A black and white drawing of a building

Description automatically generated

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Засоби оптимізації роботи СУБД PostgreSQL”*

Виконала:

студентка ІІI курсу

групи КВ-21

Оваденко В.М.

Перевірив:

**Київ – 2024**

**Репозиторій GitHub:** <https://github.com/Clemence25/BDZU>

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

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР 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.

**Варіант №14**



У другому завданні проаналізувати індекси Btree, Hash.

Умова для тригера – after insert, update

**Опис предметної області**

Дана предметна область – система управління замовленнями та доставкою косметики. Вона охоплює сукупність процесів і інструментів, які спрямовані на обробку замовлень клієнтів, управління інформацією про товари, організацію доставок та відстеження статусу замовлень. Система забезпечує зручний інтерфейс для клієнтів, де вони можуть замовляти косметичні товари, а також надає інструменти для компаній, щоб контролювати всі етапи від оформлення замовлення до його доставки.

**Опис сутностей**

Для побудови бази даних обраної області, були виділені такі сутності:

1. Клієнт(Customer)

Атрибути: ідентифікатор клієнта, ім’я, електронна скринька, номер телефону.

Призначення: збереження даних щодо клієнтів.

1. Замовлення(Order)

Атрибути: ідентифікатор замовлення, дата.

Призначення: збереження даних щодо замовлень.

1. Продукт(Product)

Атрибути: ідентифікатор товару, назва, ціна

Призначення: збереження даних щодо товарів.

1. Доставка(Delivery)

Атрибути: ідентифікатор доставки, адреса, статус доставки.

Призначення: збереження даних щодо доставок.

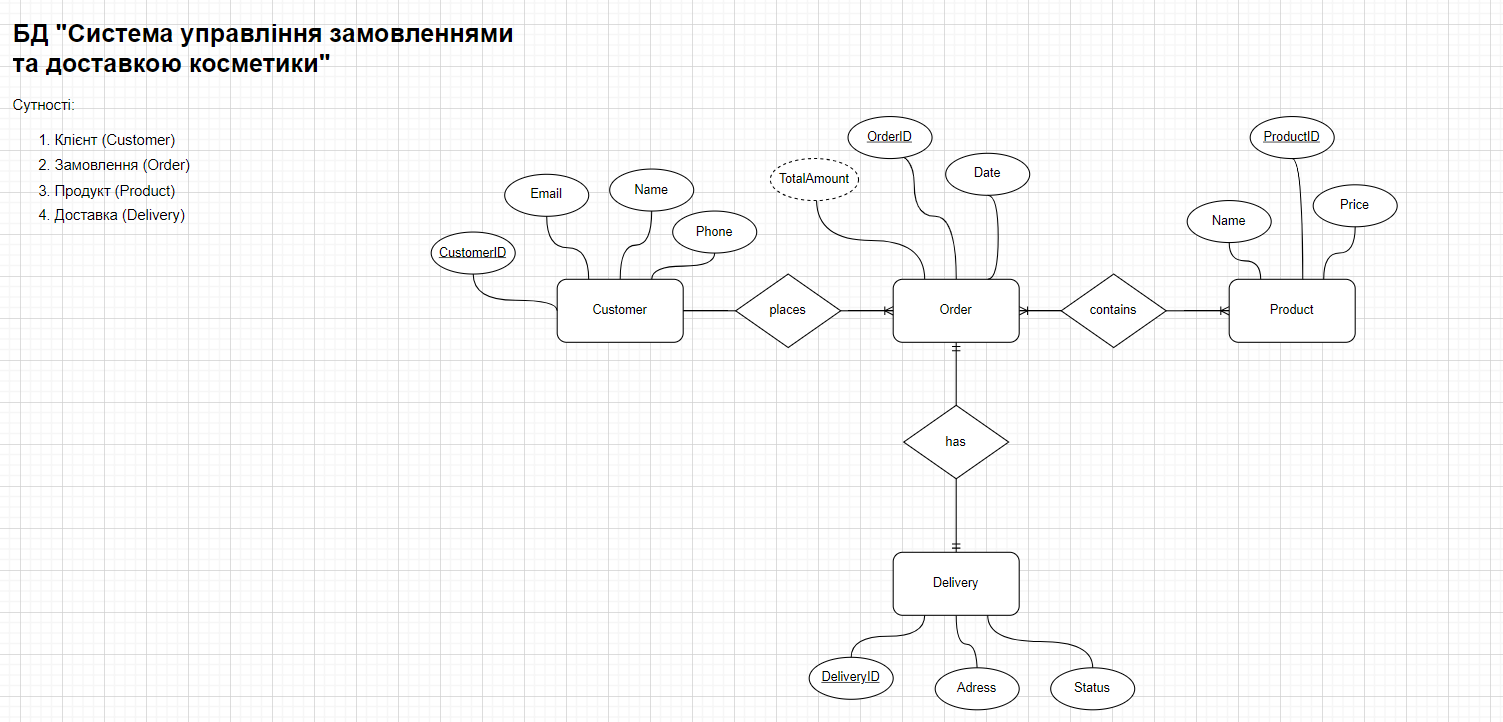
**Опис зв’язків між сутностями**

Зв’язок «Клієнт» - «Замовлення» є зв’язком 1:N. Один клієнт може мати багато замовлень, але кожне замовлення прив’язано лише до одного клієнта.

Зв’язок «Замовлення» - «Продукт» є зв’язком N:M. Одне замовлення може містити багато товарів, і один товар може входити до складу багатьох замовлень.

Зв’язок «Замовлення» - «Доставка» є зв’язком 1:1. Кожне замовлення може мати тільки одну доставку, і кожна доставка відповідає лише одному замовленню.

Графічне подання концептуальної моделі «Сутність-зв’язок» зображено на рисунку 1.



*Рисунок 1 – ER-діаграма, побудована за нотацією Пташиної лапки (Crow’s foot)*

**Перетворення концептуальної моделі у логічну модель та схему бази даних**

Сутність Customer перетворено в таблицю Customer з первинним ключем customer\_id та атрибутами name, email, phone.

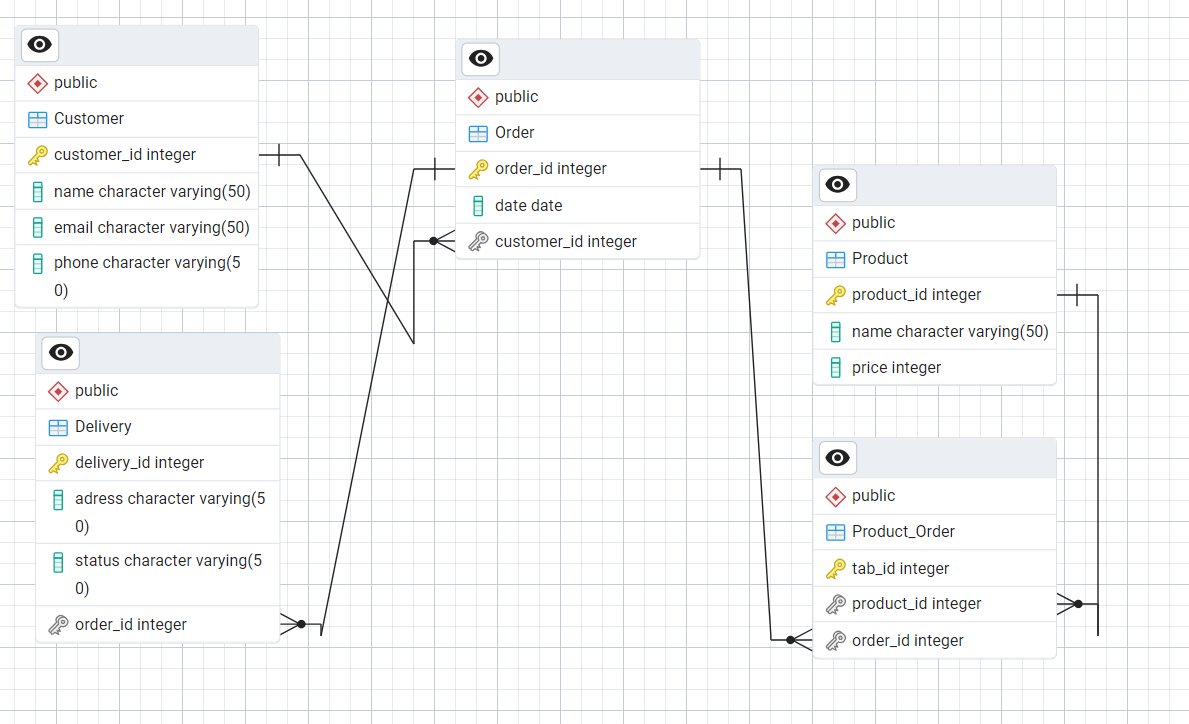
Сутність Order перетворено в таблицю Order з первинним ключем order\_id та атрибутом date та зовнішнім ключем customer\_id.

Сутність Delivery перетворено в таблицю Delivery з первинним ключем delivery\_id та атрибутами adress, status, та зовнішнім ключем order\_id.

Сутність Product перетворено в таблицю Product з первинним ключем product\_id та атрибутами name, price.

Оскільки в логічній моделі безпосередній зв’язок N:M є неможливим, а в концептуальній моделі він існує між сутностями Product i Order, то для його реалізації було створено таблицю Product\_Order, з первинним ключем tab\_id, та зовнішніми ключами product\_id i teacher\_id.

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 2.



*Рисунок 2 – Схема бази даних*

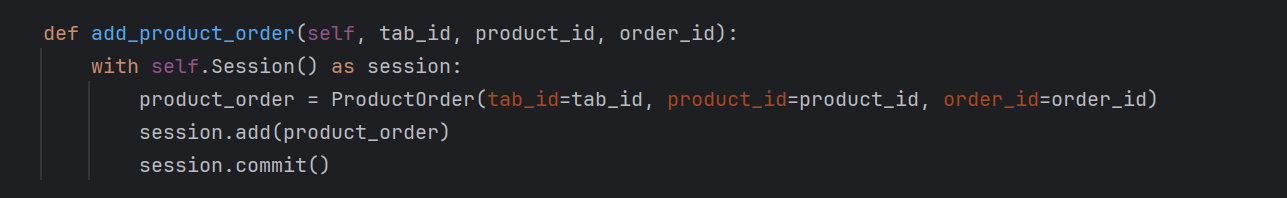
**Завдання №1**

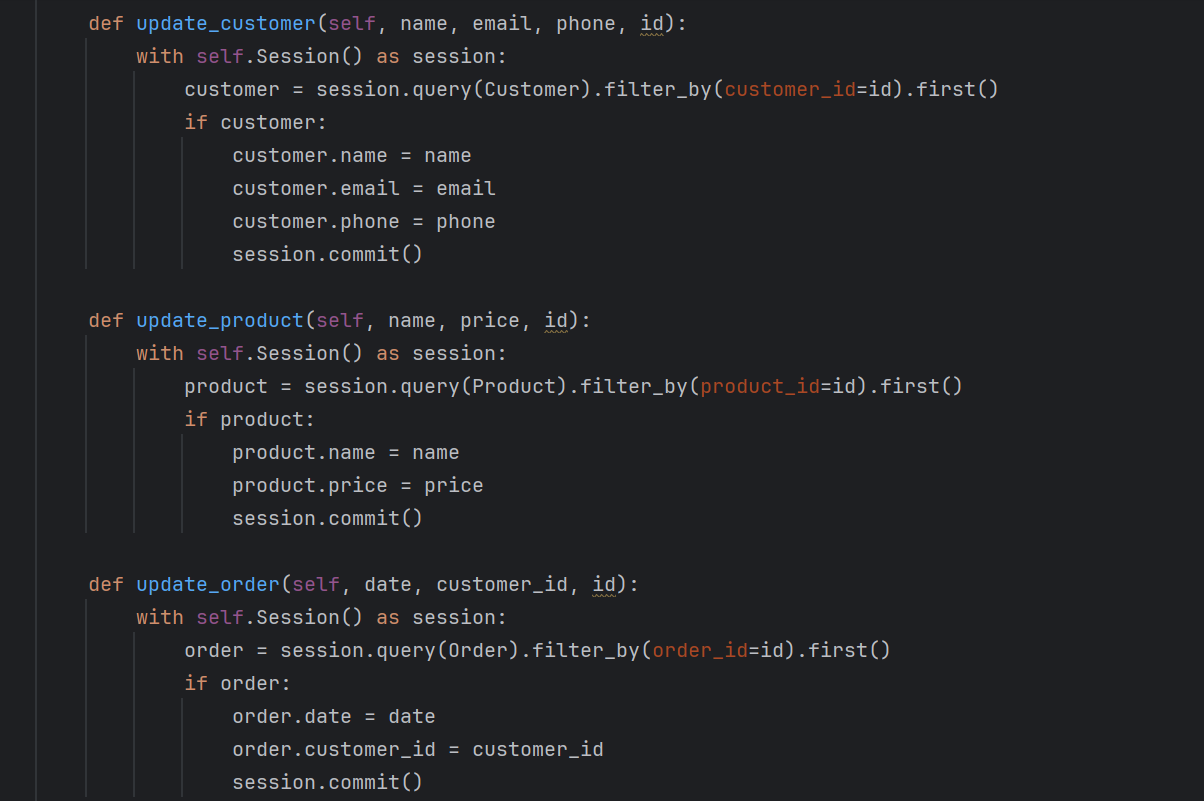
Для перетворення модулів програми, створених в розрахунковій роботі, у вигляд об'єктно-реляційної моделі було використано:

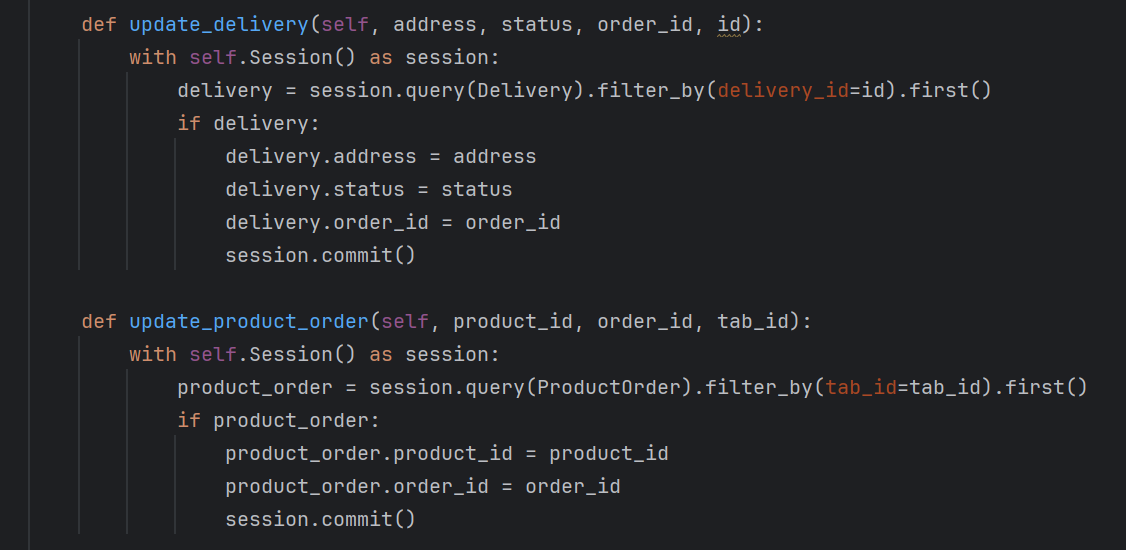
1. Бібліотека для реалізації ORM - SQLAlchemy для Python.
2. Середовище для відлагодження SQL-запитів до бази даних – pgAdmin 4.
3. СУБД - PostgreSQL 16.

