GUAP

DEPARTMENT #43

REPORT   
PROTECTED WITH ASSESSMENT

TEACHER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Senior Lecturer |  |  |  | N. V. Putilova​​ |
| position, academic degree, title |  | signature, date |  | initials, surname |

|  |
| --- |
| LABORATORY WORK REPORT #8 |
| Triggers: Ensuring Active Database Data Integrity |
| in the course: DATABASE DESIGN |
|  |

THE WORK IS COMPLETED

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STUDENT GROUP # | 4236 |  |  |  | A.V. Panchenko |
|  |  |  | signature, date |  | initials, surname |

Saint Petersburg 202 4

**Objective of the work:** To acquire skills and abilities in designing and creating database triggers, including their use to maintain active referential integrity.

**Description of the task**

Implement triggers for your database for all events ( insert, delete , update ) before and after. (6 triggers) Some of which will ensure referential integrity, the rest may have another purpose from the other proposed ones, but not less than 2 different ones. - Calculation/maintenance of computable (derived) attributes in an up-to-date state - logging (recording) of changes; - checking the correctness of the actions performed.). Computable fields can be added if necessary.

**Option 18:**

18. Delivery service: delivery address, contact person, parcel value, desired delivery time range, actual delivery time, parcel weight, delivery mark, sender company

a. all parcels sent to the area, the name of which contains the beginning " mosk ", but these are not the only letters in it

b. Street where there are no delivery addresses

c. contact person who received parcels on Stroiteley Street and Lenin Avenue

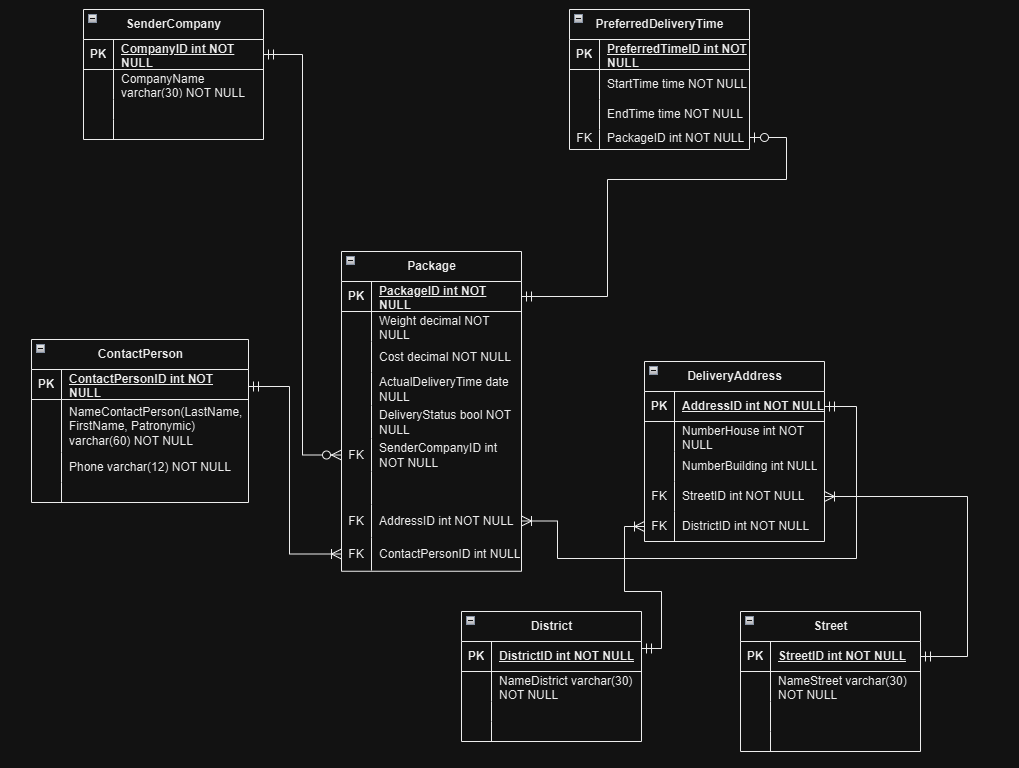
g. the company that sent more than the average number of parcels

d. parcel with weight less than average

e. contact person who received parcels from all companies starting with the letter B

f. a contact person who never received parcels last year, but received parcels in February of the current year

**Physical model of the database :**



**Purpose of developed triggers by text :**

1. **Trigger check\_cost\_before\_insert**

* **Type** : BEFORE INSERT
* **Purpose** : This trigger checks the value of the Cost field before inserting a new record into the Package table . If the package cost is negative, the trigger raises an error and prevents the insert. This ensures data integrity and prevents incorrect values from being added to the database.

2. **Trigger check\_cost\_before\_update**

* **Type** : BEFORE UPDATE
* **Purpose** : Similar to the insert trigger, this trigger checks the value of the Cost field before updating an existing record. If the new cost of the package is negative, the trigger raises an error and prevents the record from being updated. This also helps maintain data integrity.

3. **Trigger update\_status\_before\_insert**

* **Type** : BEFORE INSERT
* **Purpose** : This trigger automatically updates the DeliveryStatus field before inserting a new record into the Package table . If the ActualDeliveryTime field is not NULL, the delivery status is set to TRUE (delivered); if it is NULL, the status is set to FALSE (in transit). This allows you to automatically maintain an up-to-date delivery status without the need for manual entry.

4. **Trigger update\_status\_before\_update**

* **Type** : BEFORE UPDATE
* **Purpose** : This trigger performs the same function as the Insert trigger, but is applied when updating existing records. It updates the DeliveryStatus field based on the value of the ActualDeliveryTime field , which keeps the delivery status up to date even after other package attributes have changed.

5. **Trigger log\_after\_insert**

* **Type** : AFTER INSERT
* **Purpose** : This trigger records information about each insertion of a new record into the package\_log table . It records the package ID, the action (INSERT) and the time of the change. This allows you to track all changes in the database and provides the ability to audit user actions.

6. **Log\_after\_update trigger**

* **Type** : AFTER UPDATE
* **Purpose** : Similar to the insert trigger, this trigger records information about each update to an existing record in the package\_log table . It records the package ID, the action (UPDATE), and the time of the change. It also facilitates auditing of changes and allows tracking of the change history.

7. **Log\_after\_delete trigger**

* **Type** : AFTER DELETE
* **Purpose** : This trigger records information about each deletion of a record from the Package table . It records the ID of the deleted package, the action (DELETE), and the modification time in the log table. This allows you to keep a history of deleted records and ensures that you can restore the information if necessary.

**8. Trigger before\_package\_delete**

* **Type** : BEFORE DELETE
* **Purpose** : This trigger fires before a record is deleted from the Package table . It ensures that before a record is deleted from the Package , all related records from the PreferredDeliveryTime table that reference the record being deleted are deleted . This ensures referential integrity and prevents dangling references in the PreferredDeliveryTime table if a package is deleted.

**Script for creating triggers :**

-- Function to check the value before inserting

CREATE OR REPLACE FUNCTION check\_cost\_before\_ insert ( )

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Cost < 0 THEN

RAISE EXCEPTION 'Cost cannot be negative';

END IF;

RETURN NEW;

END;

$$

LANGUAGE plpgsql ;

-- Trigger to check the value before inserting

CREATE TRIGGER check\_cost\_before\_insert

BEFORE INSERT ON Package

FOR EACH ROW

EXECUTE FUNCTION check\_cost\_before\_insert();

-- Функция для проверки стоимости перед обновлением

CREATE OR REPLACE FUNCTION check\_cost\_before\_update()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Cost < 0 THEN

RAISE EXCEPTION 'Cost cannot be negative';

END IF;

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

-- Trigger to check the cost before updating

CREATE TRIGGER check\_cost\_before\_update

BEFORE UPDATE ON Package

FOR EACH ROW

EXECUTE FUNCTION check\_cost\_before\_ update ( );

-- Function to update delivery status

CREATE OR REPLACE FUNCTION update\_delivery\_ status ( )

RETURNS TRIGGER AS $$

BEGIN

IF NEW.ActualDeliveryTime IS NOT NULL THEN

NEW.DeliveryStatus : = TRUE; -- Plastic bag delivered

ELSE

NEW.DeliveryStatus : = FALSE; -- Plastic bag V paths

END IF;

RETURN NEW;

END;

$$

LANGUAGE plpgsql ;

OWNER TO andrejpancenko ;

-- Pre-insert trigger to update delivery status

CREATE TRIGGER update\_status\_before\_insert

BEFORE INSERT ON Package

FOR EACH ROW

EXECUTE FUNCTION update\_delivery\_status ();

-- Pre-update trigger to update delivery status

CREATE TRIGGER update\_status\_before\_update

BEFORE UPDATE ON Package

FOR EACH ROW

EXECUTE FUNCTION update\_delivery\_status ();

-- Function for logging changes

CREATE OR REPLACE FUNCTION log\_package\_ changes ( )

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO package\_log (PackageID, Action, ChangeTime)

VALUES (NEW.PackageID, 'INSERT', CURRENT\_TIMESTAMP);

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO package\_log (PackageID, Action, ChangeTime)

VALUES (NEW.PackageID, 'UPDATE', CURRENT\_TIMESTAMP);

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO package\_log (PackageID, Action, ChangeTime)

