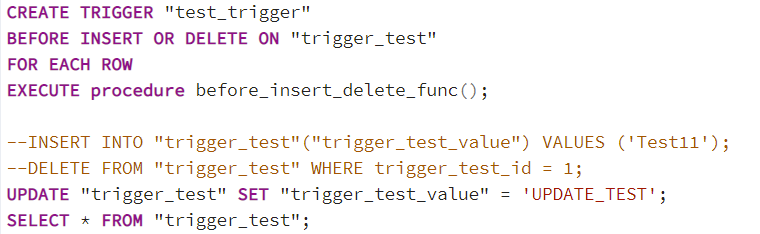
Вносимо початкові дані до створеної таблиці.



Текст розробленого тригера.



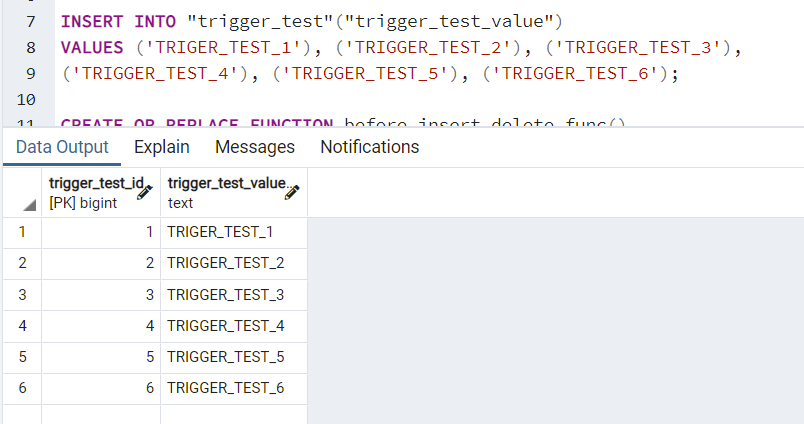
Команди, що ініціюють виконання тригера.