Також код буде надано в репозиторії **GitHub**.

Запити на додавання:  

Запити на оновлення:  




Запити на видалення:

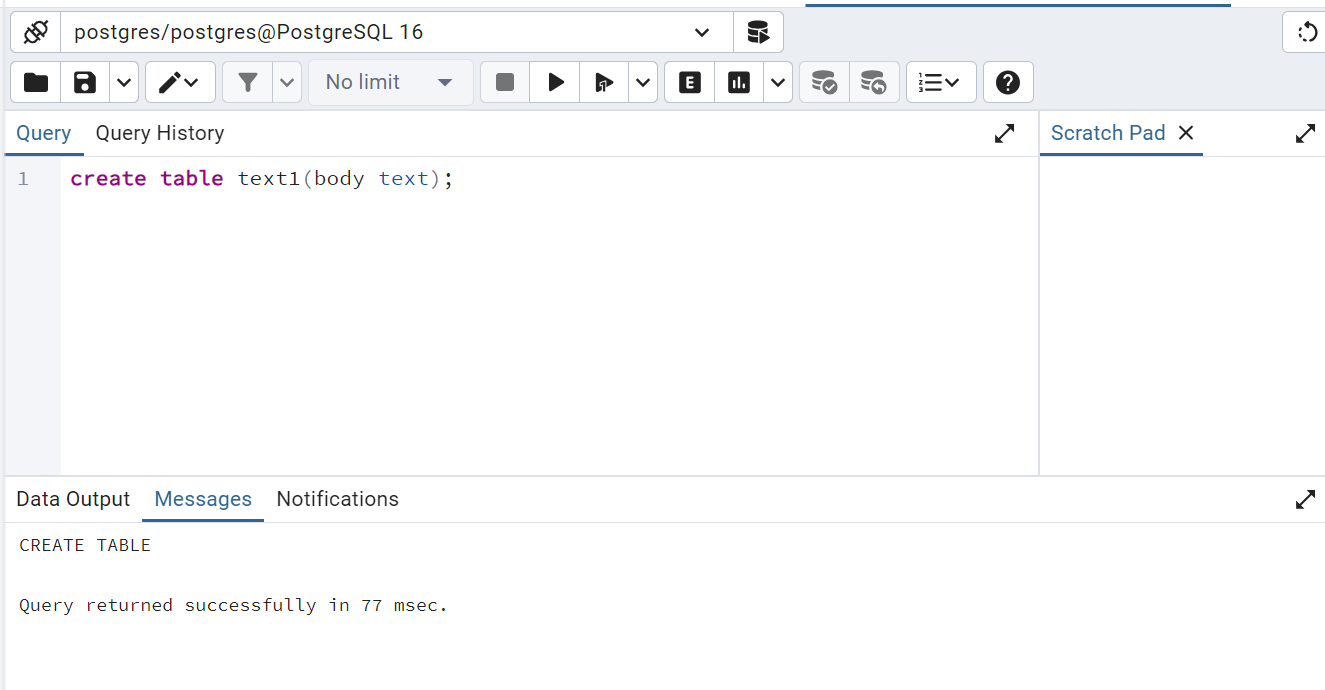




**Завдання №2**

**Створення таблиці:**

create table text1(body text);



**Вставка та генерація даних в таблиці:**

insert into text1

select

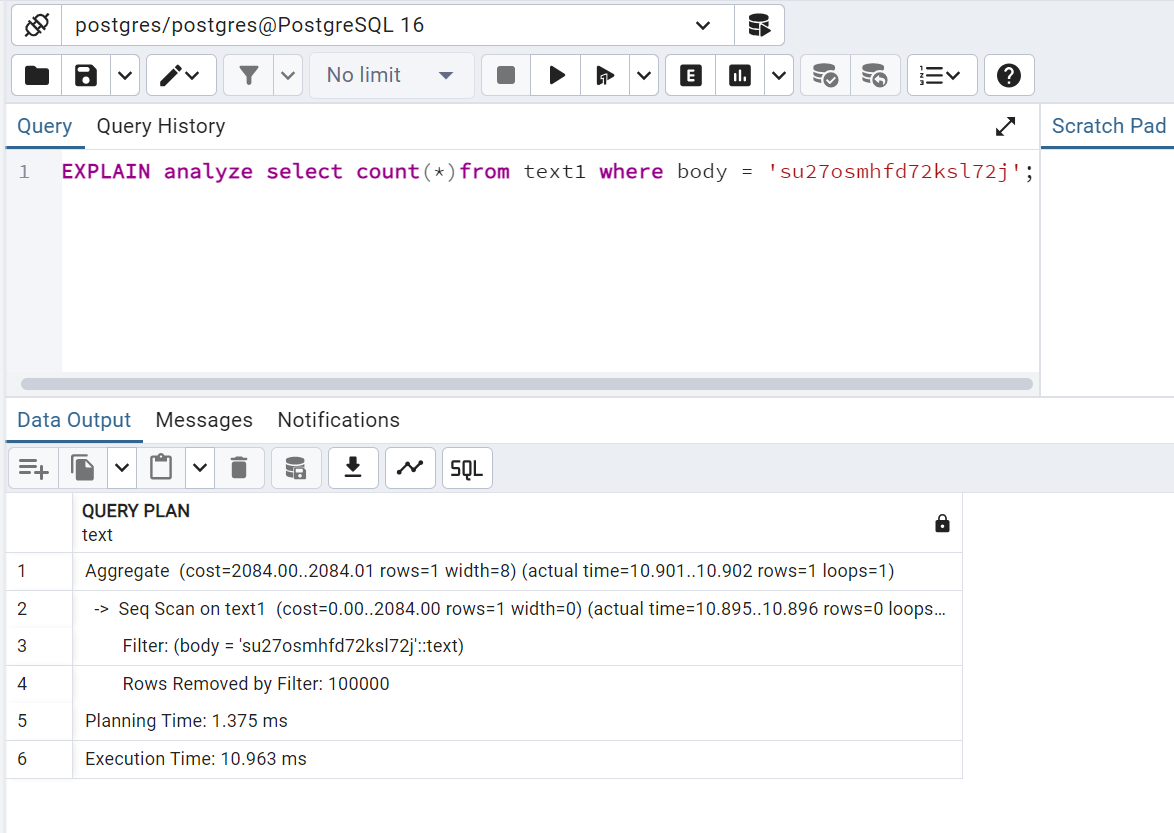
md5(random()::text)

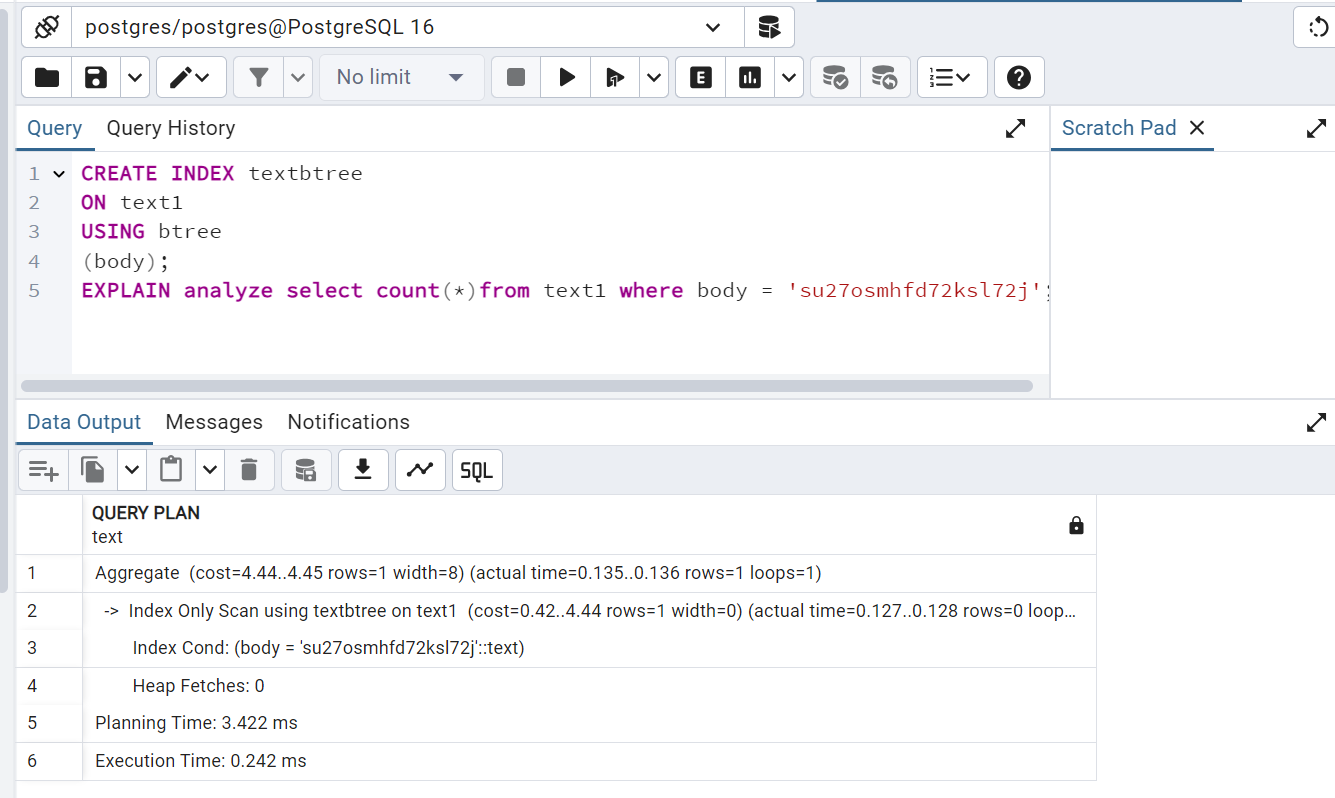
from(select\*from generate\_series(1,100000)as id)as x;



**Btree**

Час виконання без індексу:

****

Час виконання з використанням індексу:  


**Індекс B-дерево (B-tree)** — це структура даних, яка використовується в базах даних для швидкого пошуку, вставки і видалення даних. B-дерева є ефективними для операцій здебільшого в невпорядкованих масивах або наборах даних, таких як сховища ключ-значення, бази даних та файлові системи.

*Основні характеристики B-дерева:*

Балансованість: B-дерево завжди залишається балансованим, що означає, що відстань від кореня до будь-якого листа в дереві має бути приблизною однаковою. Це досягається шляхом перерозподілу ключів при вставці або видаленні.

Висока ефективність пошуку: Благодаря балансованості і структурі B-дерева, час пошуку в середньому O(log n), де n - кількість ключів в дереві.