VALUES (OLD.PackageID, 'DELETE', CURRENT\_TIMESTAMP);

END IF;

RETURN OLD; -- Для DELETE возвращаем OLD

END;

$$

LANGUAGE plpgsql;

-- Создает таблицу package\_log

CREATE TABLE IF NOT EXISTS public.package\_log

(

logid integer NOT NULL DEFAULT nextval('package\_log\_logid\_seq'::regclass),

packageid integer NOT NULL,

action character varying(10) COLLATE pg\_catalog."default" NOT NULL,

changetime timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT package\_log\_pkey PRIMARY KEY (logid)

)

TABLESPACE pg\_default;

-- Installs owner tables on andrejpancenko

ALTER TABLE IF EXISTS public.package \_log

-- Trigger For logging after inserts

CREATE TRIGGER log\_after\_insert

AFTER INSERT ON Package

FOR EACH ROW

EXECUTE FUNCTION log\_package\_changes ( ) ;

-- Trigger For logging after updates

CREATE TRIGGER log\_after\_update

AFTER UPDATE ON Package

FOR EACH ROW

EXECUTE FUNCTION log\_package\_changes ( ) ;

-- Trigger For logging after removals

CREATE TRIGGER log\_after\_delete

AFTER DELETE ON Package

FOR EACH ROW

EXECUTE FUNCTION log\_package\_changes ( ) ;

-- Function to remove related entries in PreferredDeliveryTime before removing from Package

CREATE OR REPLACE FUNCTION delete\_preferred\_time\_before\_package\_ delete ( )

RETURNS TRIGGER AS $$

BEGIN

-- Deletes All records from PreferredDeliveryTime associated with the deleted package

DELETE FROM PreferredDeliveryTime WHERE PackageID = OLD.PackageID ;

RETURN OLD;

END;

$$

LANGUAGE plpgsql ;

-- Trigger that causes removal from PreferredDeliveryTime before removal from Package

CREATE TRIGGER before\_package\_delete

BEFORE DELETE ON Package

FOR EACH ROW

EXECUTE FUNCTION delete\_preferred\_time\_before\_package\_ delete ( );

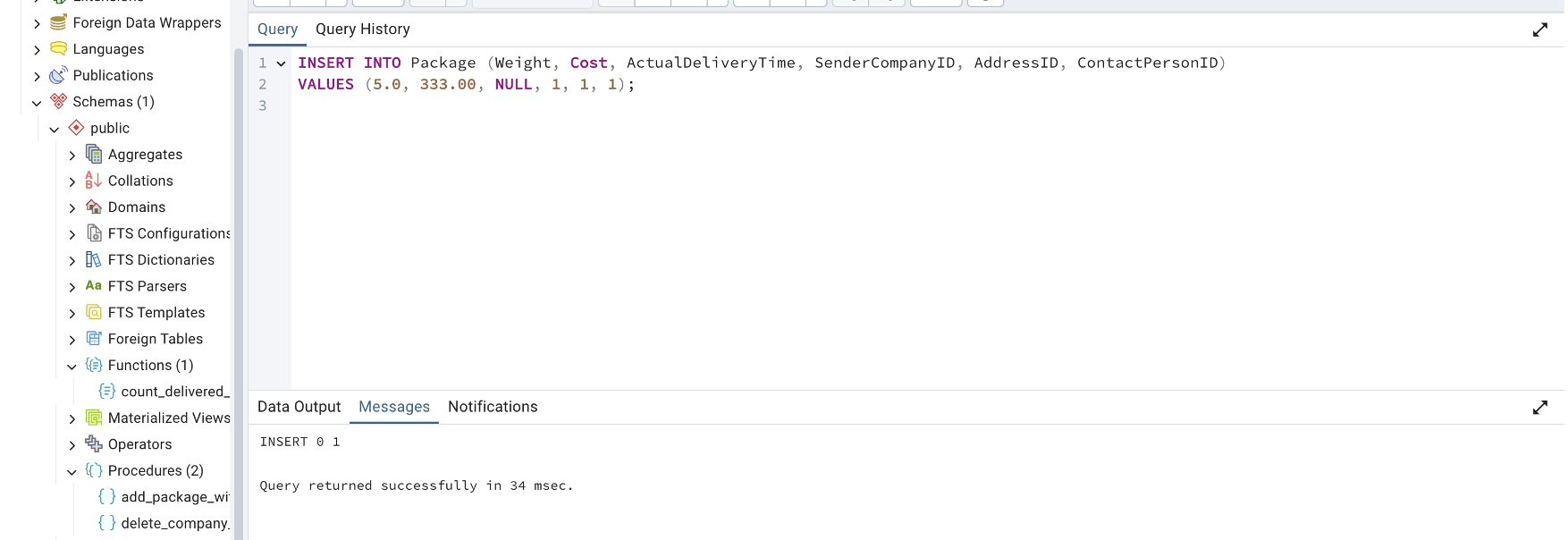
**SQL statements and screenshots of data sets illustrating the operation of triggers**

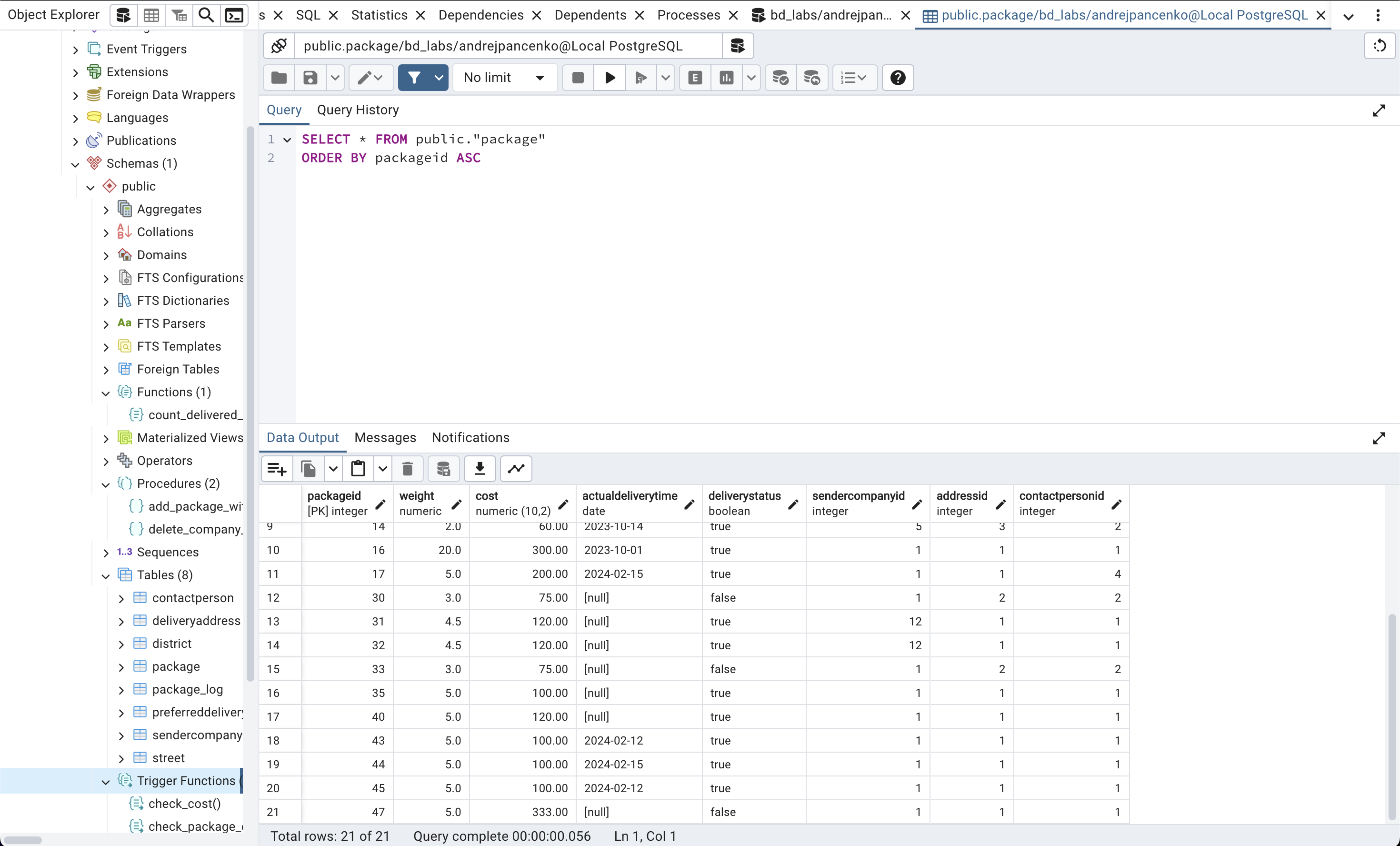
1. Trigger: Check cost before insert ( check\_cost\_before\_insert )

Insert with correct value (should be successful):

INSERT INTO Package (Weight, Cost, ActualDeliveryTime , SenderCompanyID , AddressID , ContactPersonID )

VALUES (5.0, 100.00, NULL, 1, 1, 1);





Insert with invalid value (should cause an error):

INSERT INTO Package (Weight, Cost, ActualDeliveryTime , SenderCompanyID , AddressID , ContactPersonID )