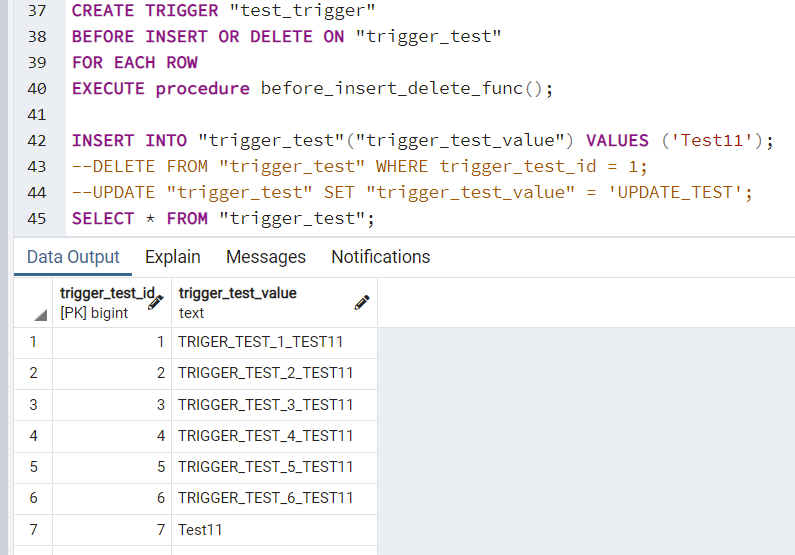


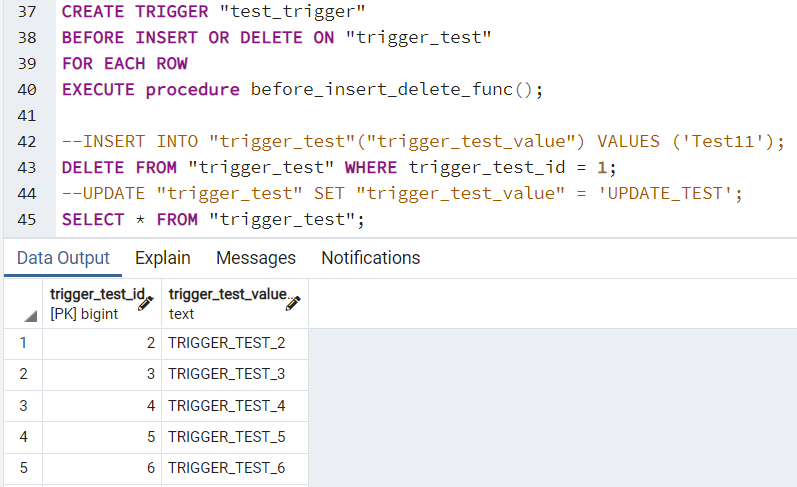
**Принцип роботи**

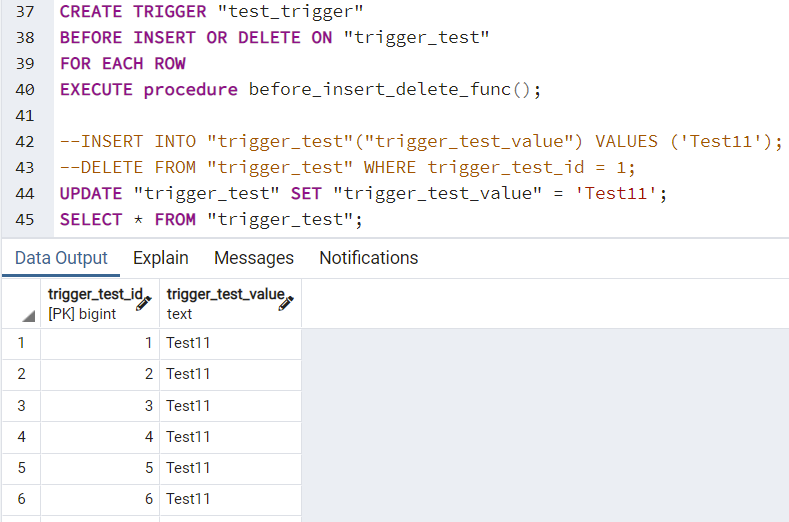
Тригер “test\_trigger” стає активним до того, як відбудеться директива вставки нового рядка або видалення певного, вже існуючого, рядка таблиці “trigger\_test”.Якщо значення “trigger test value” буде рівне значенню Test11, то до кінців кожного елемента стовпця “trigger test value” додастся значення “\_Test11”, та виводиться повідомлення 'trigger\_test\_value equal Test11' та повертає NEW. А при запиті на видалення виводить повідомлення 'trigger test value NOT equal Test11' та повертає OLD.

**Тестування тригера**

****Рисунок 37 - Вставка початкових значень до таблиці

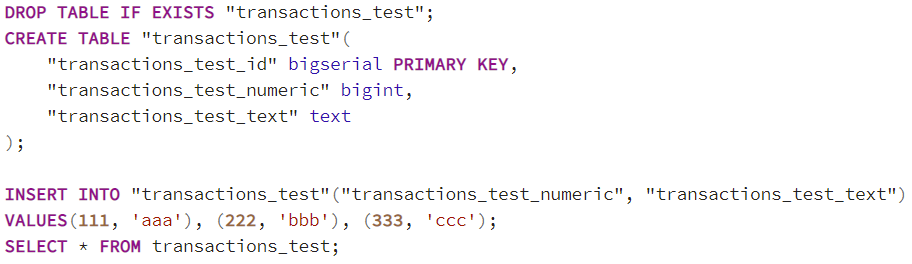
  
Рисунок 38 - Виконання та результати роботи тригера для запиту вставки.

  
Рисунок 39 - Виконання та результати роботи тригера для запиту видалення.