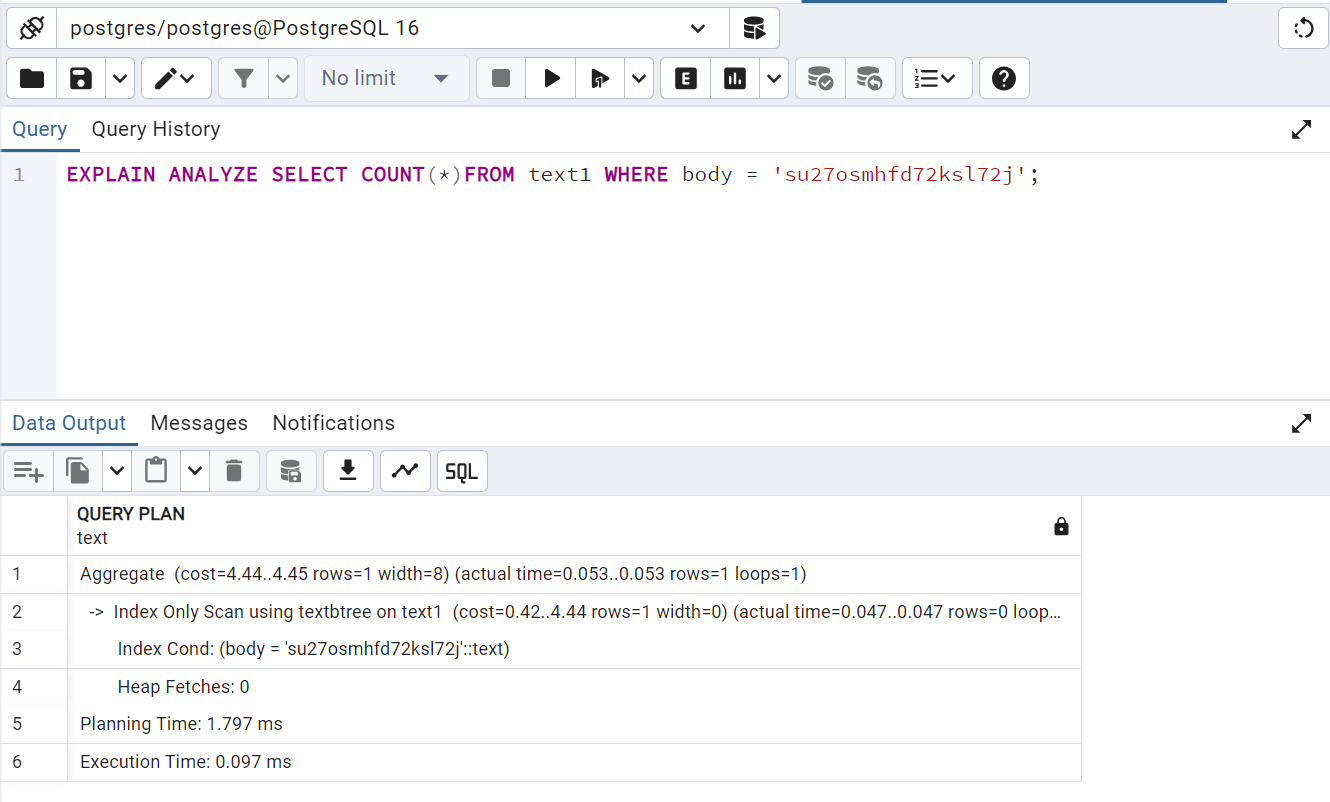
Підтримка діапазонних запитів: B-дерева дозволяють ефективно виконувати діапазонні запити, так як діапазон ключів може бути легко знайдений в порівнянні з бінарними деревами пошуку (BST).

Підтримка вставки та видалення: Операції вставки і видалення можуть бути виконані досить ефективно, оскільки дерево може легко адаптуватися до змін.

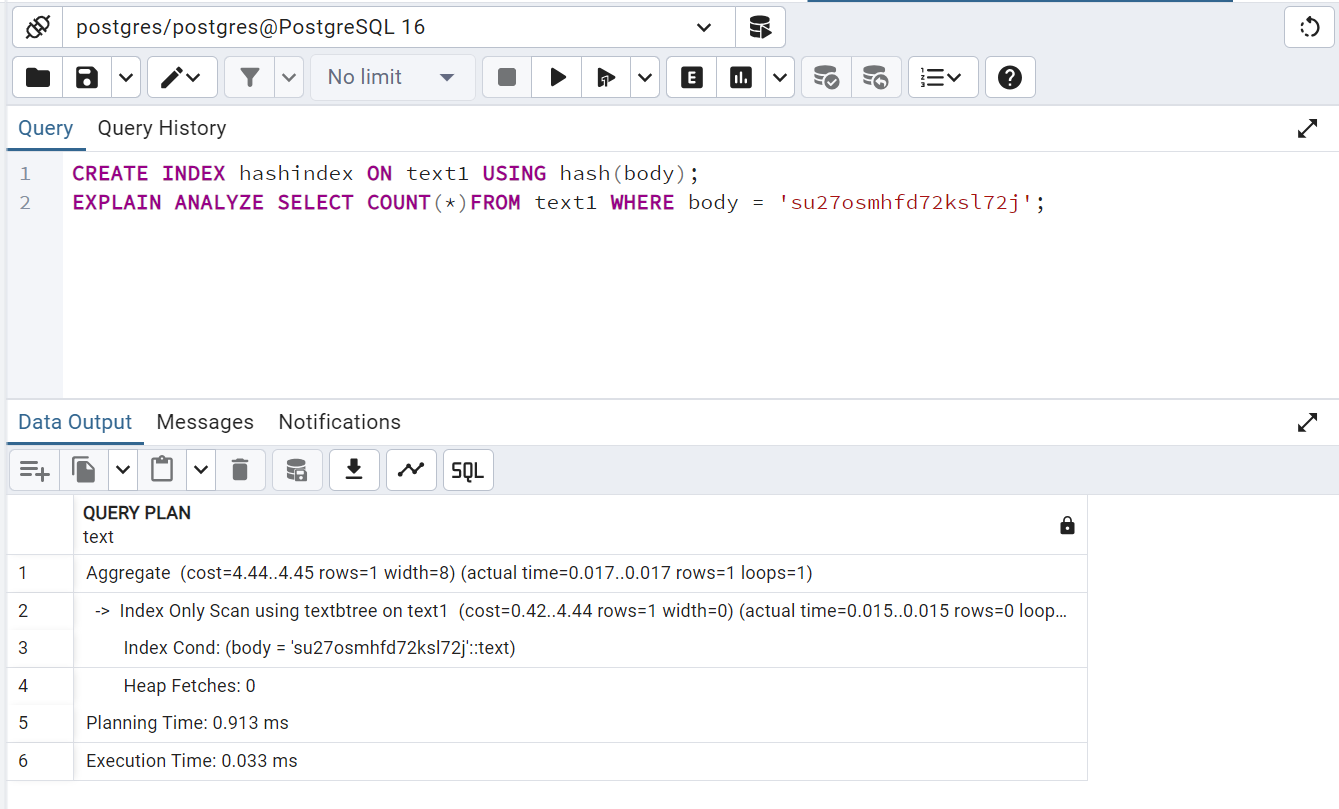
B-дерева широко використовуються в базах даних для індексації даних і покращення швидкодії операцій пошуку. Їх використання дозволяє прискорити доступ до даних, зменшити кількість блокувань і покращити загальну продуктивність бази даних.

**Hash**

Час виконаня без індексу:



Час виконання з використання індексу:



**Хеш-індекс** — це тип індексу в базі даних, який використовує хешфункції для швидкого пошуку конкретного ключа в таблиці. Основна ідея полягає в тому, що величина (хеш-код), отримана застосуванням хеш-функції до ключа, використовується для визначення позиції, де має знаходитися відповідний запис в таблиці.

*Основні характеристики хеш-індексів:*

Хешфункція: Хеш-індекс використовує хеш-функцію для генерації унікального коду для кожного ключа. Цей хеш-код служить величиною, яка визначає позицію запису в індексі або основній таблиці.

Швидкий пошук: Основною перевагою хеш-індексу є швидкість пошуку, оскільки, в середньому, час пошуку є константним (O(1)). Однак в реальних умовах може бути необхідно враховувати випадки колізій (два різних ключі мають однаковий хеш-код).

Колізії: Колізії виникають, коли два різних ключі мають однаковий хеш-код. Для вирішення колізій використовуються різні техніки, такі як відкрите адресування (всі ключі, які мають колізію, розташовуються в одній таблиці), ланцюжки (створення списків для розташування всіх ключів з однаковим хешем) та інші.

Використання пам'яті: Хеш-індекси можуть вимагати значно менше пам'яті порівняно з іншими видами індексів, оскільки вони не зберігають велику кількість додаткової інформації.

Непідтримка діапазонних запитів: Однак хеш-індекс не підтримує ефективний пошук для діапазонів ключів, оскільки вони розташовані випадковим чином.

Хеш-індекси є ефективними для оперативного пошуку точних значень великої кількості даних, але важливо уникати ситуацій, коли колізії стають частими, так як це може суттєво вплинути на швидкість виконання операцій.

**Завдання №3**

***after insert***

Тригер автоматично створює запис у таблиці “Delivery” після додавання нового запису до таблиці “Order”. Це дозволяє автоматизувати створення замовлень доставки для нових замовлень, забезпечуючи збереження цілісності даних між таблицями.

**Код тригеру:**

CREATE OR REPLACE FUNCTION create\_delivery()

RETURNS TRIGGER AS $$

DECLARE

new\_delivery\_id INTEGER;

BEGIN

INSERT INTO "Delivery" (delivery\_id, address, status, order\_id)

VALUES ((SELECT nextval('delivery\_delivery\_id\_seq') FROM generate\_series(1,1)), 'Address TBD', 'Pending', NEW.order\_id);

RETURN NEW;

EXCEPTION

WHEN OTHERS THEN

RAISE WARNING 'Error in AfterInsertTrigger: %', SQLERRM;

RETURN NULL;

END;

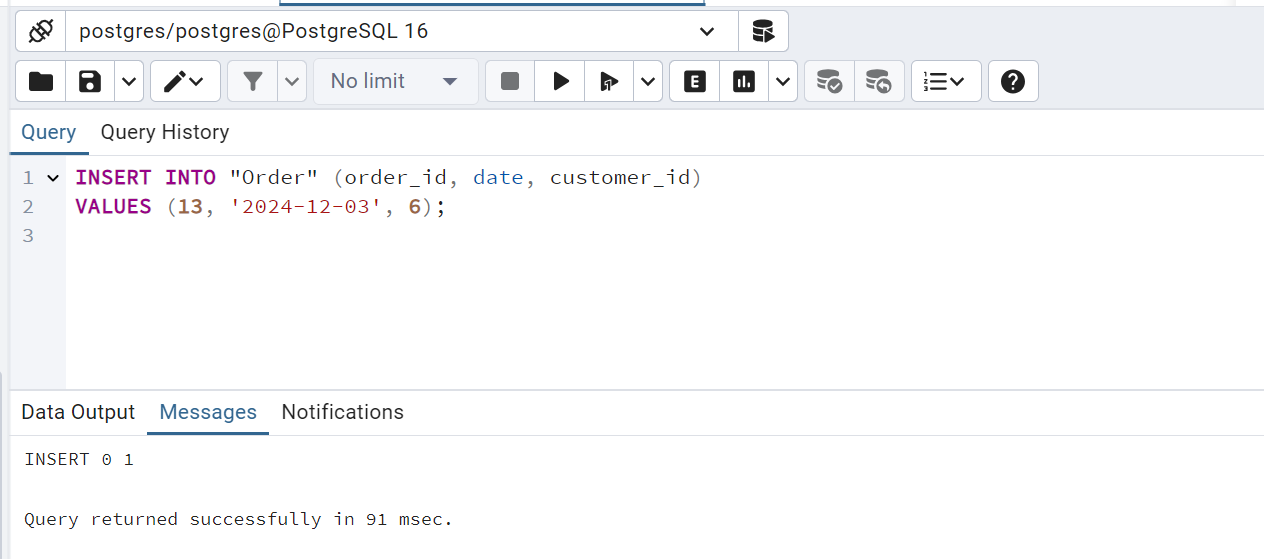
$$ LANGUAGE plpgsql;