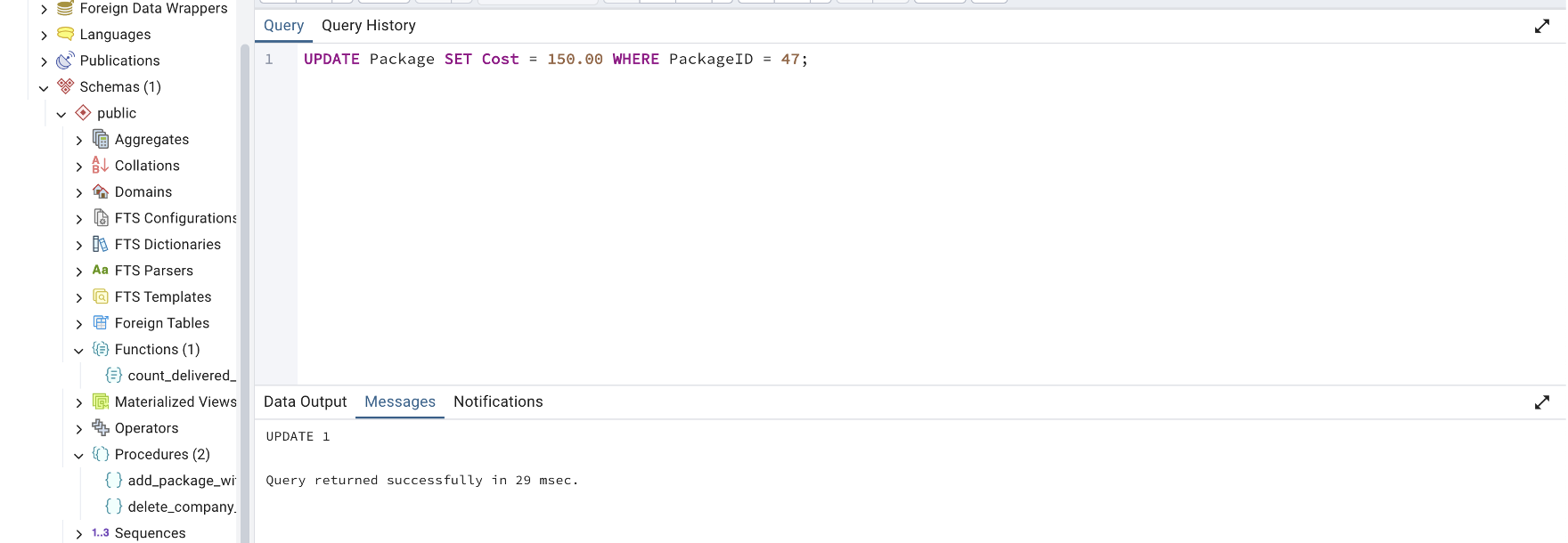
VALUES (5.0, -50.00, NULL, 1, 1, 1); -- This is will cause mistake because of​ trigger check\_cost\_before\_insert .



2. Trigger: Check cost before update ( check \_ cost \_ before \_ update )

Update with correct cost (should be successful):

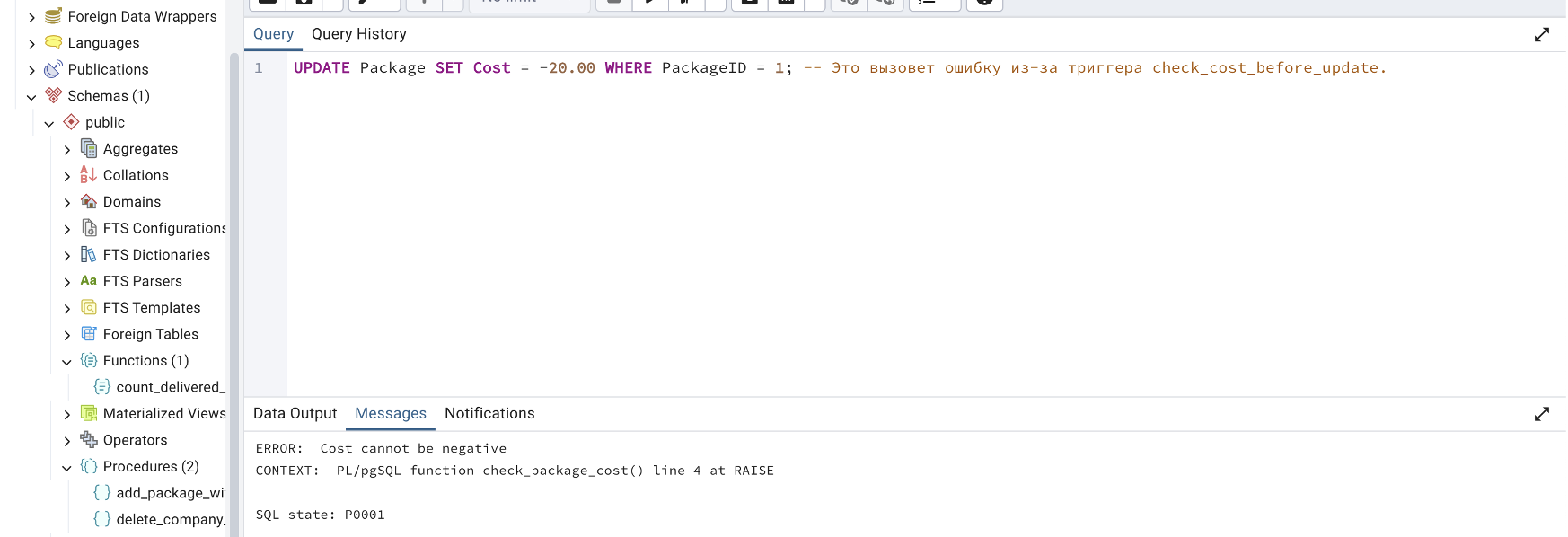
UPDATE Package SET Cost = 150.00 WHERE PackageID = 47;





Update with incorrect value (should cause error):

UPDATE Package SET Cost = -20.00 WHERE PackageID = 1; -- This will cause mistake because of​ trigger check\_cost\_before\_update .