****Рисунок 40 - Перевірка роботи тригера на дію відмінну від вставки та видалення.

# **Завдання №4**

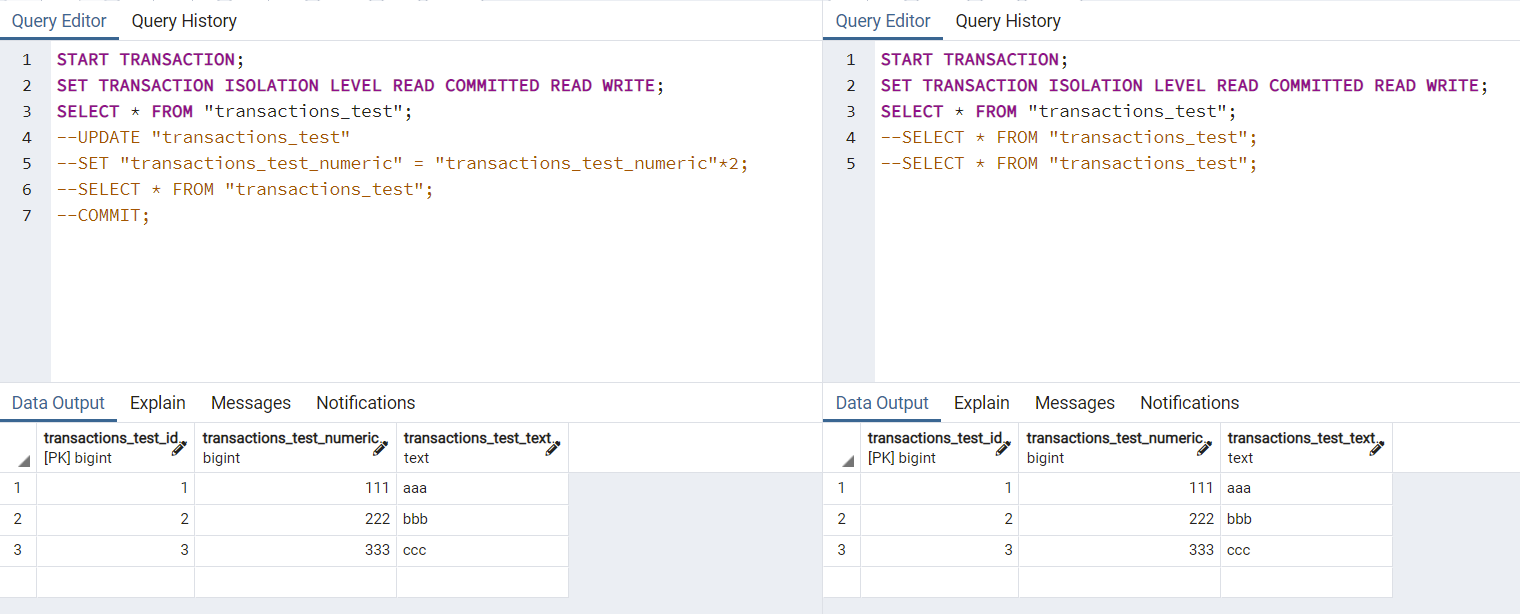
Для даного завдання було створено та заповнено таблицю, яка має наступний вигляд.

  
Рисунок 41 - Створення та заповнення таблиці, для подальшого виконання завдання №4

**READ COMMITTED**

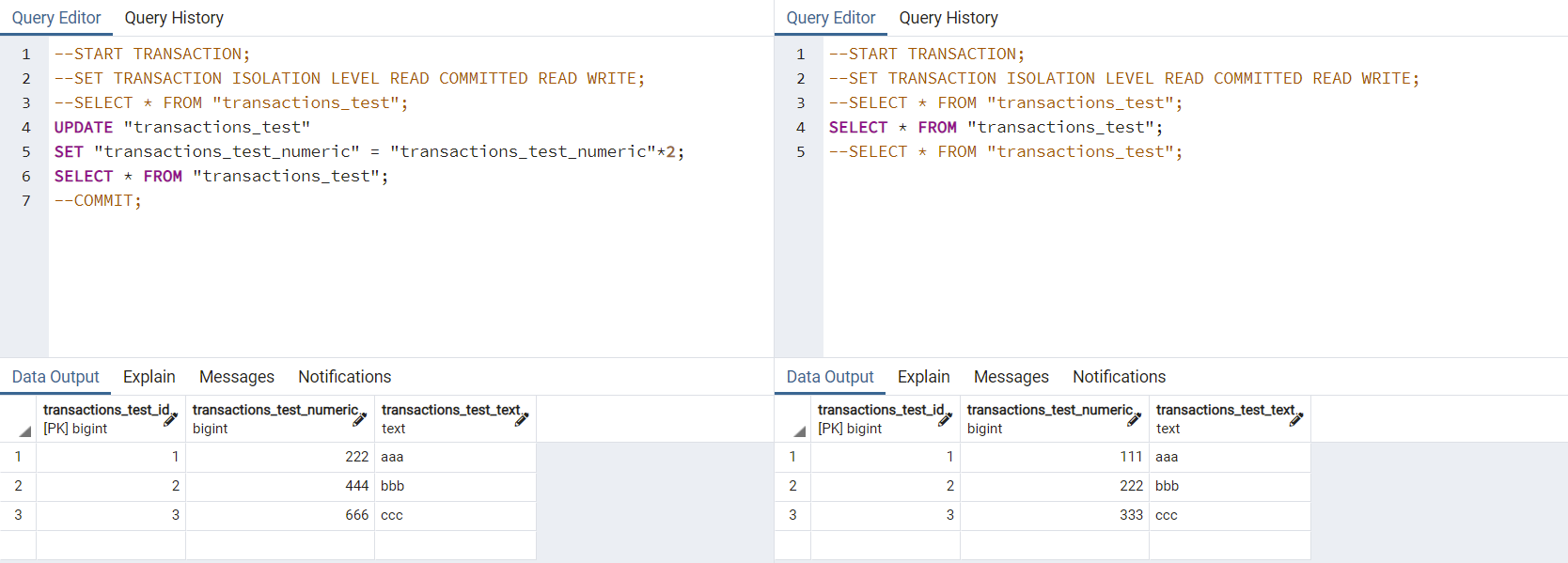
На даному рівні ізоляції одна транзакція не бачить змін у базі даних, викликаних іншою, до тих пір поки та не завершить своє виконання, командою COMMITTED або ROLLBACK.