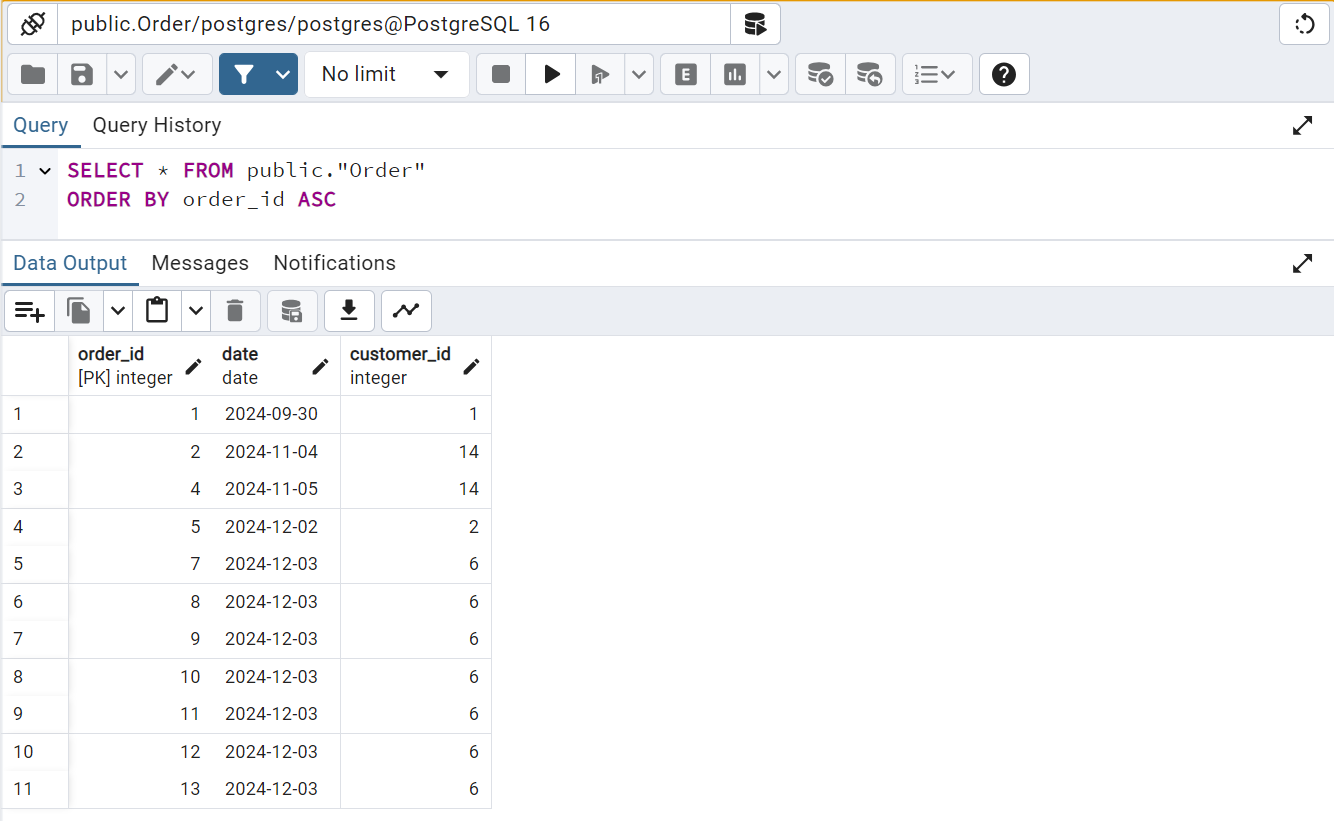
CREATE TRIGGER AfterInsertTrigger

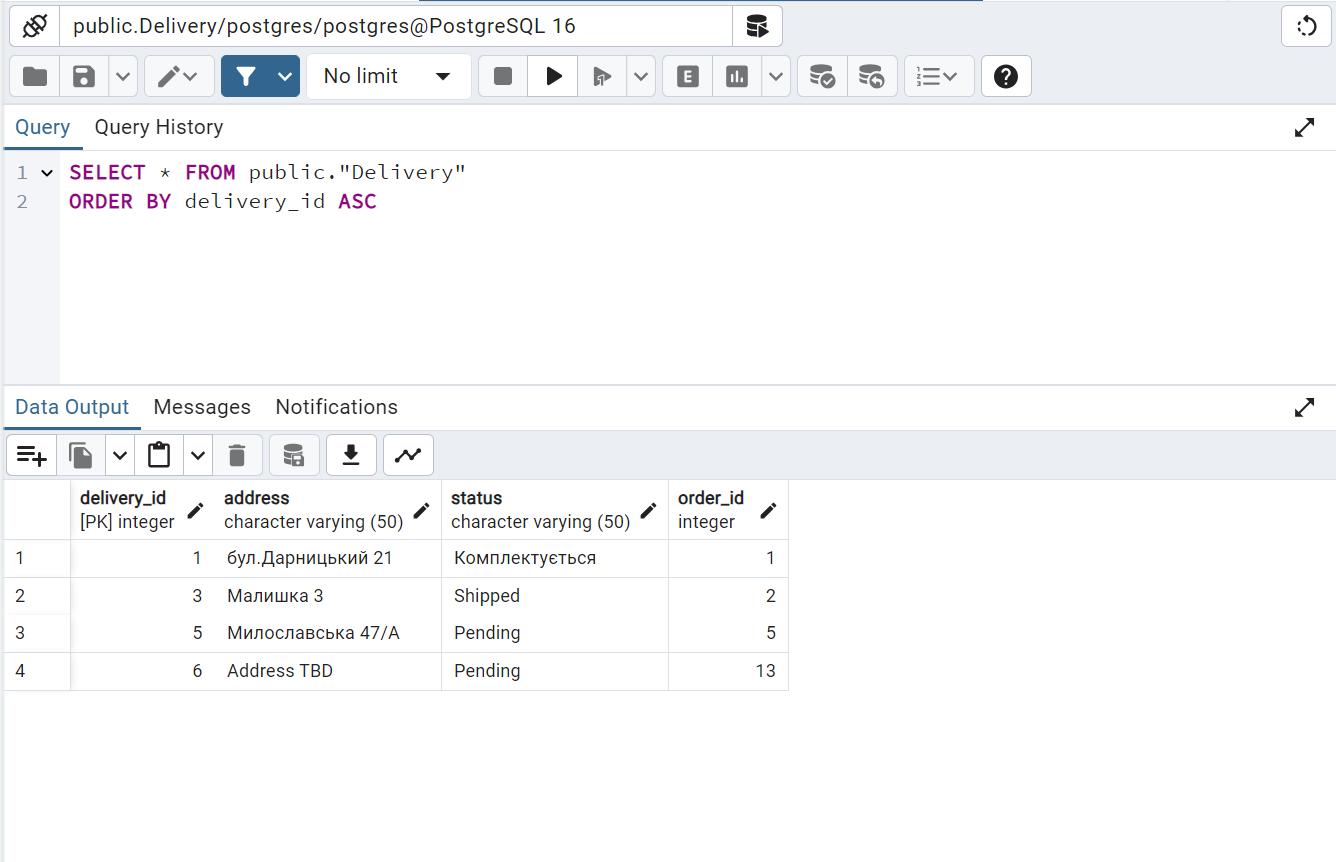
AFTER INSERT ON "Order"

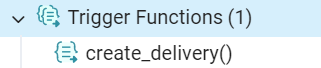
FOR EACH ROW

EXECUTE FUNCTION create\_delivery();

**Тестування роботи тригеру:**  








***After update:***

Тригер UpdateTrigger автоматично спрацьовує після оновлення запису в таблиці "Delivery". Його завдання — створювати записи в таблиці "Event", створеній для тестування цього тригеру, із деталями про зміну статусу доставки status відповідного замовлення.

**Код тригеру:**

CREATE OR REPLACE FUNCTION log\_delivery\_update()

RETURNS TRIGGER AS $$

BEGIN

IF OLD.status = 'Pending' THEN

INSERT INTO "Event" (event\_id, message)

VALUES (

(SELECT nextval('event\_event\_id\_seq')),

'Change status order ' || OLD.order\_id || ' to new status ' || NEW.status

);

ELSE

INSERT INTO "Event" (event\_id, message)

VALUES (

(SELECT nextval('event\_event\_id\_seq')),

'Change status order ' || OLD.order\_id || ' Unknown ' || NEW.status

);

END IF;

RETURN NEW;

END;

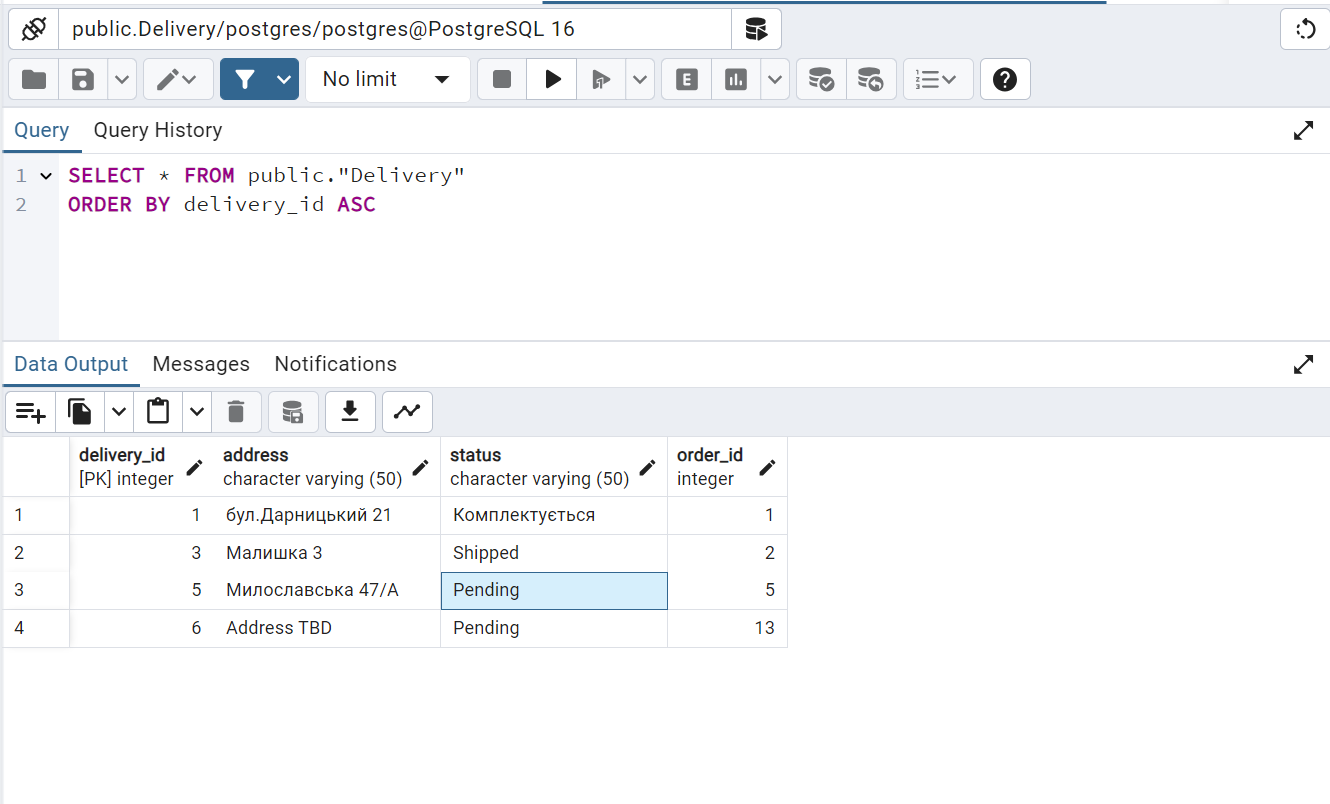
$$ LANGUAGE plpgsql;

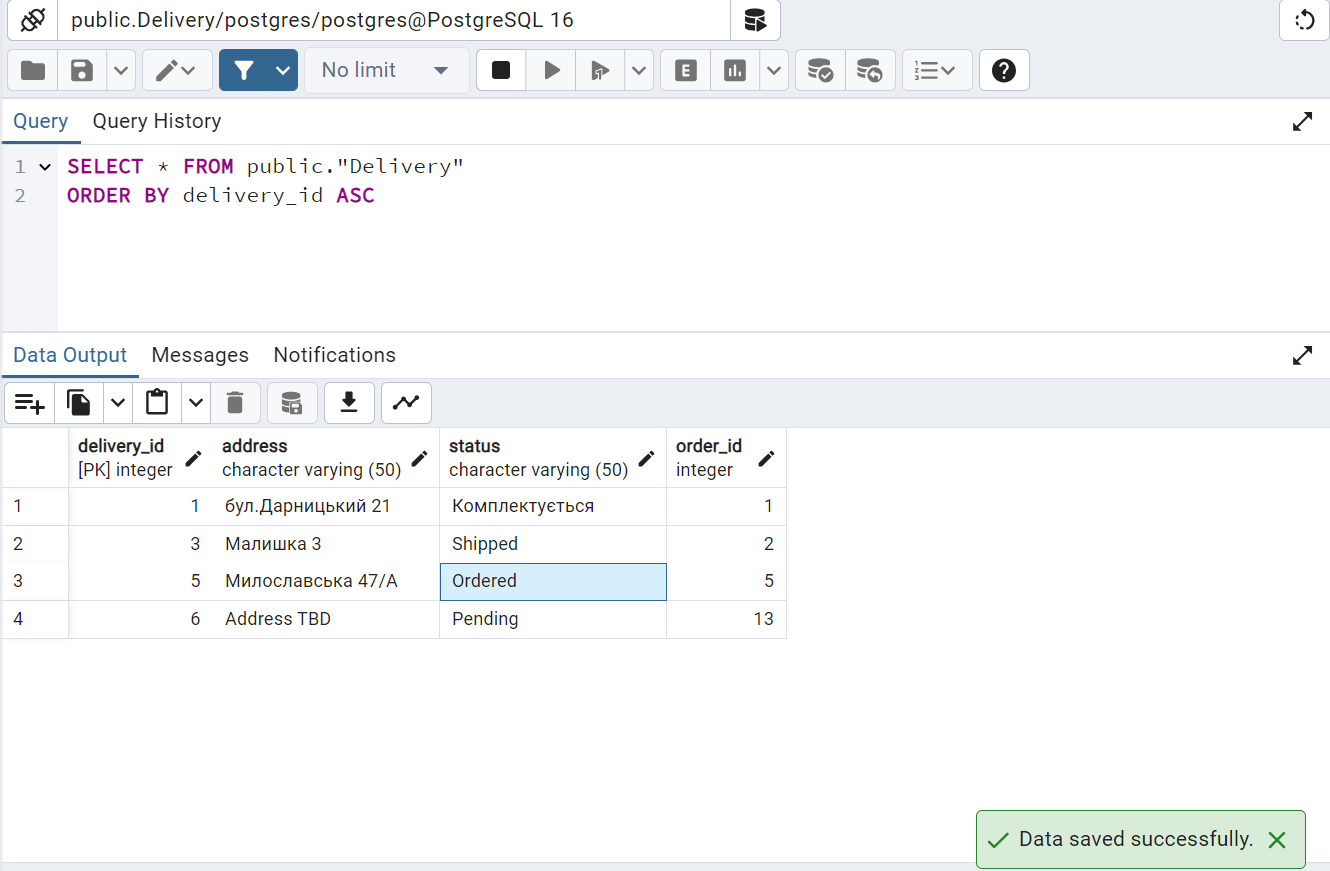
CREATE TRIGGER UpdateTrigger

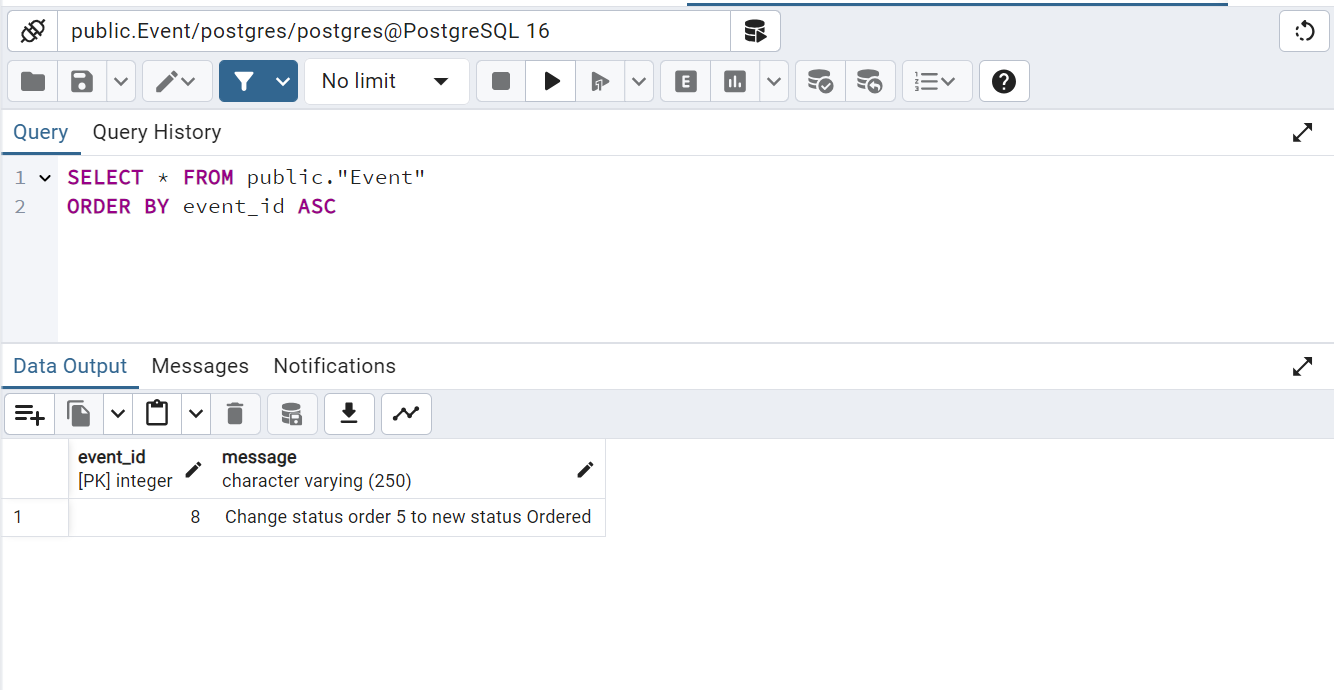
AFTER UPDATE ON "Delivery"

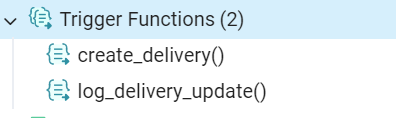
FOR EACH ROW

EXECUTE FUNCTION log\_delivery\_update();

**Тестування роботи тригеру:  
**

****

****



***Тригер з циклом FOR IN:***

Тригер ForIn використовується для управління станом замовлення та оновлення інформації про кількість товарів у таблиці «Product», залежно від змін у статусі доставки status в таблиці «Delivery». Тригер спрацьовує після додаванняAFTER INSERT або оновлення UPDATE OF status запису в таблиці «Delivery». Тригер перевіряє значення нового статусу NEW.status замовлення і виконує дії відповідно до трьох можливих станів:

* *Pending* (в процесі очікування),
* *Shipped* (відправлено),
* *Cancelled* (скасовано).