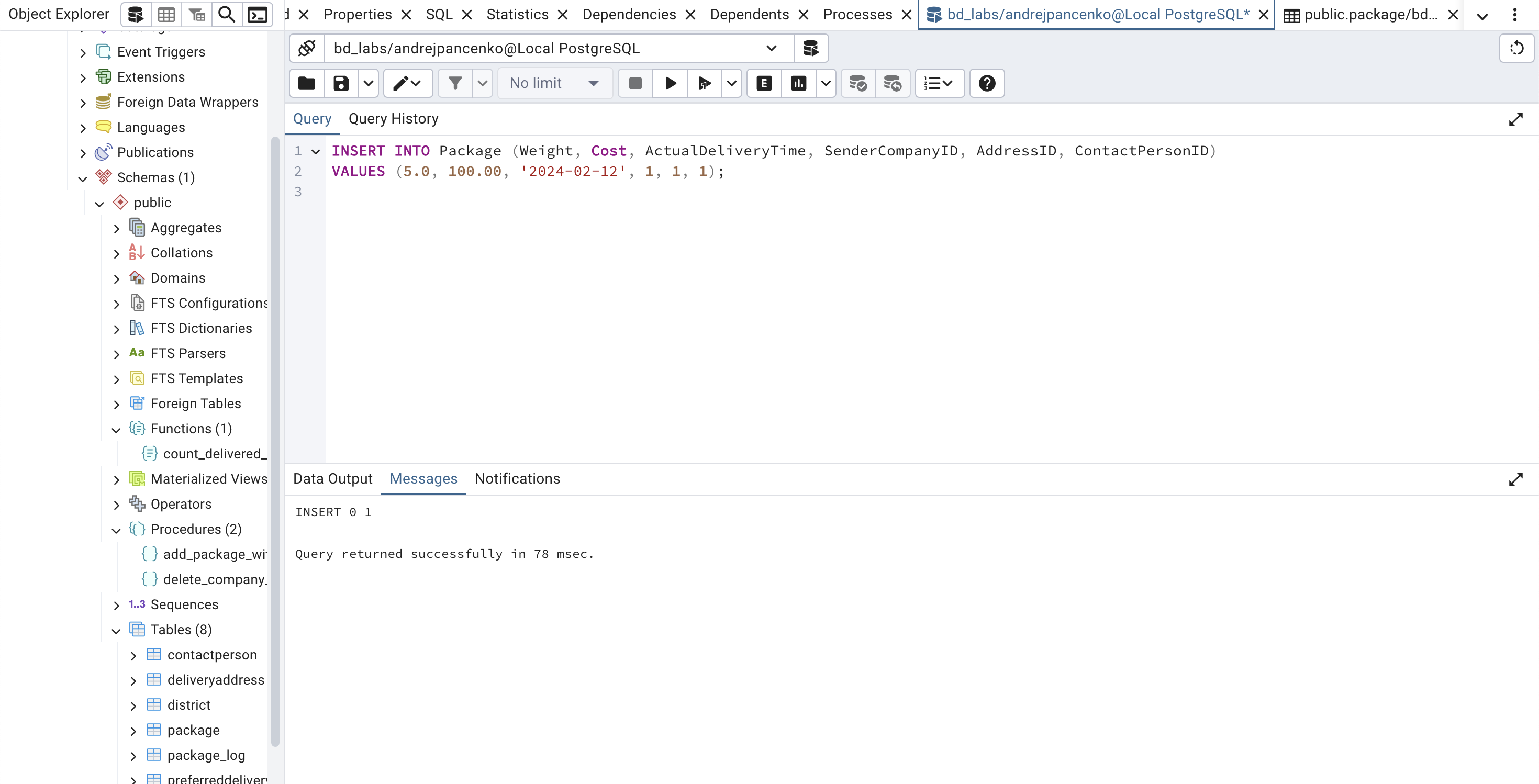


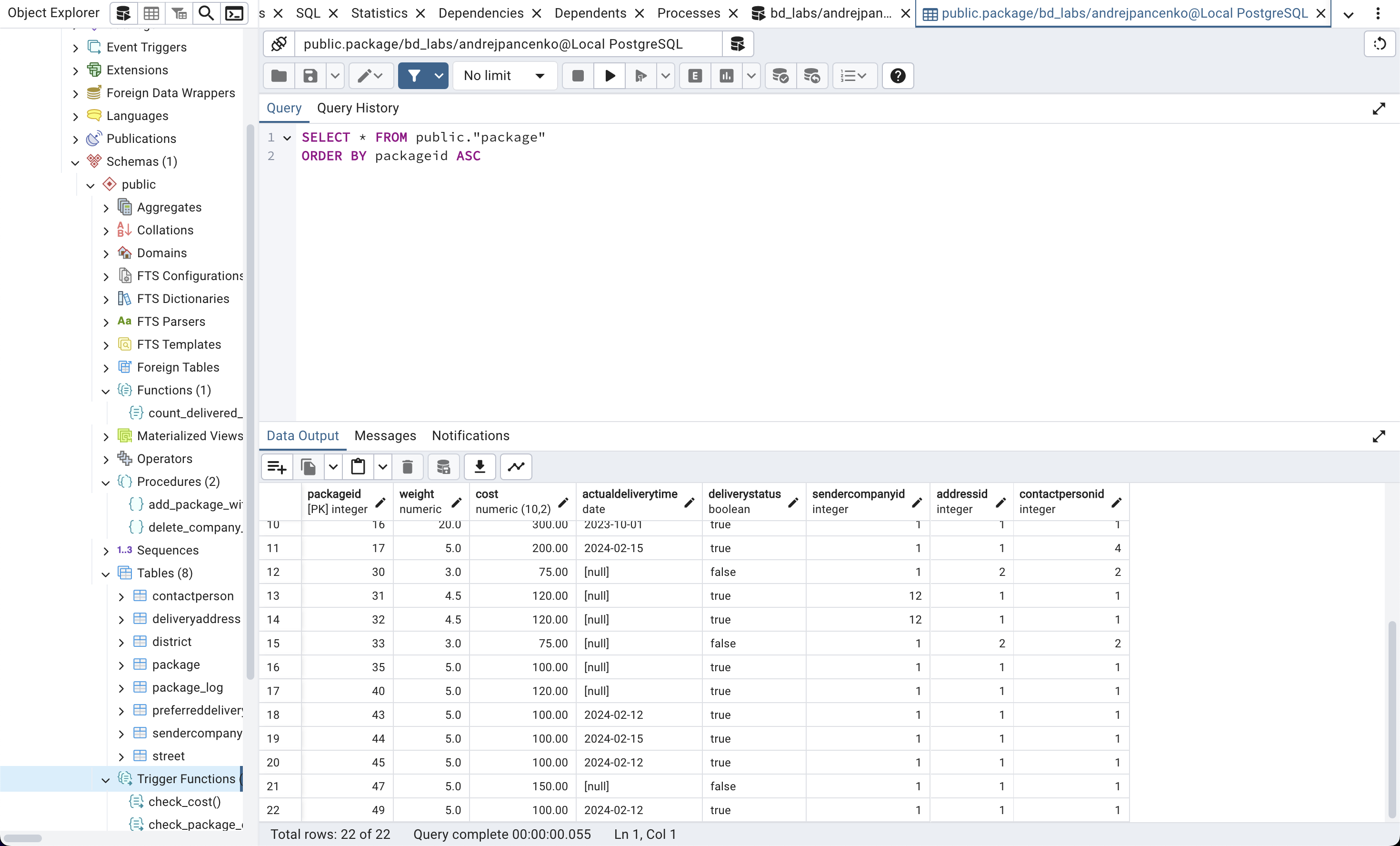
3. Trigger: Update delivery status before insertion ( update\_status\_before\_insert )

Insert with correct delivery date (status must be TRUE):

INSERT INTO Package (Weight, Cost, ActualDeliveryTime , SenderCompanyID , AddressID , ContactPersonID )

VALUES(5.0, 100.00, '2024-02-12', 1, 1, 1);