Спочатку запустимо дві транзакції у двох вікнах й будемо послідовно виконувати команди.

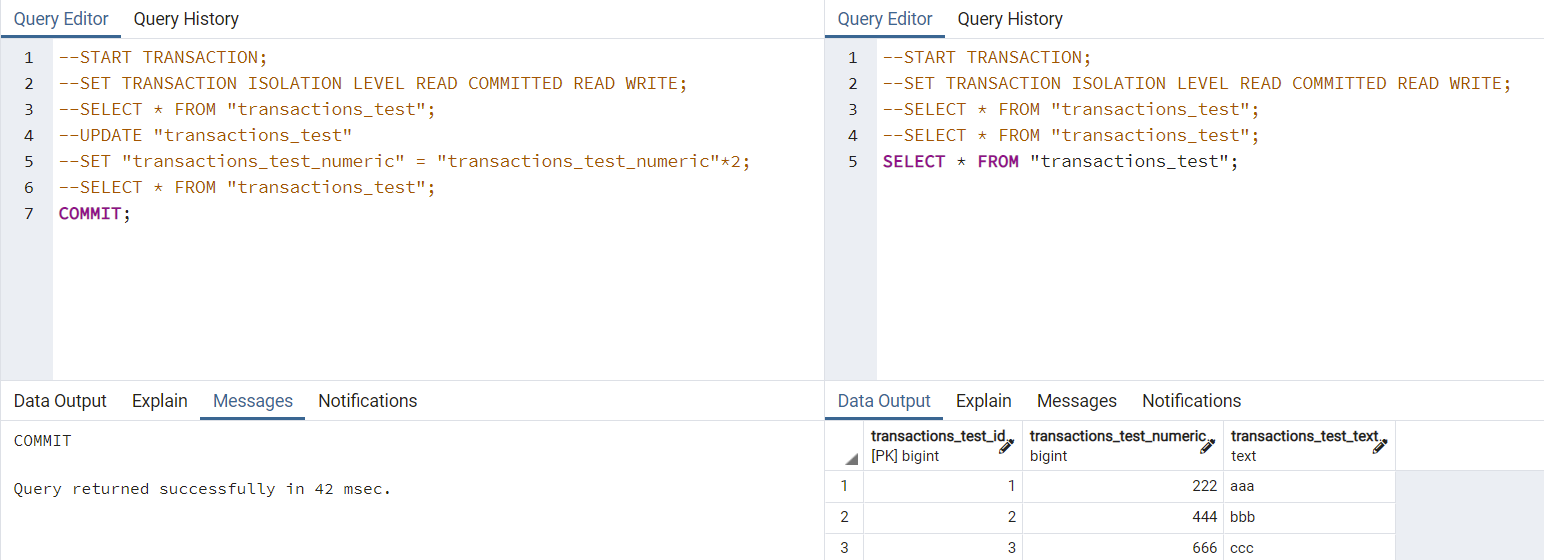
  
Рисунок 42

Бачимо, що отримані результати у двох таблицях поки, що однакові.

Тепер виконуємо директиву редагування у першій транзакції та виводимо отримані результати.

  
Рисунок 43

Після виконання даної частини, можна побачити, що дані у першій транзакції змінилися, а у другій залишилися без змін.  
 Тепер у першій транзакції виконаємо commit, а в другій - select.