*Якщо статус Pending:*

Це означає, що замовлення перебуває в процесі підготовки. У цьому випадку:

* + Кількість товарів на складі (quantity) зменшується на кількість товарів, вказаних у замовленні.
  + Зарезервовані товари (reserved) збільшуються на ту ж кількість.

*Якщо статус Shipped:*

Це означає, що замовлення було відправлено клієнту. У цьому випадку:

* + Кількість зарезервованих товарів (reserved) зменшується.
  + Кількість на складі (quantity) не змінюється, оскільки товари вже були зарезервовані під це замовлення.

*Якщо статус Cancelled:*

Це означає, що замовлення було скасовано. У цьому випадку:

* + Резерв (reserved) повертається в кількість на складі (quantity).
  + Резерв (reserved) очищується (стає NULL).

**Код тригеру:**CREATE OR REPLACE FUNCTION manage\_order\_status()

RETURNS TRIGGER AS $$

DECLARE

product\_row RECORD;

BEGIN

IF NEW.status = 'Pending' THEN

FOR product\_row IN

SELECT po.product\_id, po.quantity

FROM "Product\_Order" po

WHERE po.order\_id = NEW.order\_id

LOOP

UPDATE "Product"

SET quantity = quantity - product\_row.quantity,

reserved = COALESCE(reserved, 0) + product\_row.quantity

WHERE product\_id = product\_row.product\_id;

END LOOP;

ELSIF NEW.status = 'Shipped' THEN

FOR product\_row IN

SELECT po.product\_id, po.quantity

FROM "Product\_Order" po

WHERE po.order\_id = NEW.order\_id

LOOP

UPDATE "Product"

SET reserved = reserved - product\_row.quantity

WHERE product\_id = product\_row.product\_id;

END LOOP;

ELSIF NEW.status = 'Cancelled' THEN

FOR product\_row IN

SELECT po.product\_id, po.quantity

FROM "Product\_Order" po

WHERE po.order\_id = NEW.order\_id

LOOP

UPDATE "Product"

SET quantity = quantity + product\_row.quantity,

reserved = NULL

WHERE product\_id = product\_row.product\_id;

END LOOP;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER manage\_order\_status

AFTER INSERT OR UPDATE OF status

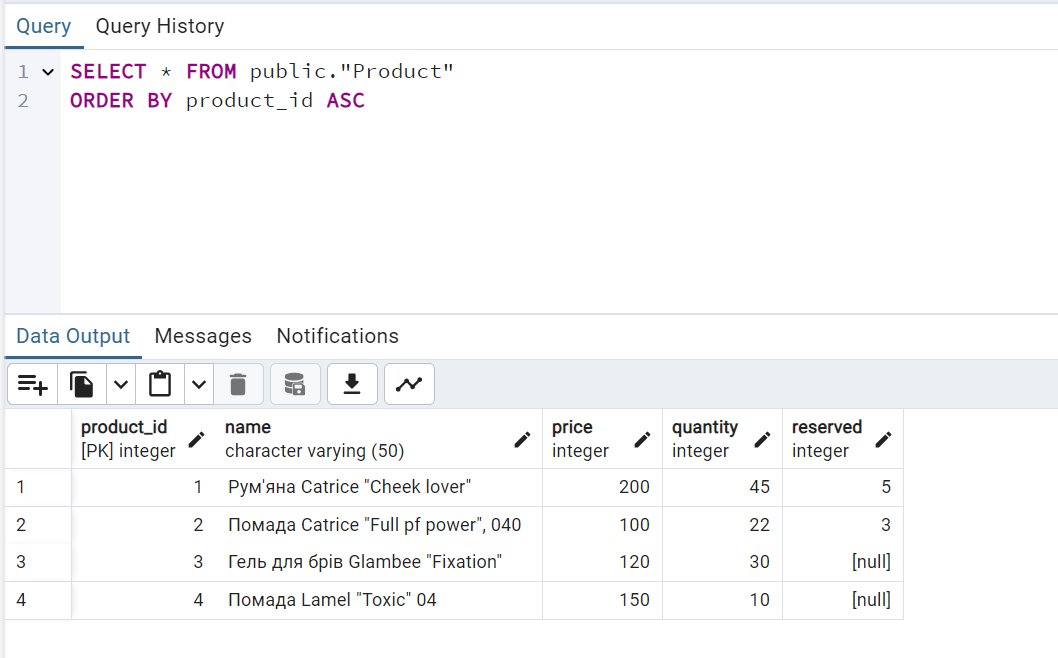
ON "Delivery"

FOR EACH ROW

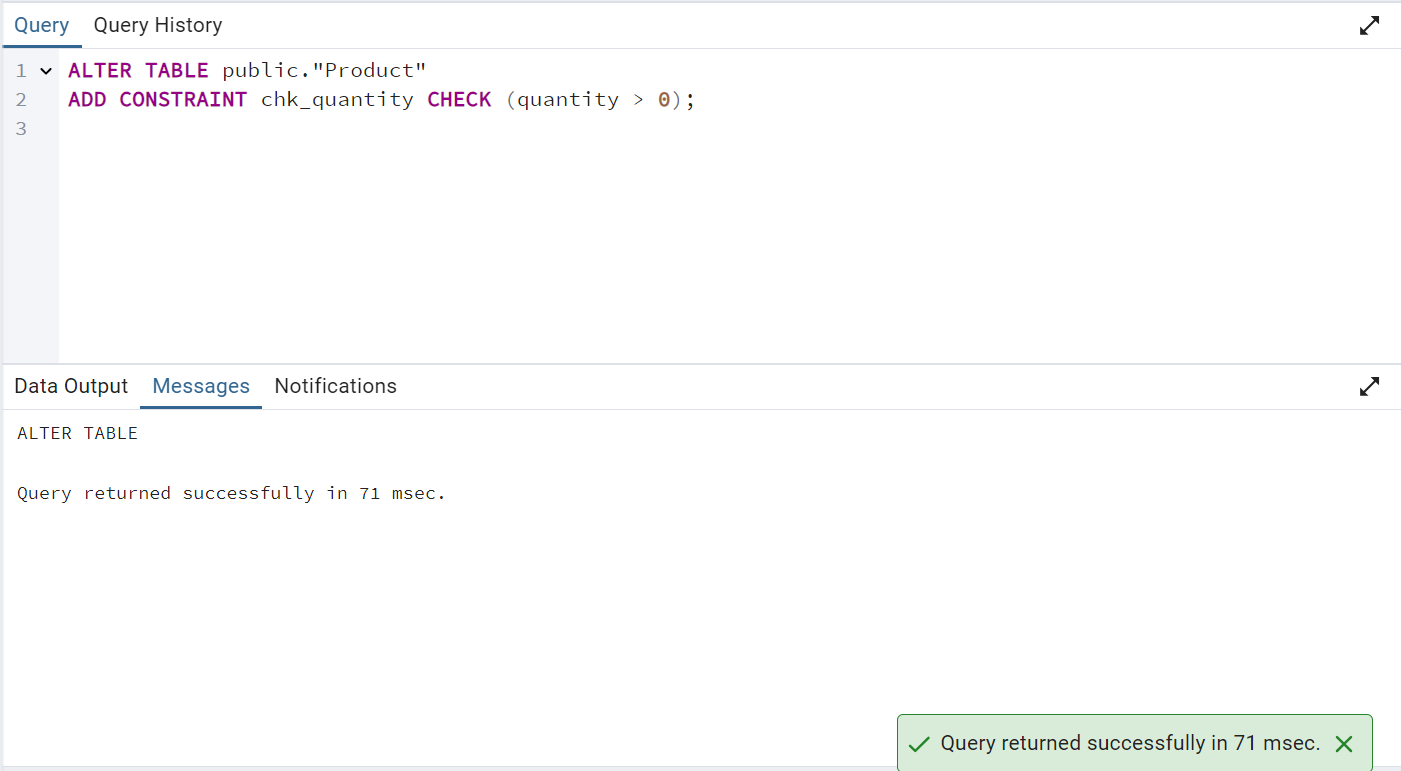
EXECUTE FUNCTION manage\_order\_status();

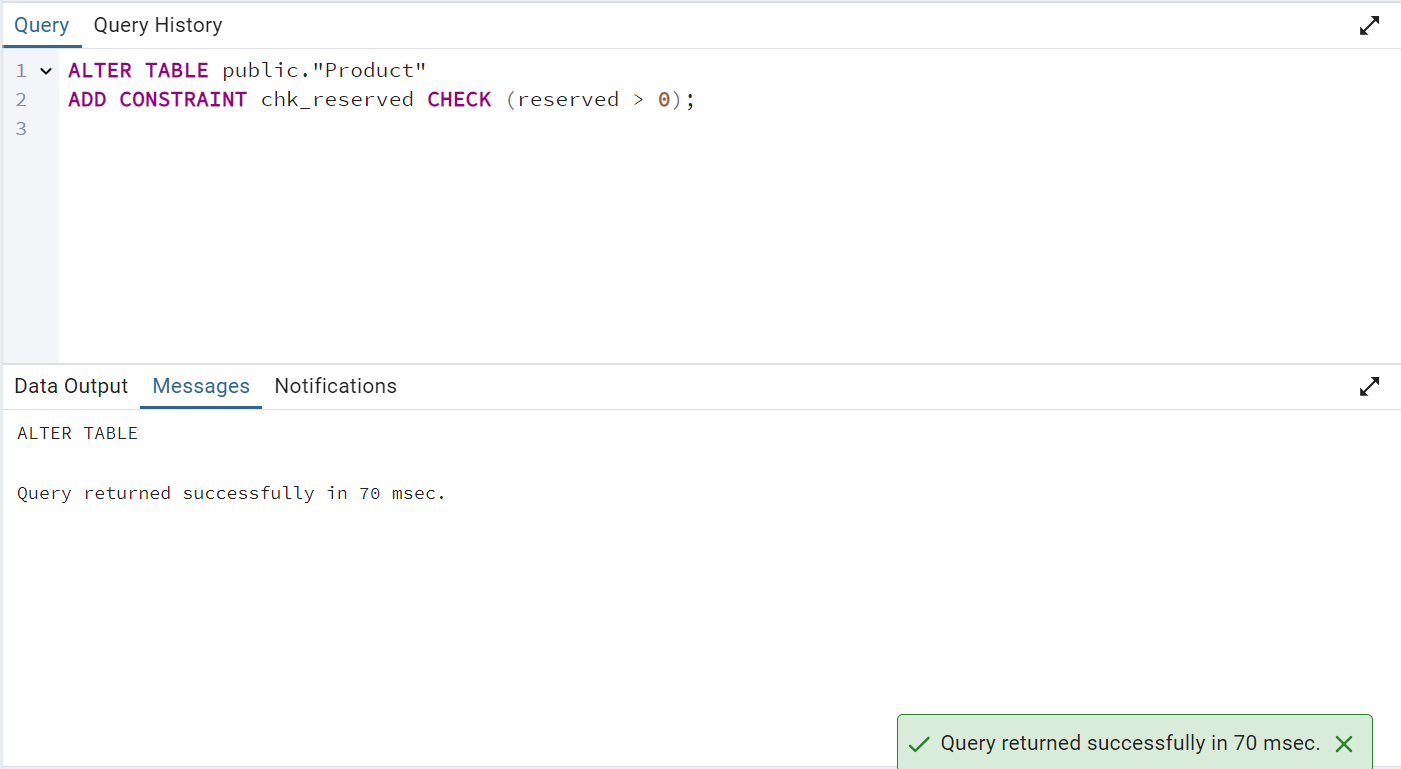
**Таблиця «Product» до виконання тригеру:**