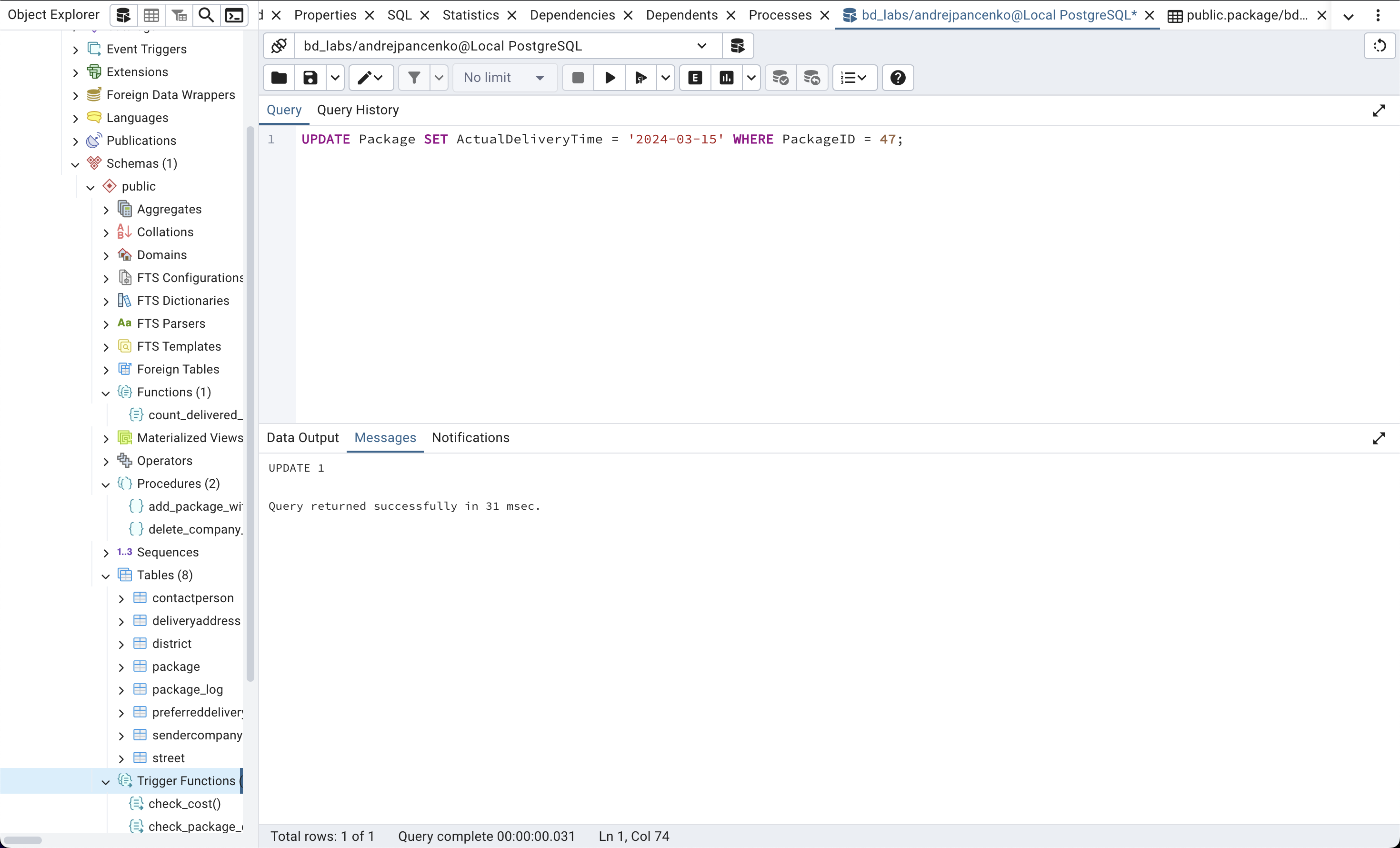


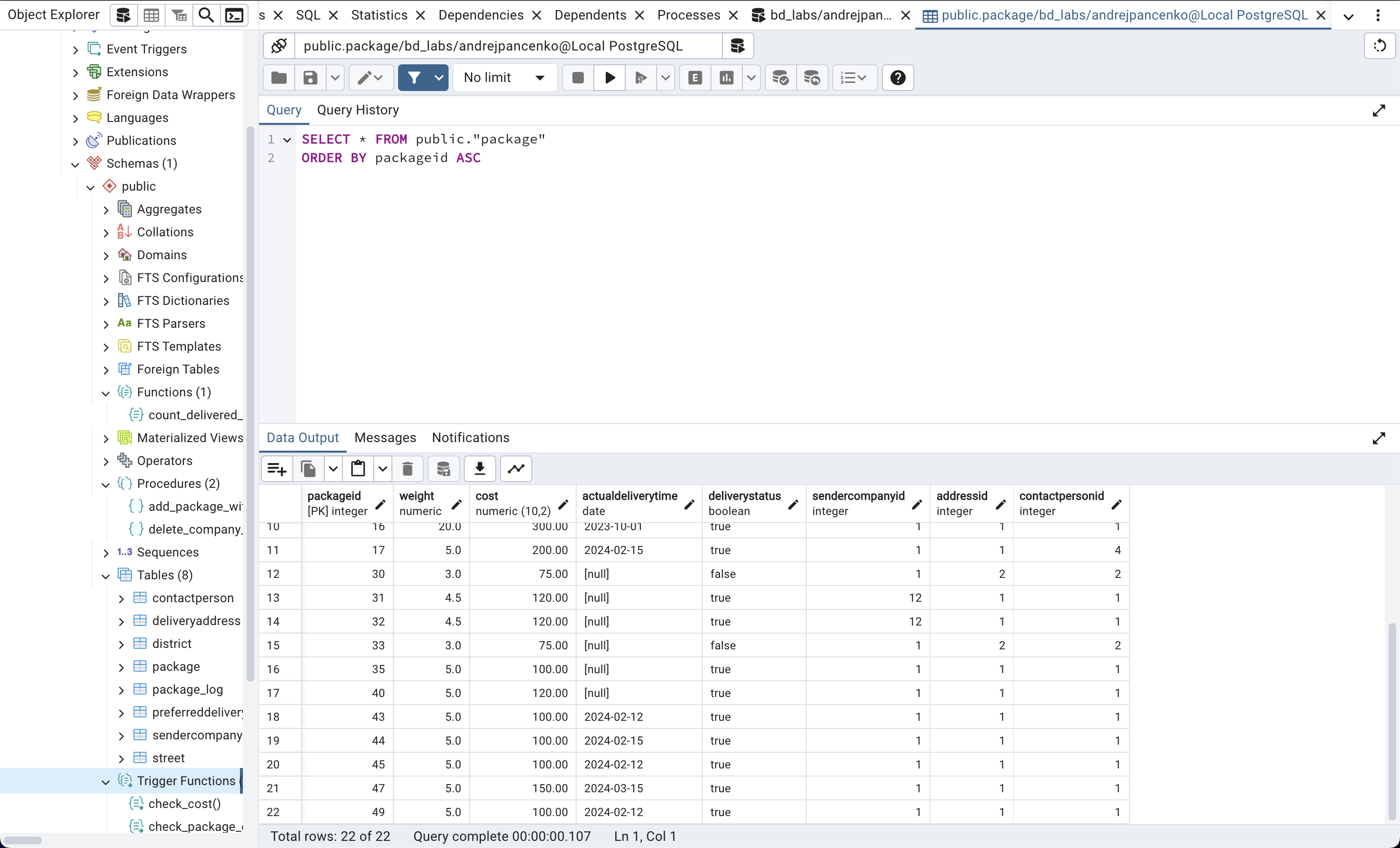


4. Trigger: Update delivery status before update ( update\_status\_before\_update )

Update with new delivery date (status should change to TRUE):

UPDATE Package SET ActualDeliveryTime = '2024-03-15' WHERE PackageID = 1;



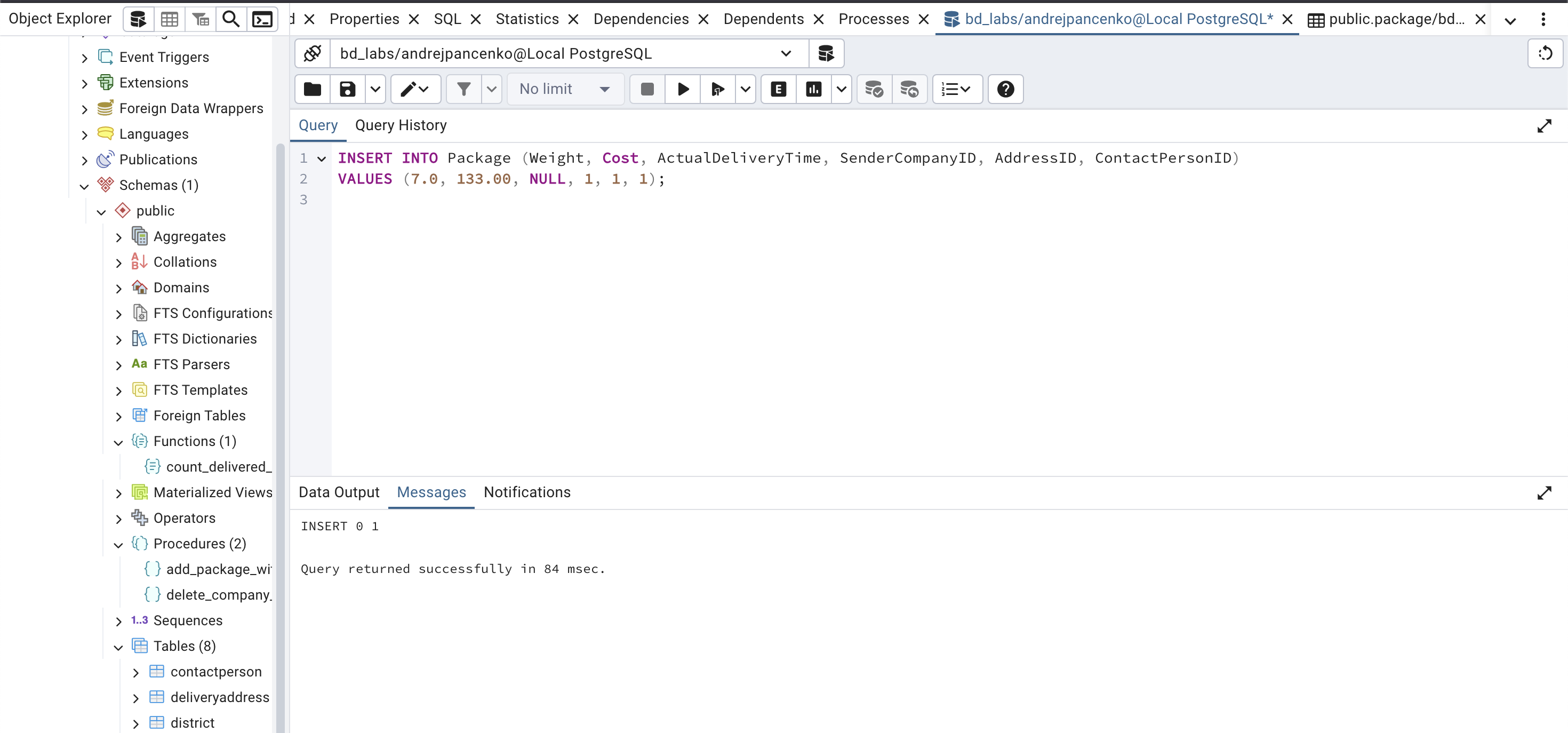


5. Trigger: Logging changes after insertion ( log\_after\_insert )

Insert with correct value (log should be written):