  
Рисунок 44

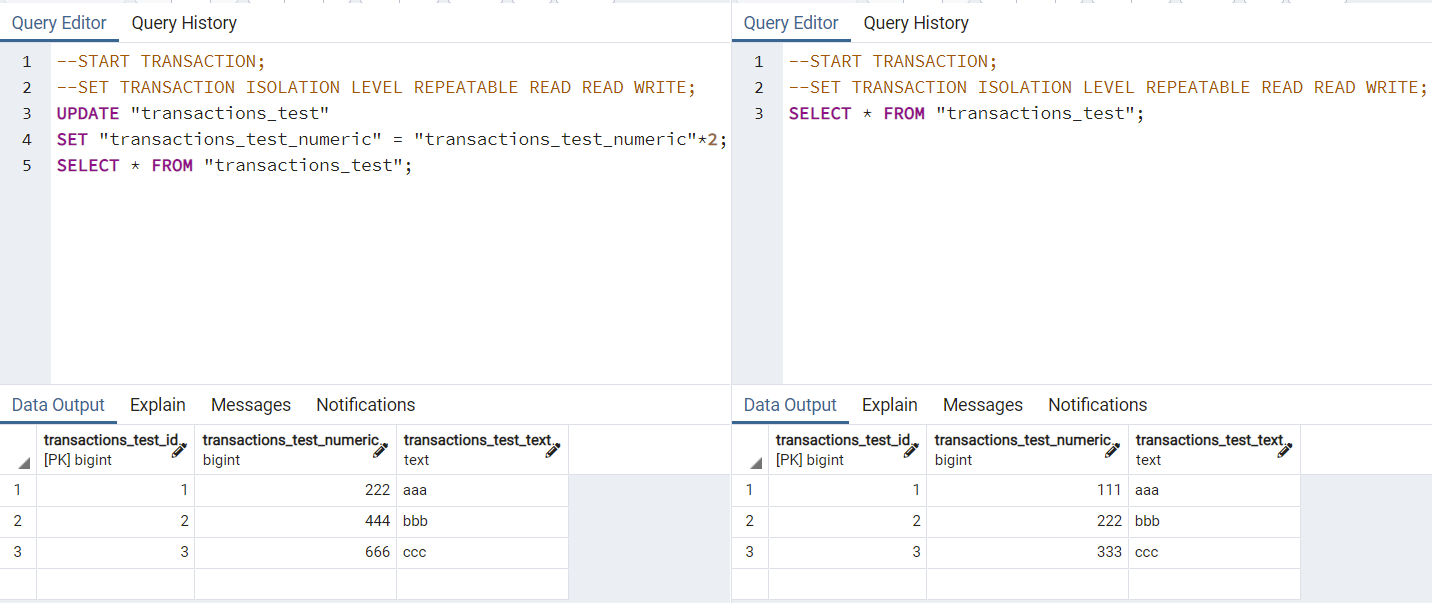
Після виконання даної частини можемо бачити, що у разі завершення першої, друга транзакція виконала запит, вивівши вже ті дані, що були закомічені першою транзакцією.

Таким чином ми маємо феномен, який має назву “Unrepeatable read” у другій транзакції. Друга транзакція ще не закінчена, але ми бачимо вже закомічені дані з першої транзакції. Тому це рівень ізоляції має таку назву, а саме “READ COMMITTED”, що означає, що в транзакція будемо бачити при читанні дані, які в момент читання були закомічені в БД.

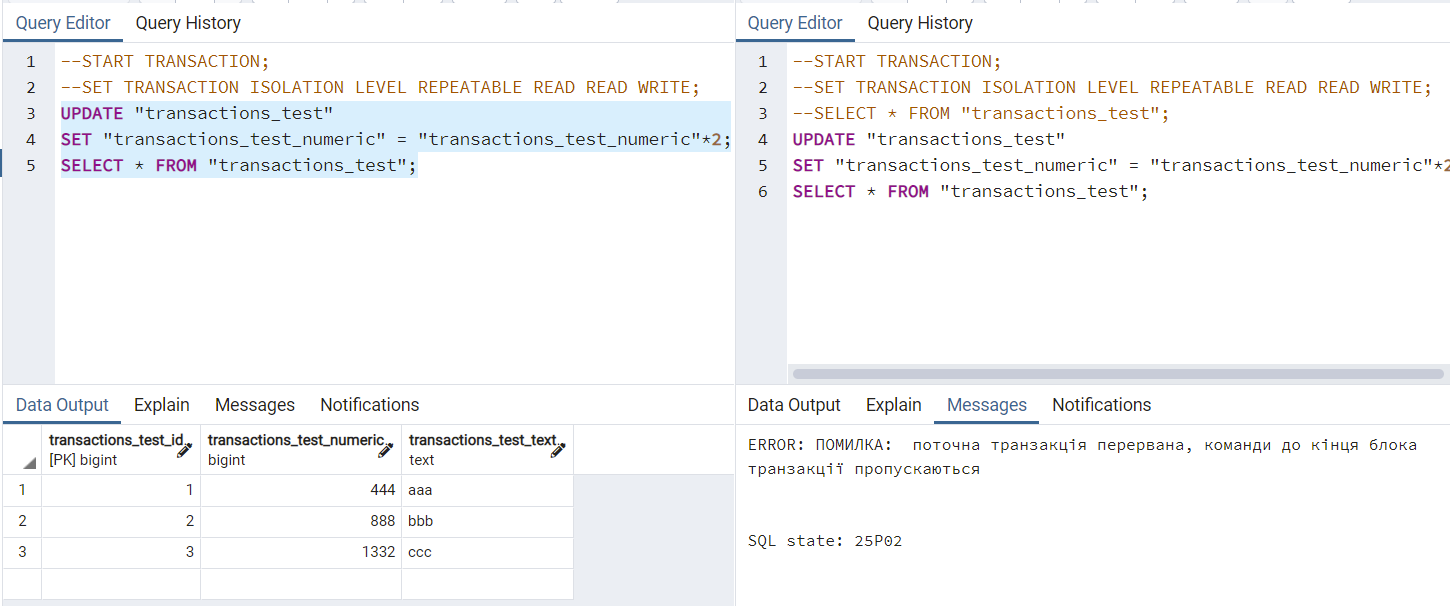
Щоб усунути даний недолік слід перейти до рівня ізоляції, який має назву “REPEATABLE READ”.