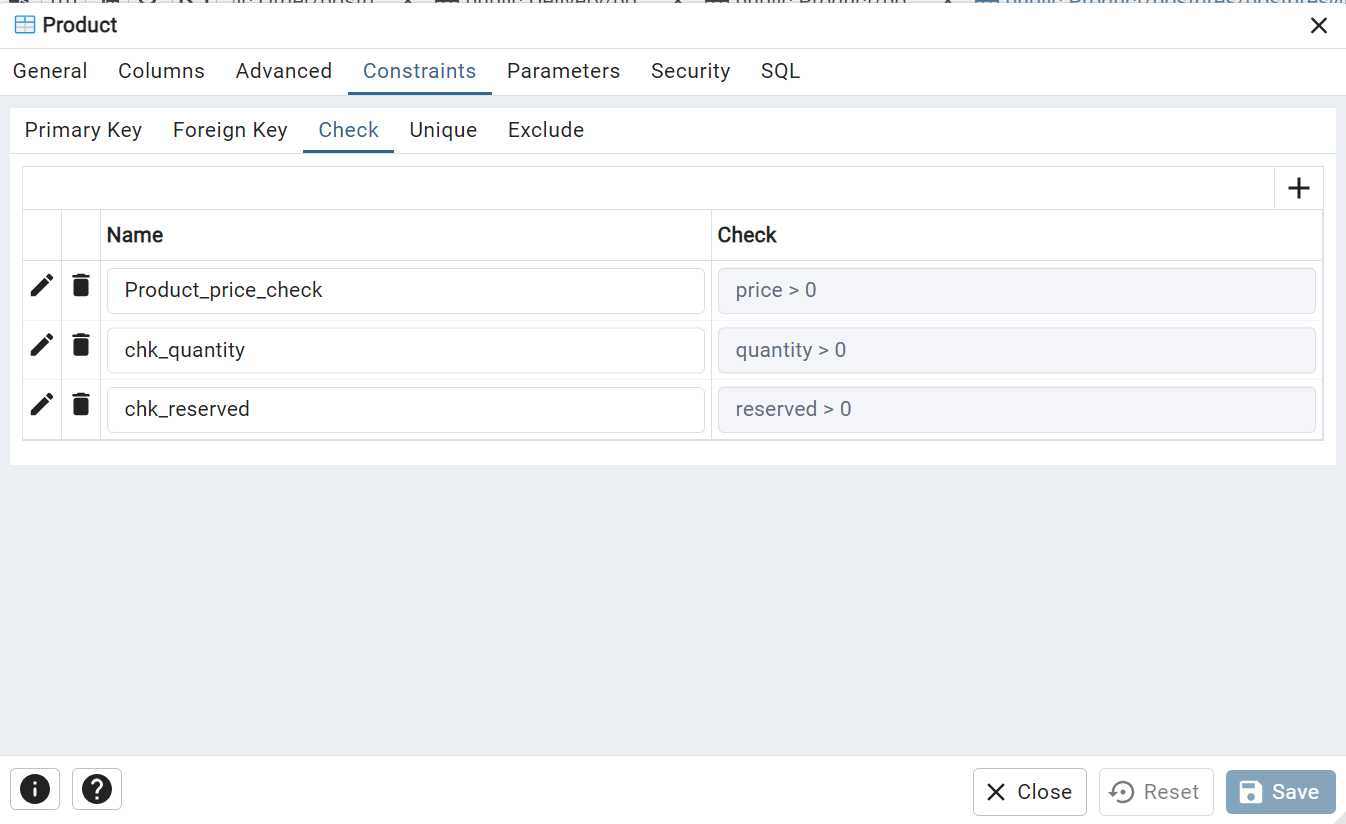
Ми додали стовбці quantity (кількість товарів на складі) та reserved (зерезервовані).

****

Також ми додали умову CHECK, щоб не було від’ємних чисел:

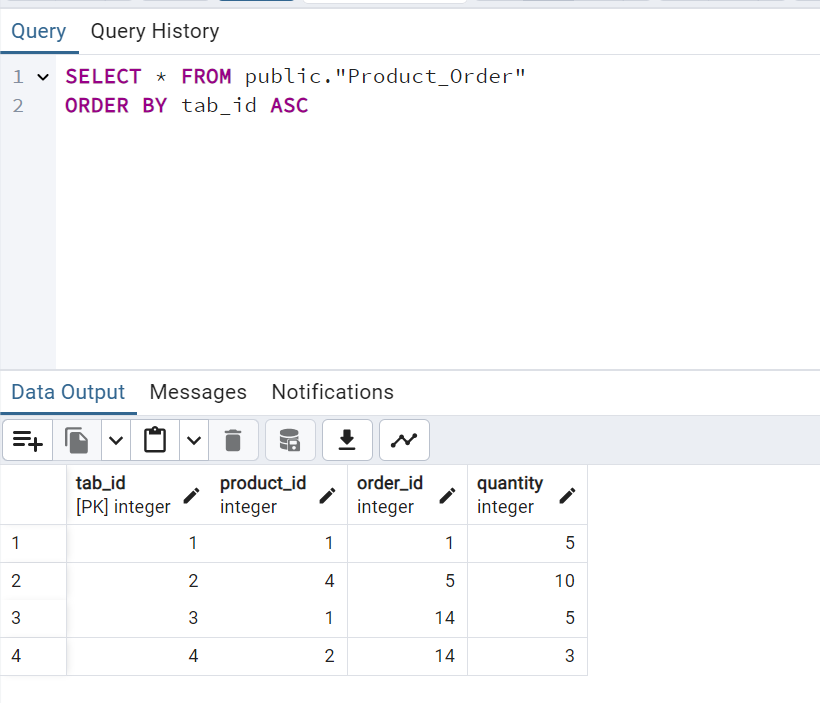
****

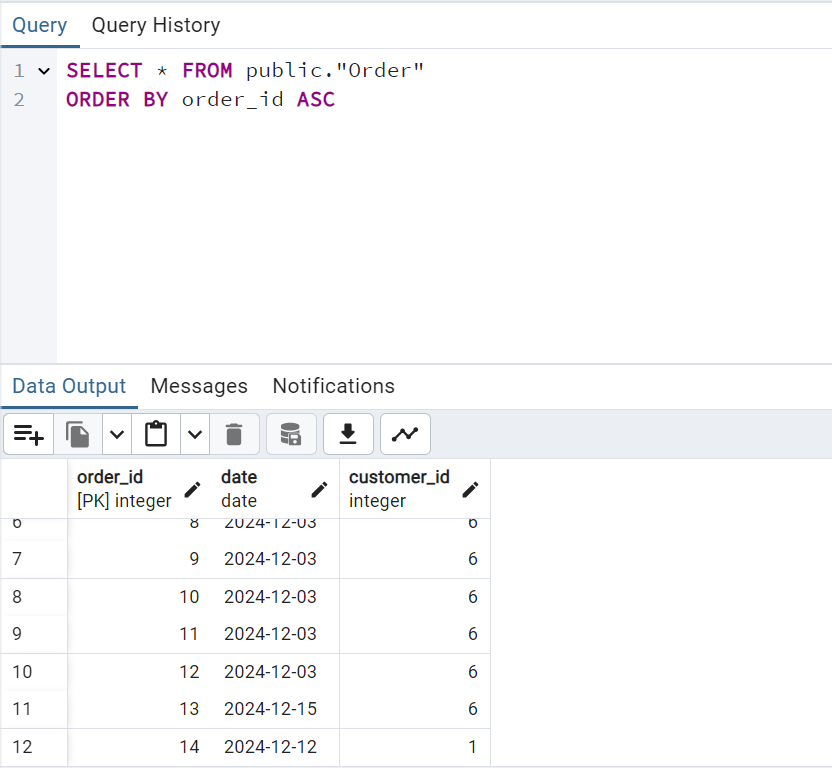
****

****

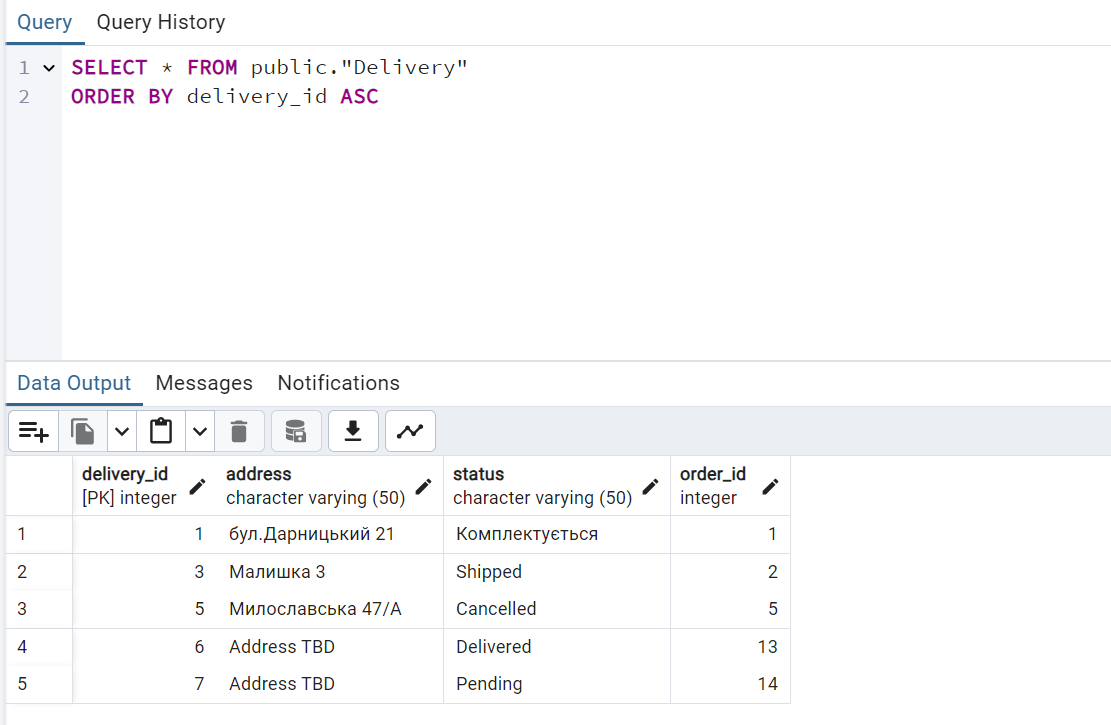
**Таблиця «Product\_Order» до виконання тригеру:**

Ми також додали стовпчик quantity (кількість товарів на складі).

****

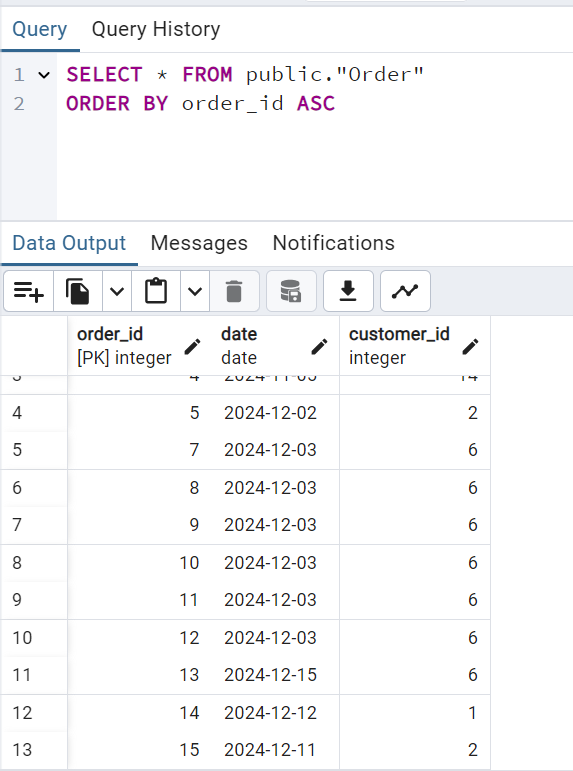
**Таблиця «Order» до виконання тригеру:  
**