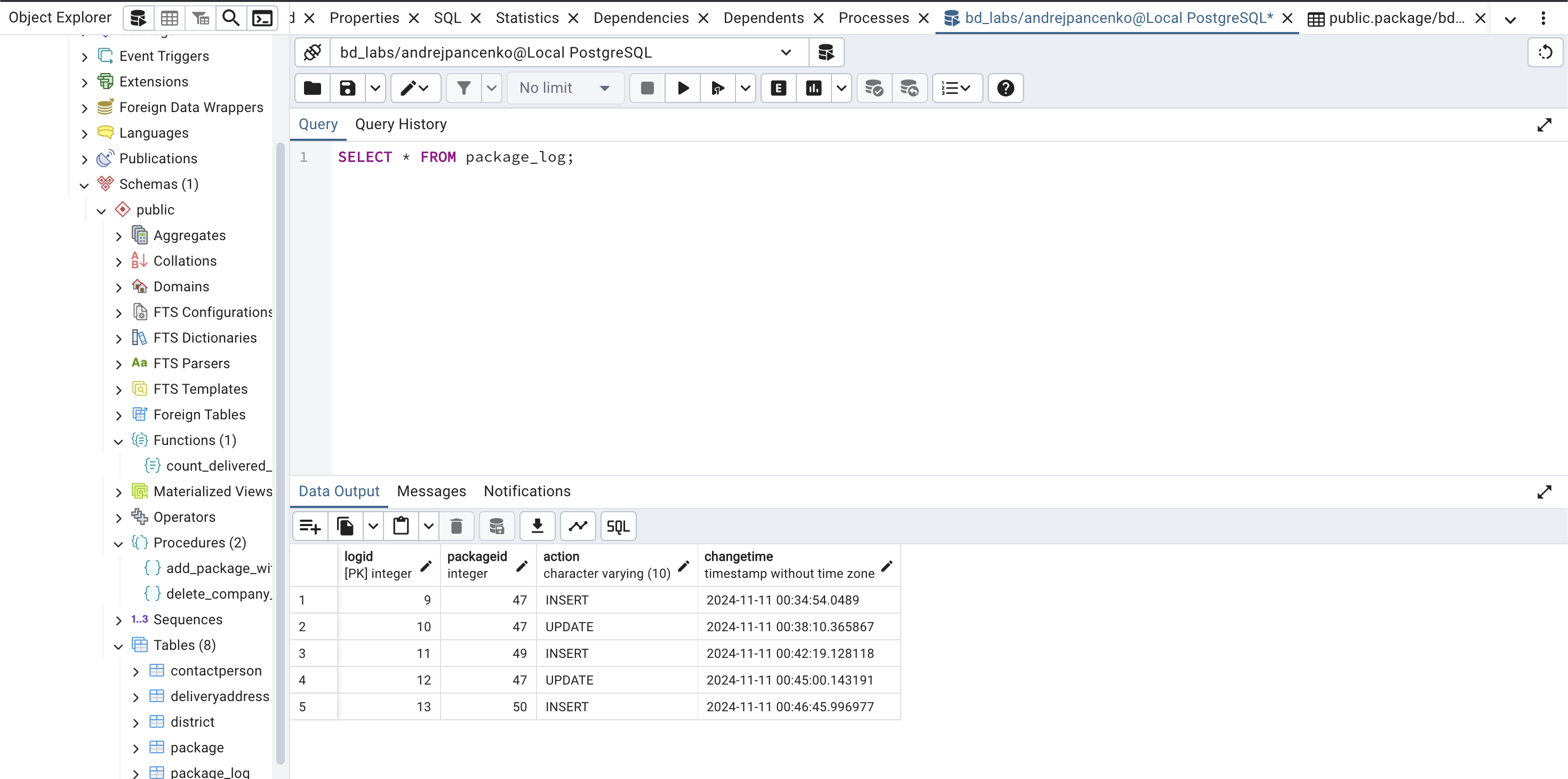
INSERT INTO Package (Weight, Cost, ActualDeliveryTime , SenderCompanyID , AddressID , ContactPersonID )

VALUES (5.0, 100.00, NULL, 1, 1, 1);



Checking logs after insertion:

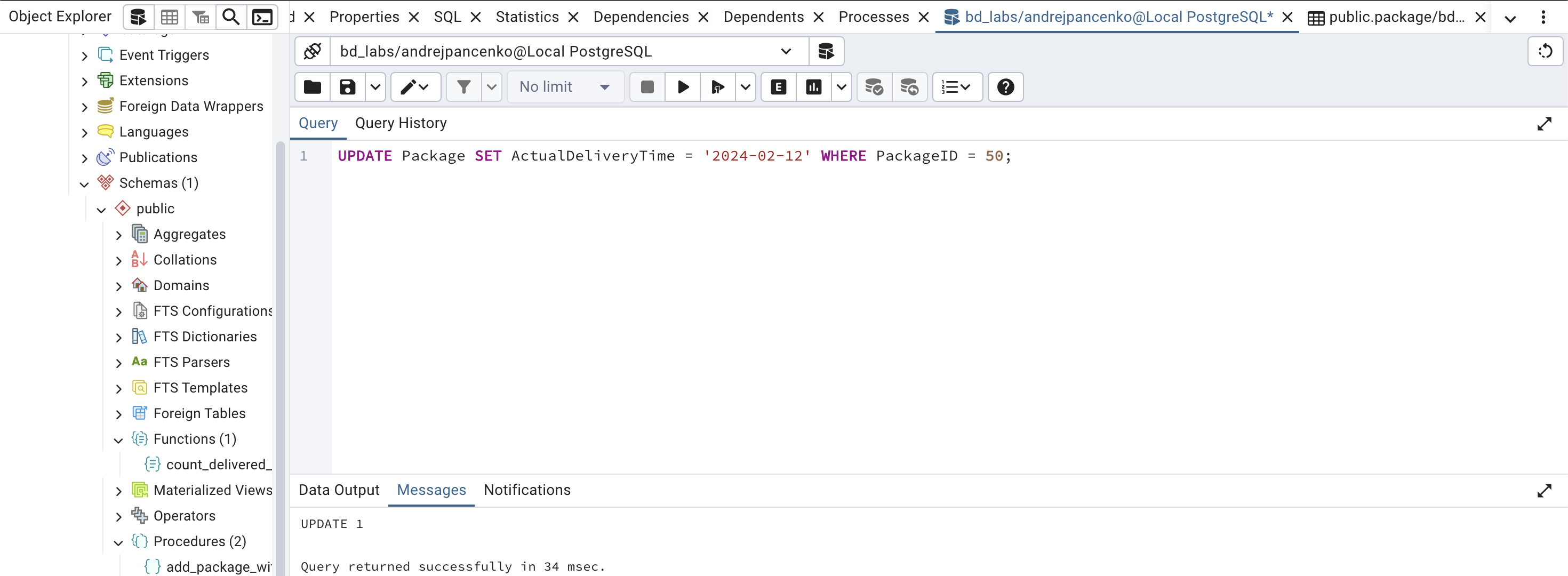
SELECT \* FROM package\_log ;​​



6. Trigger: Logging changes after update ( log\_after\_update )

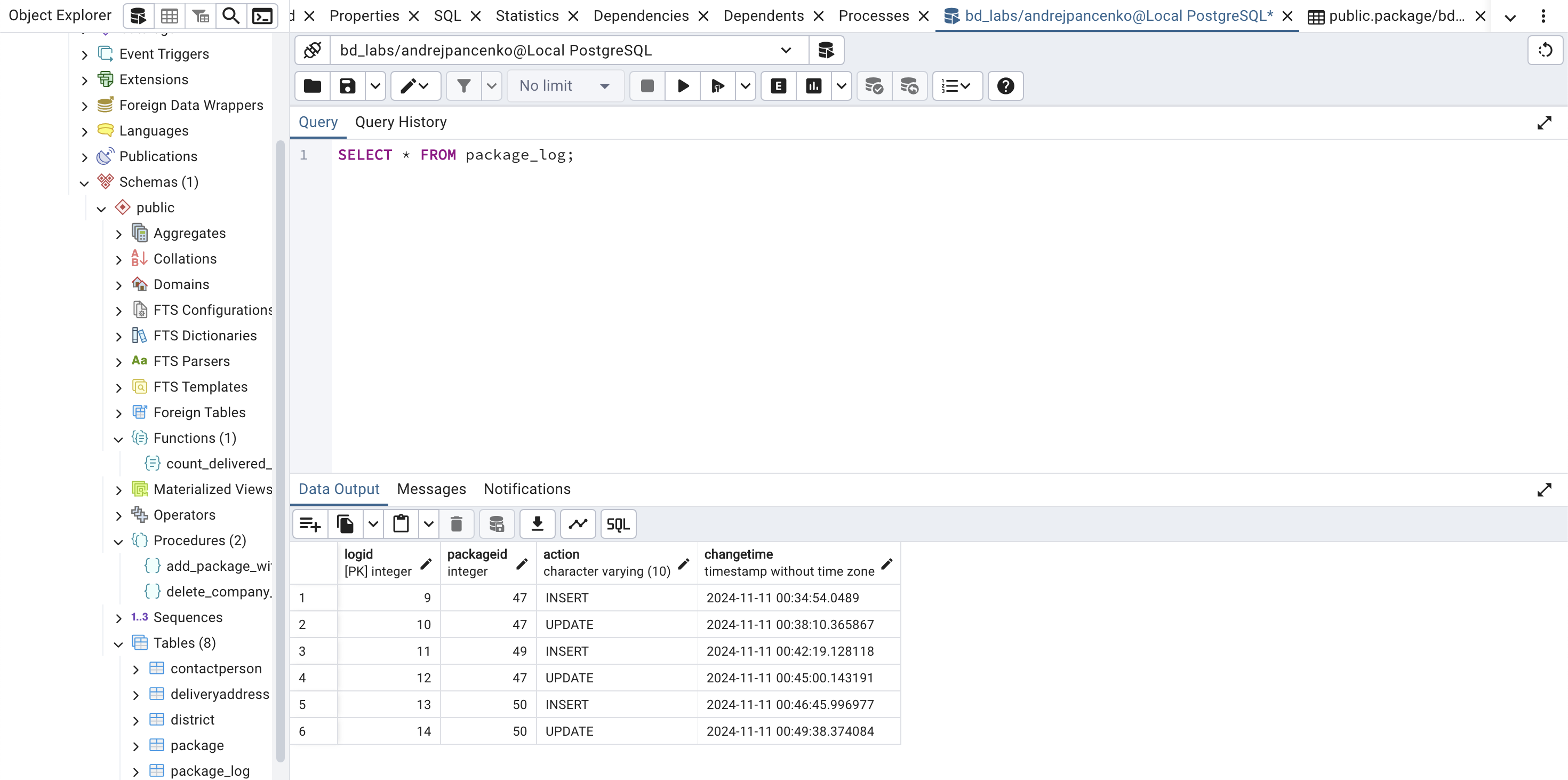
Updating an existing entry (a log should be written):

UPDATE Package SET ActualDeliveryTime = '2024-02-12' WHERE PackageID = 1;



Checking logs after update:

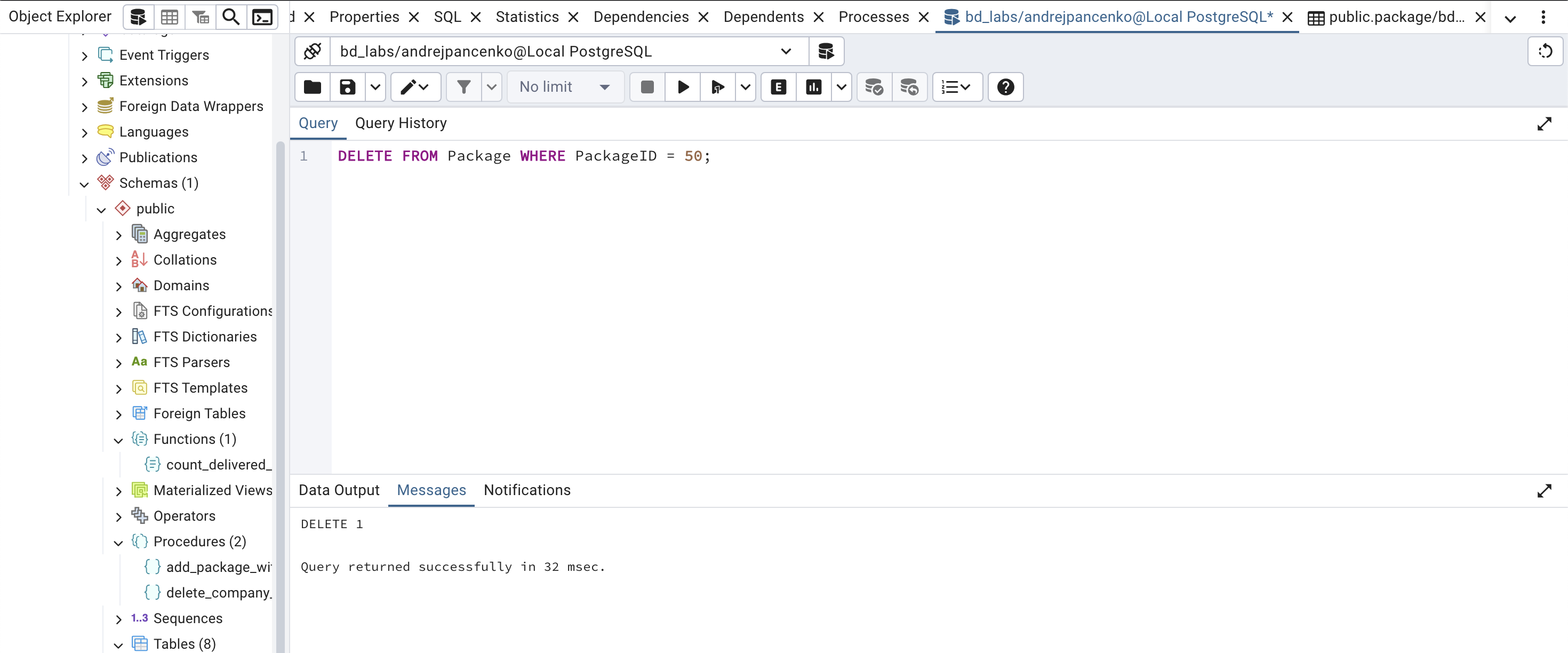
SELECT \* FROM package\_log ;



7. Trigger: Logging changes after deletion ( log\_after\_delete )

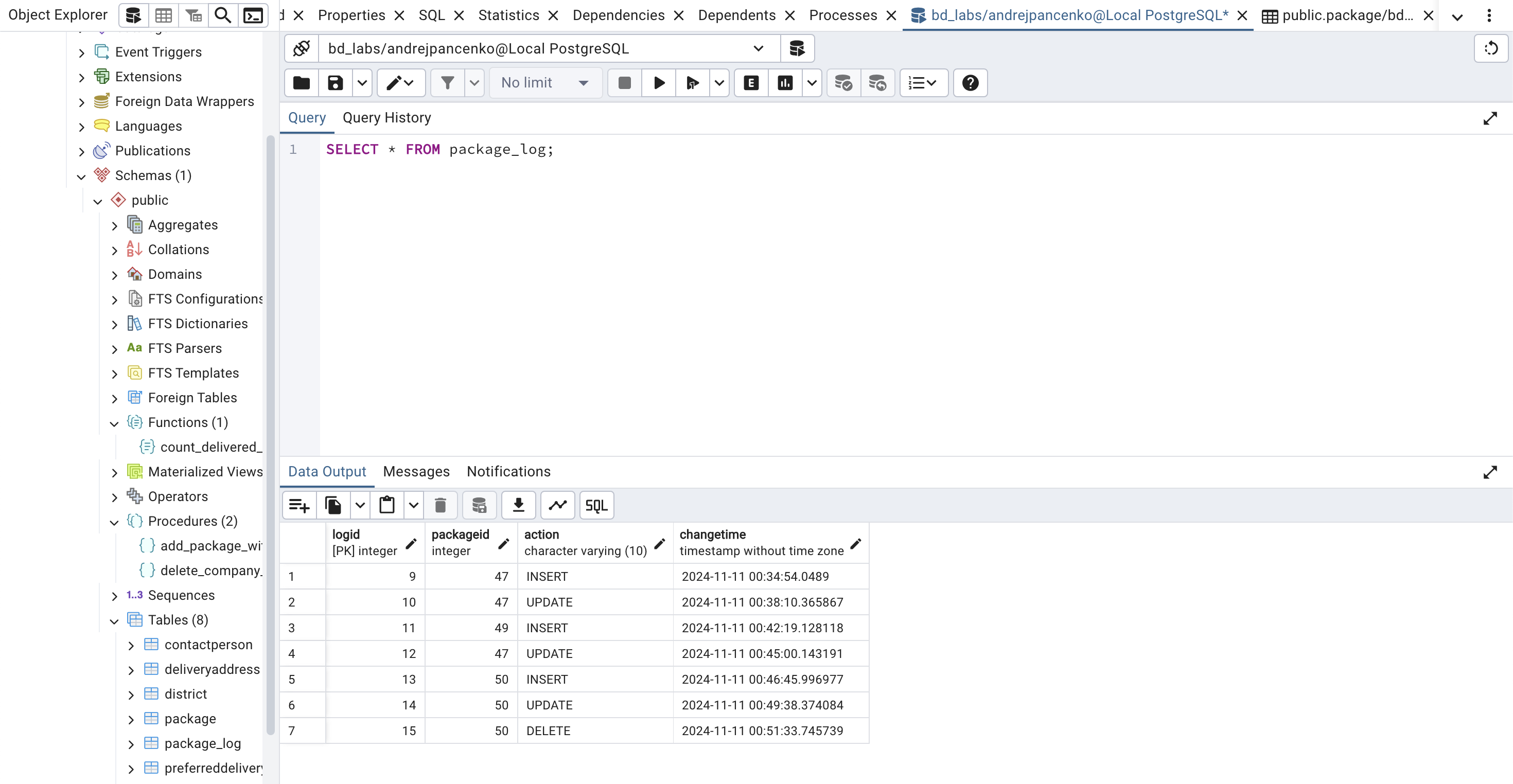
Deleting a record (a log should be written):

DELETE FROM Package WHERE PackageID = 50;



Checking logs after deletion:

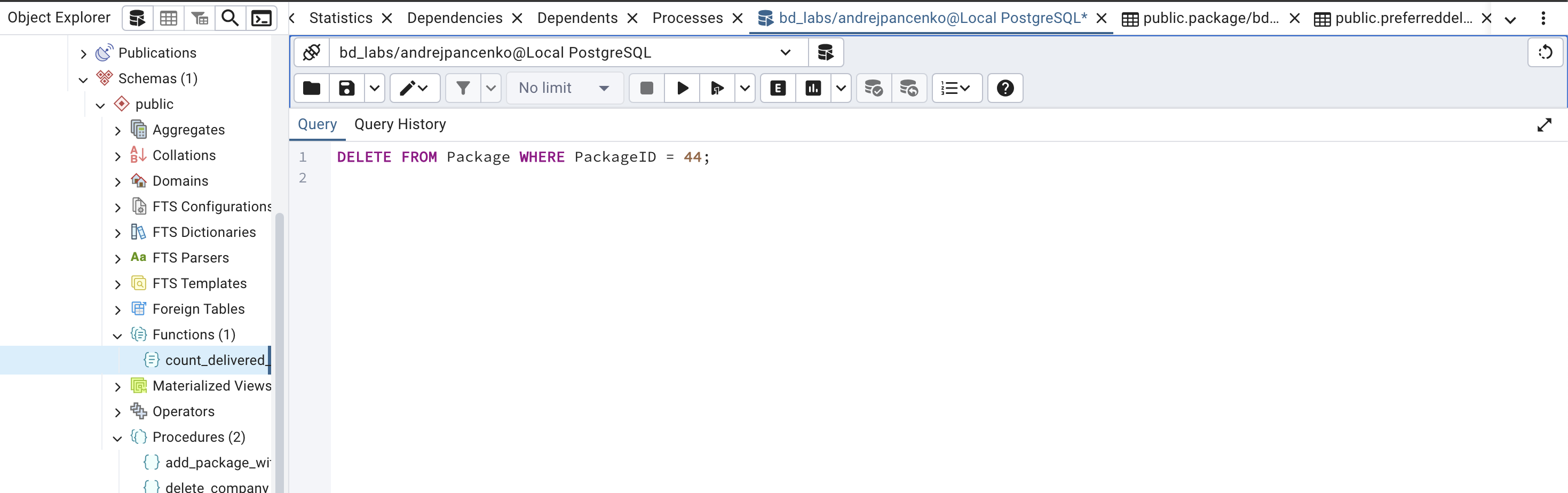
SELECT \* FROM package\_log ;