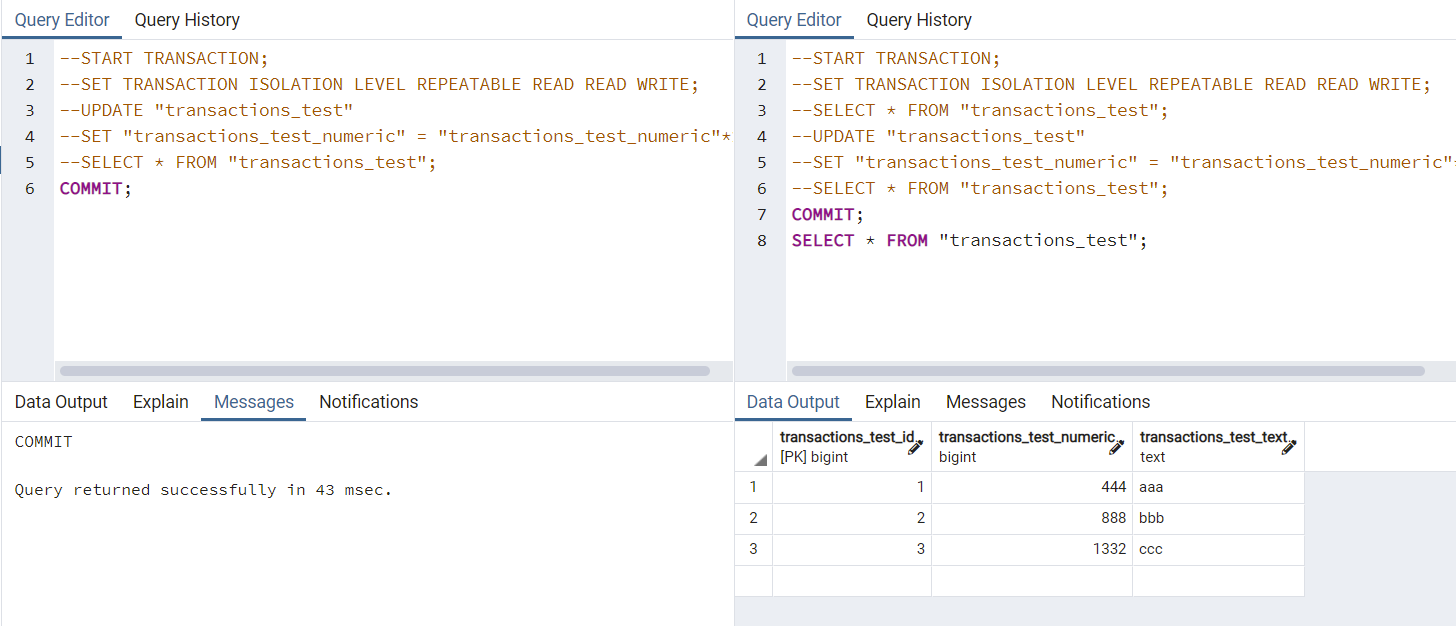
**REPEATABLE READ**

На даному рівні ізоляції друга транзакція також не бачитиме змінені дані першою транзакцією, але також не зможе отримати доступ до тих самих даних.

  
Рисунок 45

На даному рисунку видно, що друга транзакція не бачить змін з першої.