**Таблиця «Delivery» до виконання тригеру:**

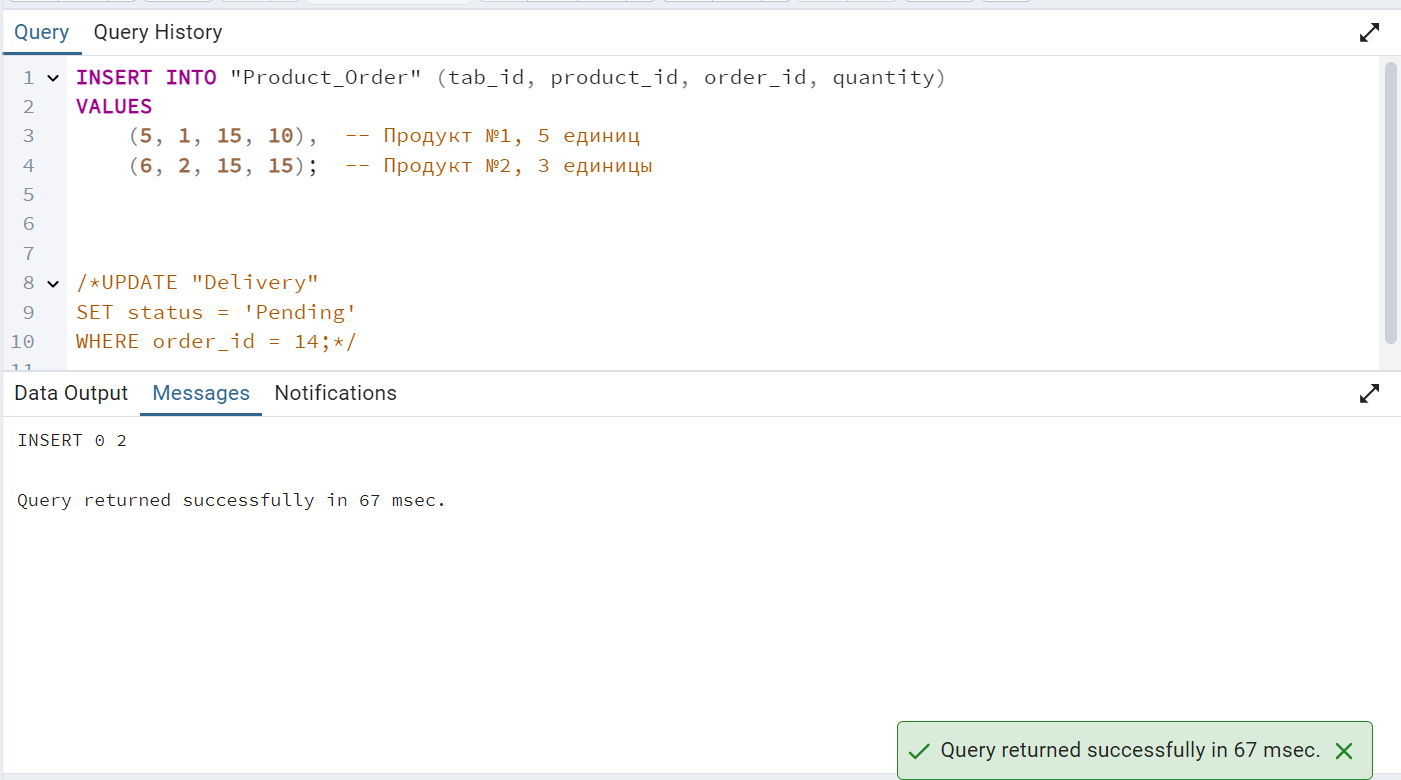
****

**Тестування роботи тригеру:**

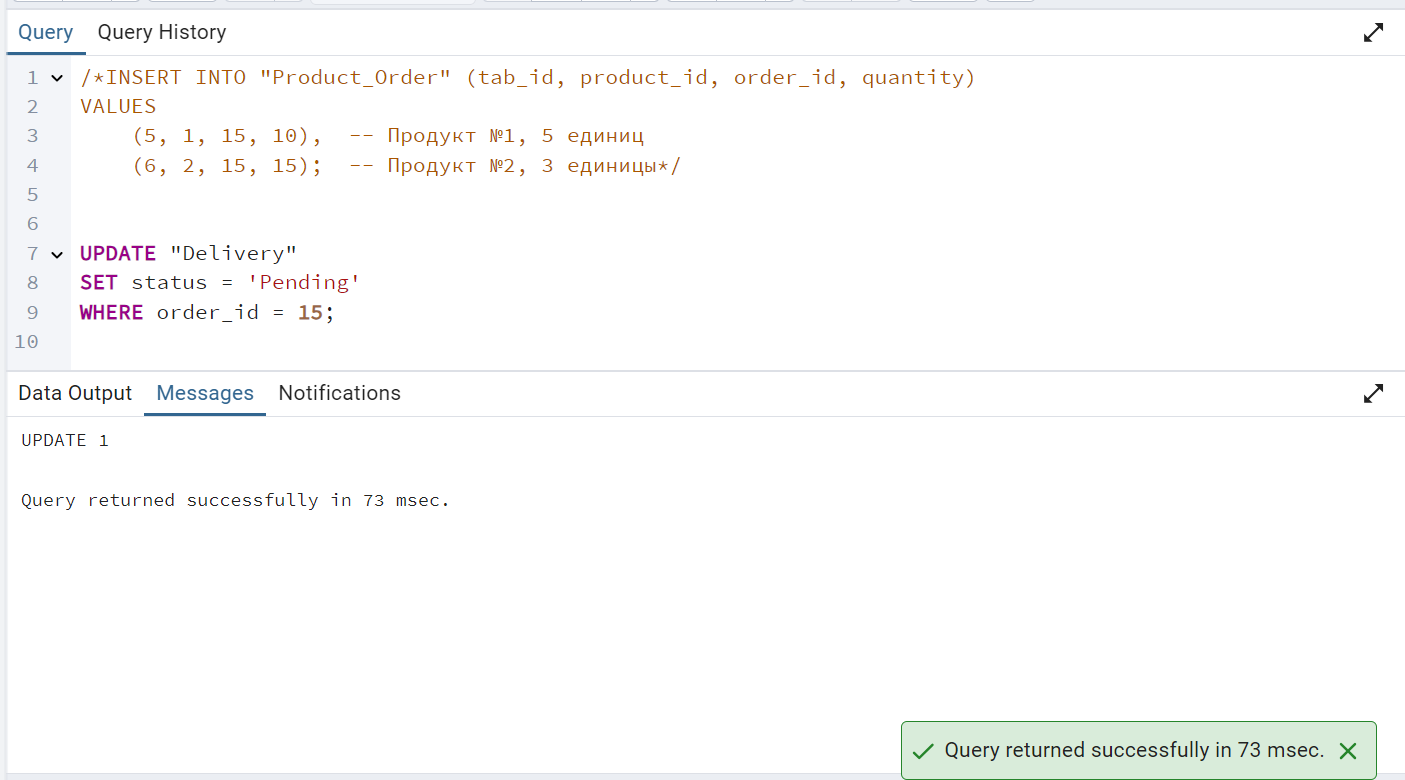
Спочатку додаємо 15 замовлення в таблиці «Order».

****

Процес додавання нових записів до таблиці “Product\_Order».

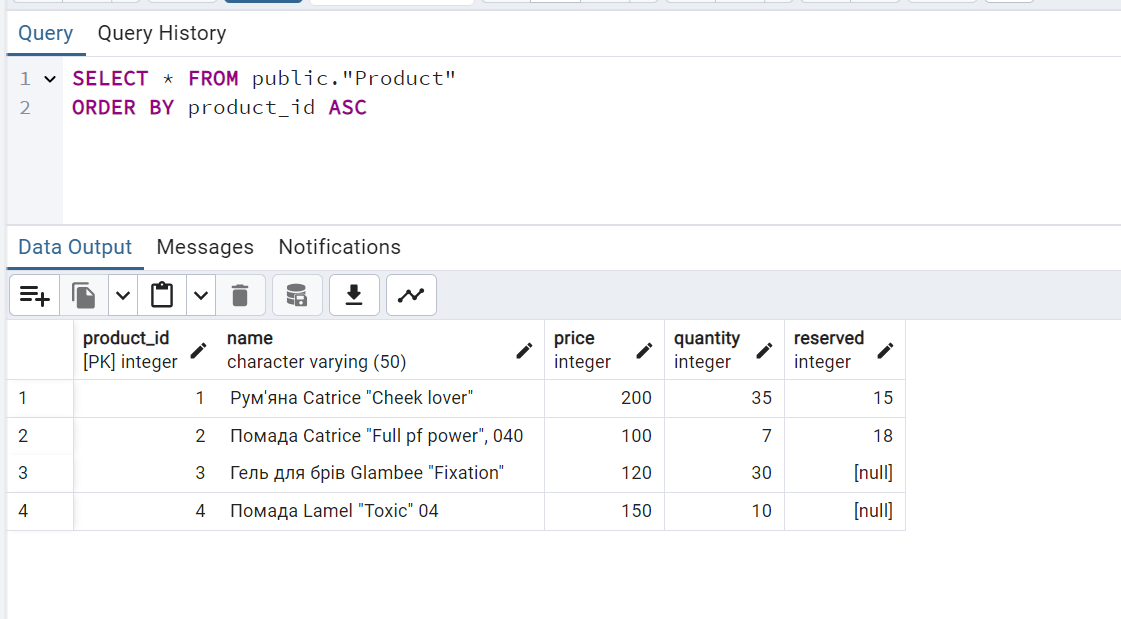
****

Процес оновлення статусу доставки в таблиці «Delivery» для 15 замовлення.

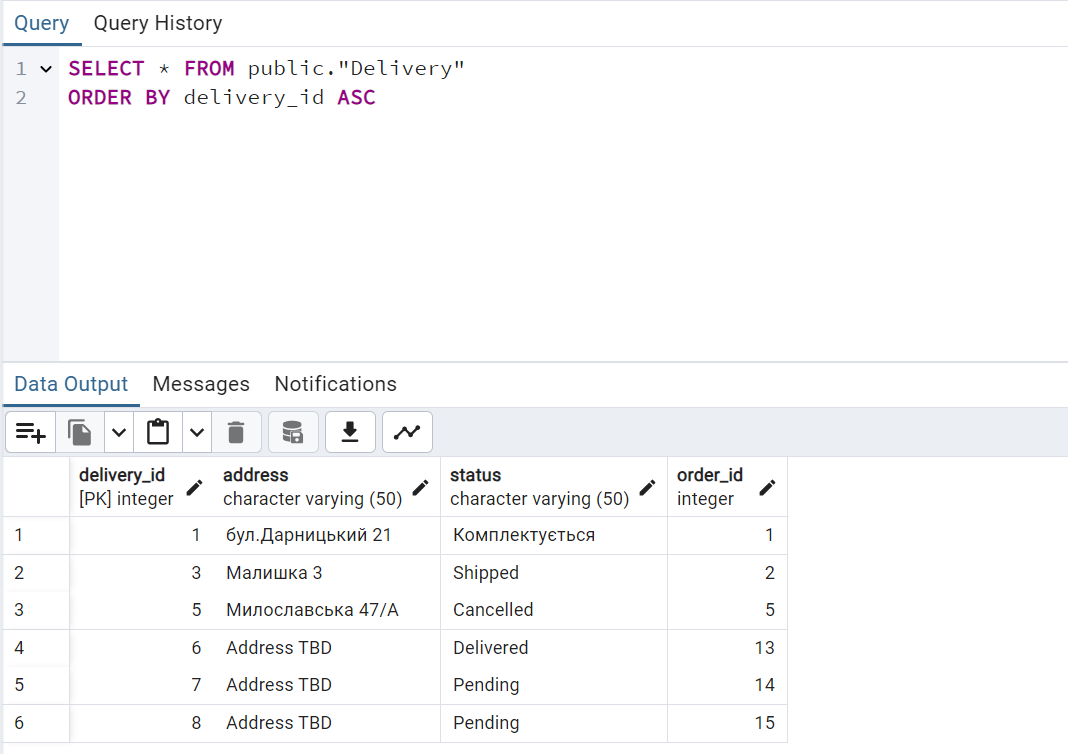
****

**Результат роботи тригеру:**

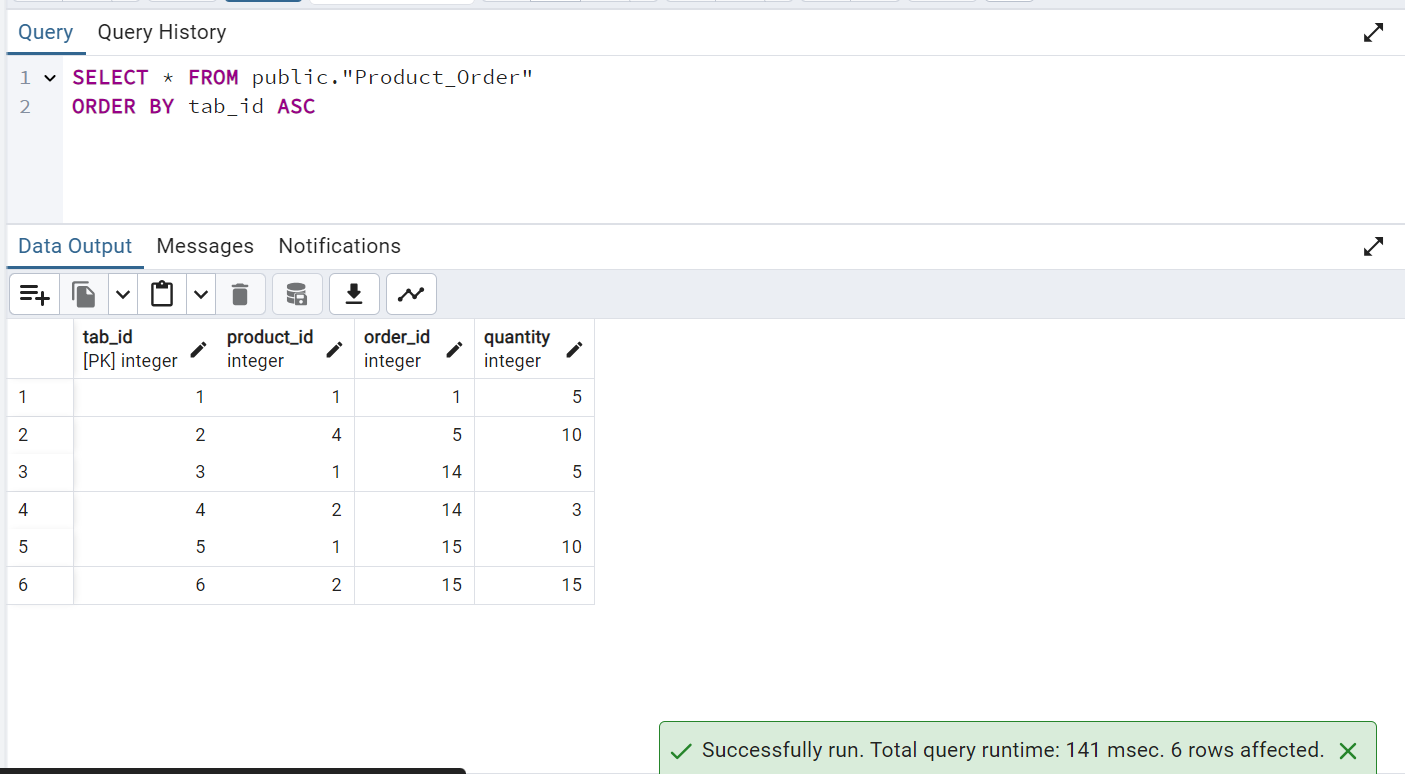
Оскільки статус цього замовлення “Pending”, то ми віднімаємо 10 й 15 з quantity (кількість товарів на складі) й додаємо до reserved (зерезервовані).