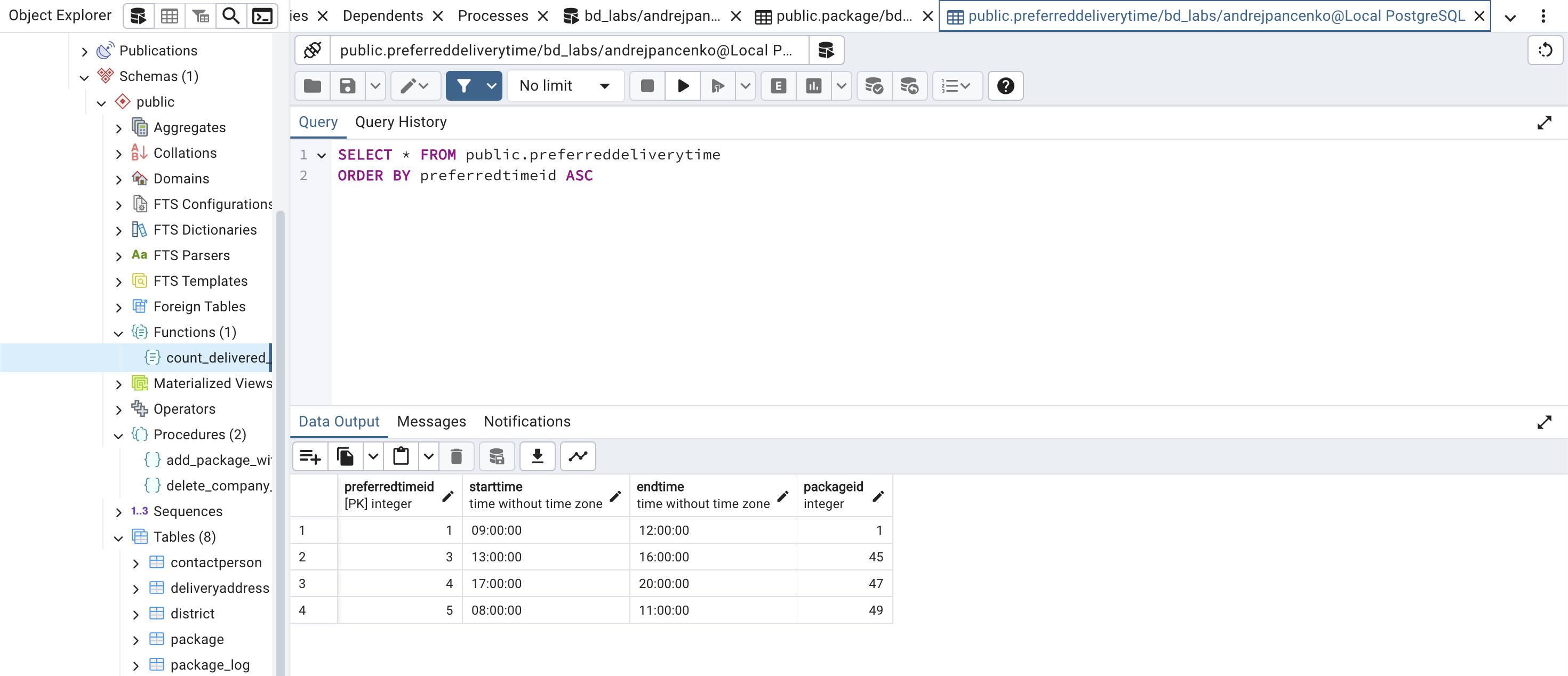


8. Trigger: Remove related records from PreferredDeliveryTime before removing from Package

Deleting a record from the Package table (related records from the PreferredDeliveryTime table must be deleted ):

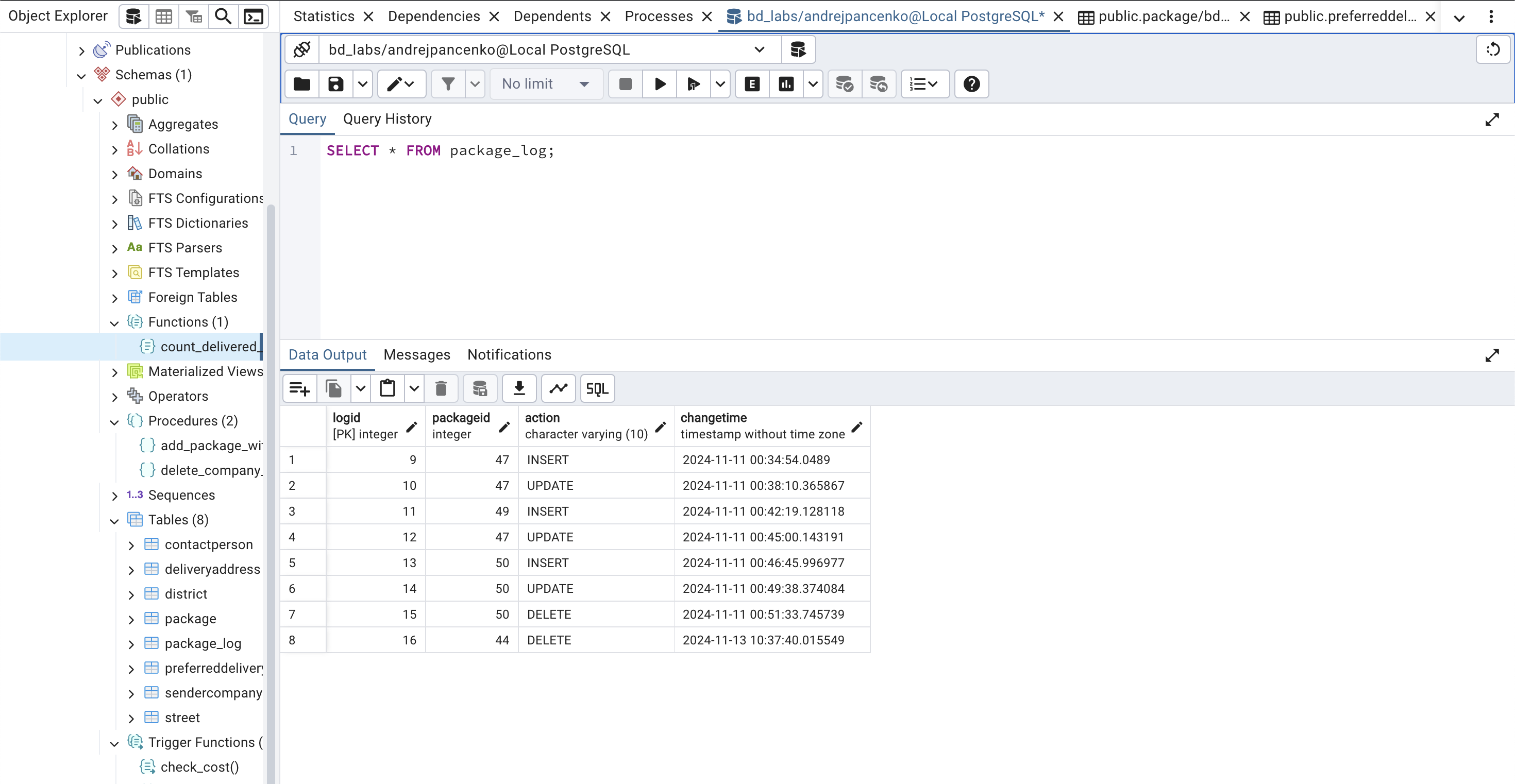
DELETE FROM Package WHERE PackageID = 44;





Checking logs after deletion:

SELECT \* FROM package\_log ;



**Conclusions about the use of triggers in the developed database :**

Using triggers in the developed database significantly improves data management and increases its integrity. Triggers such as check\_cost\_before\_insert and check\_cost\_before\_update ensure that the entered data is validated, preventing records with incorrect values, such as negative cost, from being inserted or updated. This helps maintain the integrity of the database and avoid errors. The update\_status\_before\_insert and update\_status\_before\_update triggers automatically set the package delivery status depending on whether the actual delivery date is available, which simplifies data management and ensures that the data is up-to-date. Logging changes using the log\_after\_insert, log\_after\_update and log\_after\_delete triggers records all actions that occur in the Package table , allowing for auditing and tracking the change history. This is useful for analysis and debugging. In addition, triggers allow you to implement complex business rules directly at the database level, reducing the need for additional logic at the application level. They can also roll back transactions if errors or data integrity violations are detected, which provides an additional layer of protection. In general, using triggers reduces the load on client applications and makes the system more resilient to errors, which ultimately improves the quality of work with data and increases the efficiency of development.