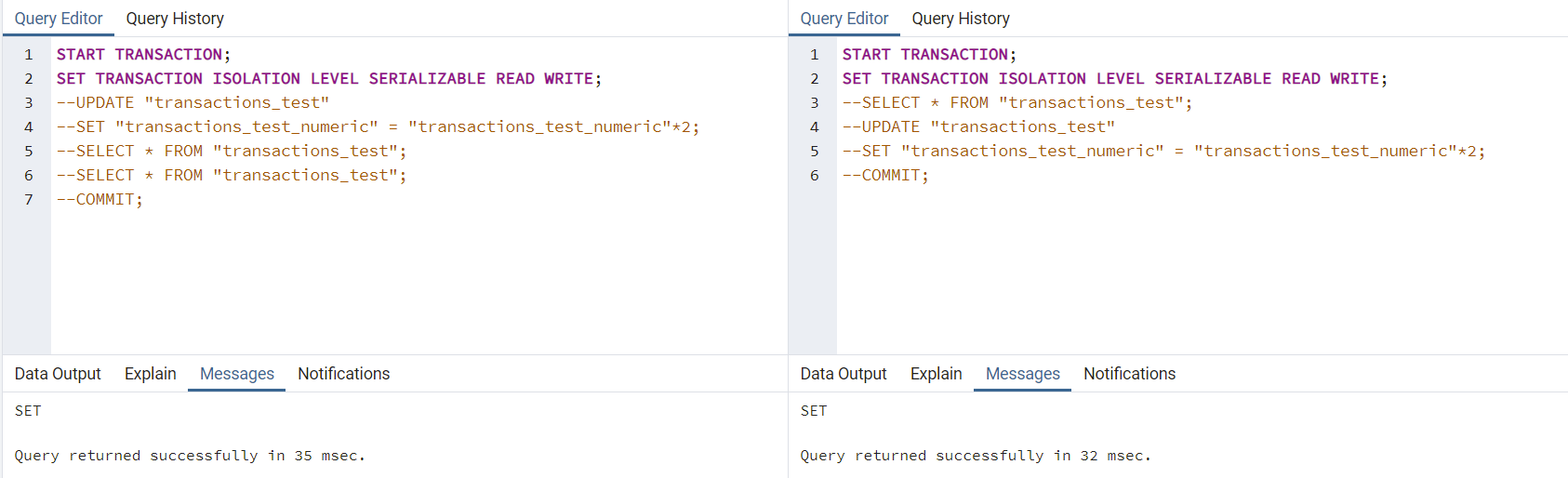
  
Рисунок 46

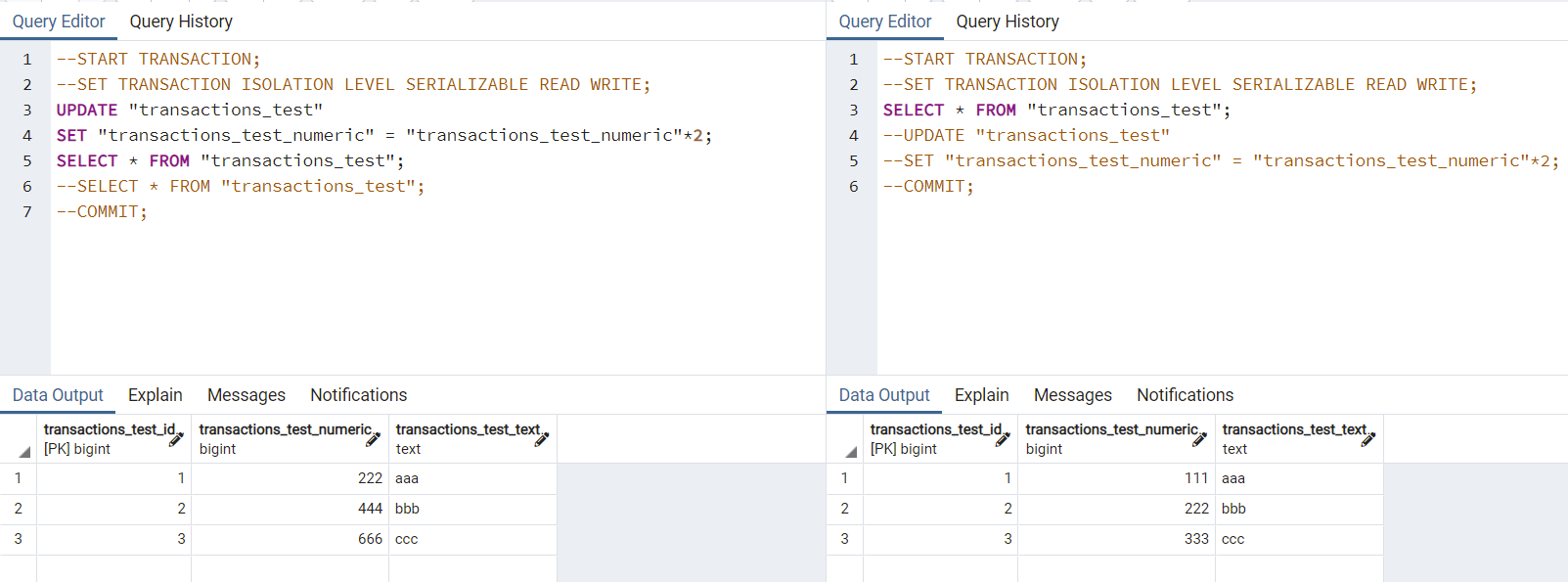
  
Рисунок 47

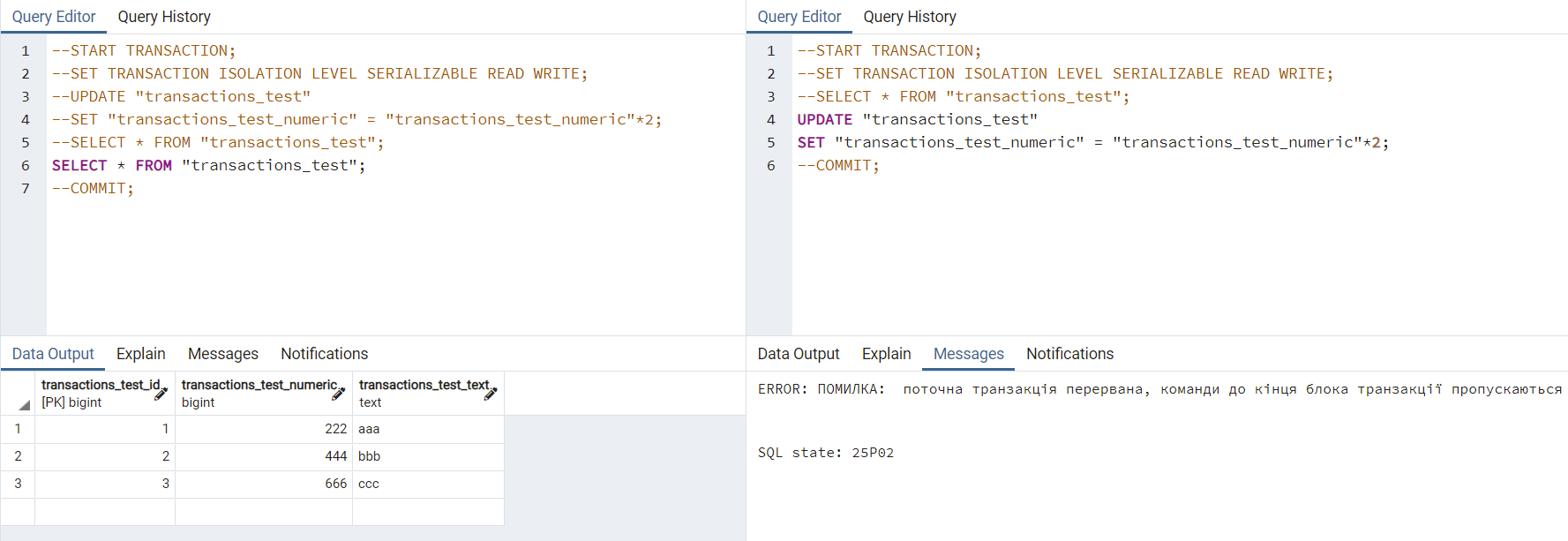
На рисунках 46 та 47 видно, що не виникає читання фантомів, повторного читання, а також заборонено одночасний доступ до незбережених даних. Хоча класично даний рівень ізоляції призначений для попередження повторного читання.

**SERIALIZABLE**

На даному рівні ізоляції транзакції поводять себе так, ніби вони не знають одна про одну, тобто вони не можуть вплинути одна на одну й одночасний доступ суворо заборонений.

  
Рисунок 48

  
Рисунок 49

  
Рисунок 50

Після виконання фрагменту коду (рисунок 49), у першій транзакції все пройшло добре. А от під час виконання фрагменту коду на рис. 3, у другому вікні виникли проблеми, а саме, дане вікно зависло, і після того як було примусово зупинено його роботу, за другою спробою з’явилося повідомлення про помилку.

Що призводить до того, що можна зробити лише 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 даний рівень ізоляції такого не дозволяє. На цьому рівні ізоляції ми отримуємо максимальну узгодженість даних і можемо бути впевнені, що зайві дані не будуть зафіксовані.