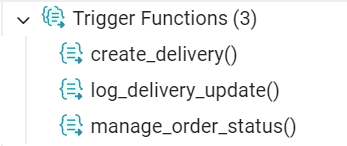
**Таблиця «Product»  
**

**Таблиця «Delivery»**

****

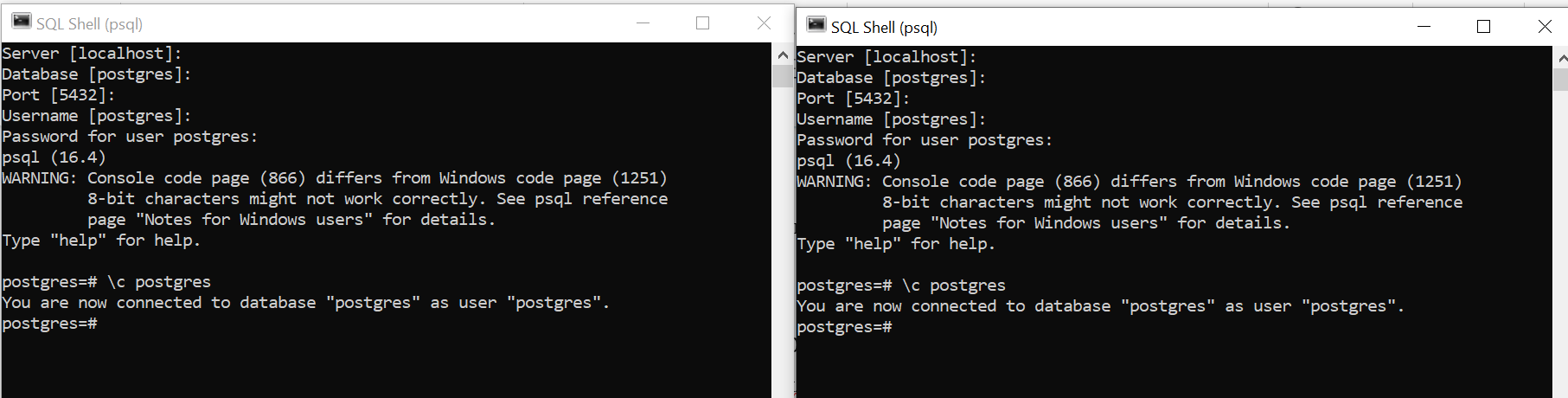
**Таблиця «Product\_Order»**

****

****

**Завдання №4**

Спочатку треба під’єднатись до БД:

****

**READ COMMITED**

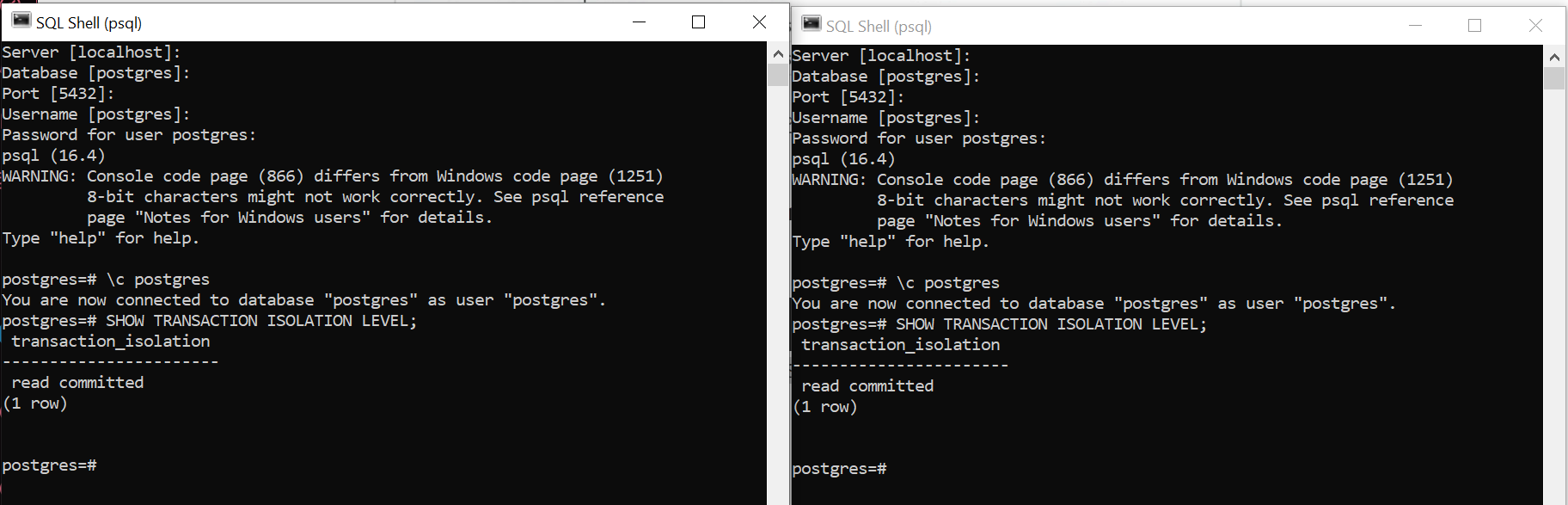
READ COMMITTED - це рівень ізоляції транзакцій в системах управління базами даних (СУБД). Цей рівень забезпечує менший рівень ізоляції порівняно з SERIALIZABLE, але при цьому він є більш продуктивним і менше схильним до блокування ресурсів.

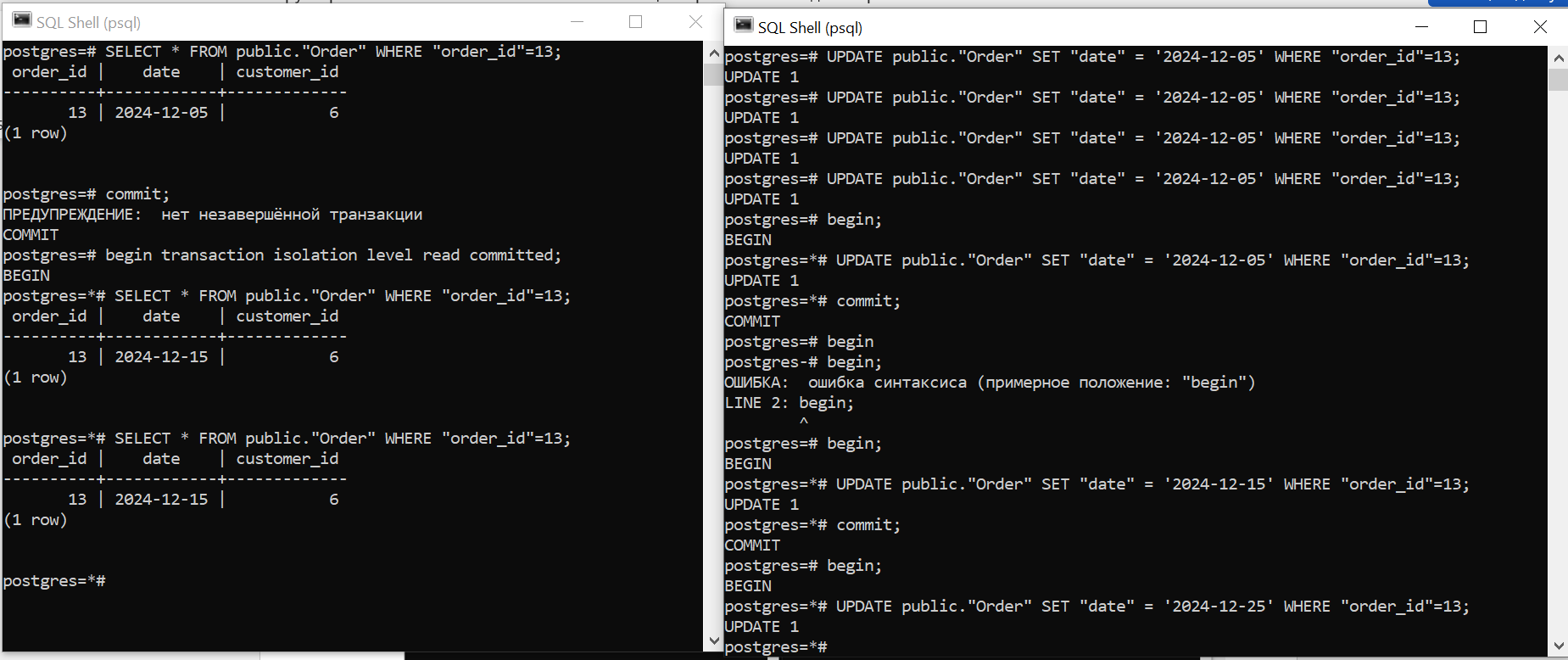
Основні характеристики рівня ізоляції READ COMMITTED:

1. Блокування для запису, але не для читання: Транзакція, яка працює на рівні READ COMMITTED, блокує рядки даних для запису, коли вона їх змінює, але не блокує їх для читання. Це означає, що інші транзакції можуть читати дані, навіть якщо вони ще не були фіксовані (збережені).
2. Недопущення "грязних читань": Транзакція READ COMMITTED не допускає "грязних читань". Це означає, що транзакція не може читати незафіксовані зміни інших транзакцій.
3. Допущення "неповторних читань" і "фантомних вставок": Однак, READ COMMITTED дозволяє "неповторні читання", тобто можливість читання одного і того ж рядка, який був змінений іншою транзакцією після початку поточної транзакції. Також можливі "фантомні вставки" або "фантомні оновлення", тобто вставка або оновлення нових рядків в поточній транзакції.
4. Читання фіксованих змінених даних: Транзакція може читати тільки фіксовані (збережені) зміни інших транзакцій. Якщо транзакція A змінює дані, транзакція B може прочитати тільки фіксовані дані A, а не їхні зміни.

READ COMMITTED є рівнем ізоляції, який часто використовується в багатьох системах, оскільки він надає гарний баланс між ізоляцією і продуктивністю, але слід враховувати його особливості при розробці додатків, особливо у випадках, коли критична стійкість до "грязних читань".

Рівень ізоляції **за замовчуванням** у postgres — це READ COMMITTED. Щоб підтвердити, виконаємо цю команду в psql:

****



**REPEATABLE READ**

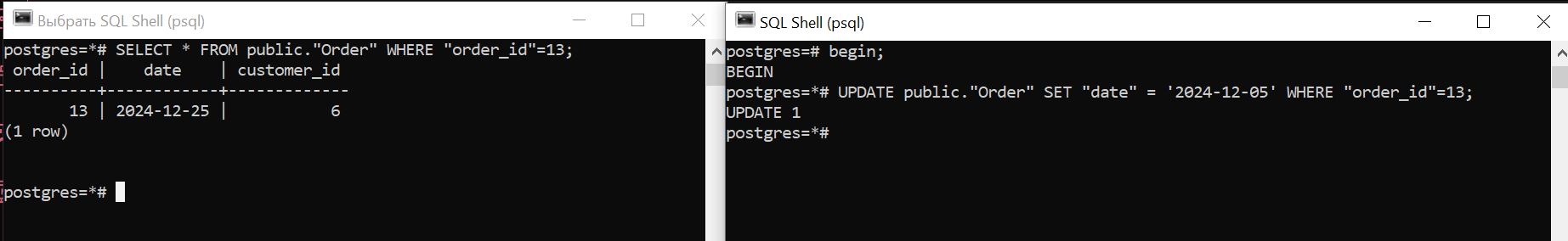
Цей рівень забезпечує високий рівень ізоляції транзакцій, забезпечуючи виключність читань, але водночас дозволяючи "фантомні оновлення".

Основні характеристики рівня ізоляції REPEATABLE READ:

Виділений час життя транзакції: Транзакція на рівні REPEATABLE READ має свій власний виділений час життя, і весь час цієї транзакції транзакційний простір даних залишається консистентним.

1. Блокування для читання і запису: Транзакція, яка працює на рівні REPEATABLE READ, блокує рядки даних, які вона читає, а також ті, що задіяні у виразі WHERE у SELECT-запитах. Це має на меті уникнення "грязних читань", "неповторних читань" і "фантомних вставок".
2. Запобігання неповторним читанням: Транзакція на рівні REPEATABLE READ гарантує, що якщо вона читає дані певного рядка, цей рядок залишатиметься незмінним протягом усього часу життя транзакції, навіть якщо інші транзакції модифікують цей рядок.
3. Фантомні оновлення: Однак REPEATABLE READ не гарантує відсутності "фантомних оновлень". Іншими словами, хоча транзакція блокує рядки, які вона читає, нові рядки можуть бути додані в результат виконання SELECT-запиту протягом транзакції. Це може призвести до того, що одна і та ж сама транзакція при повторному виконанні одного і того ж самого запиту побачить нові рядки.

REPEATABLE READ є компромісом між високим рівнем ізоляції та продуктивністю, і він підходить для багатьох випадків використання, де важлива стійкість до "грязних читань" і "неповторних читань", але можливі "фантомні оновлення".



**SERIALIZABLE**

SERIALIZABLE - це найвищий рівень ізоляції транзакцій в базах даних. Цей рівень забезпечує повну ізоляцію транзакцій, що означає, що жодна транзакція не може бачити змін, внесених іншими транзакціями, поки вони не будуть закінчені. Рівень ізоляції SERIALIZABLE запобігає всім видам аномалій, таких як "грязне читання", "неповторне читання", "фантомні читання", "грязні оновлення" і "фантомні оновлення".

Основні характеристики рівня ізоляції SERIALIZABLE:

1. Повна блокада для читання і запису: Транзакція, яка працює на рівні SERIALIZABLE, блокує всі рядки даних, з якими вона взаємодіє, для читання і запису. Це забезпечує повну ізоляцію від інших транзакцій і уникнення всіх видів конфліктів.
2. Запобігання фантомним читанням і вставкам: SERIALIZABLE гарантує, що транзакції не побачать нових рядків, які були вставлені іншими транзакціями, і не зможуть вставити нові рядки в той самий набір даних після читання. Це включає в себе запобігання "фантомним читанням" і "фантомним вставкам".
3. Недопущення фантомних оновлень: SERIALIZABLE також запобігає "фантомним оновленням", гарантуючи, що транзакції не зможуть оновлювати дані, які були змінені іншими транзакціями після початкового читання.

Стабільність порядку читання: Цей рівень також гарантує стабільність порядку читання, тобто результати SELECT-запитів залишаються незмінними в межах транзакції.

Хоча SERIALIZABLE надає максимальний рівень ізоляції, він може також призвести до значного блокування ресурсів і впливати на продуктивність системи. Тому обирайте його тільки в тих випадках, коли висока ізоляція є критичною, і ви готові пожертвувати частиною продуктивності.

