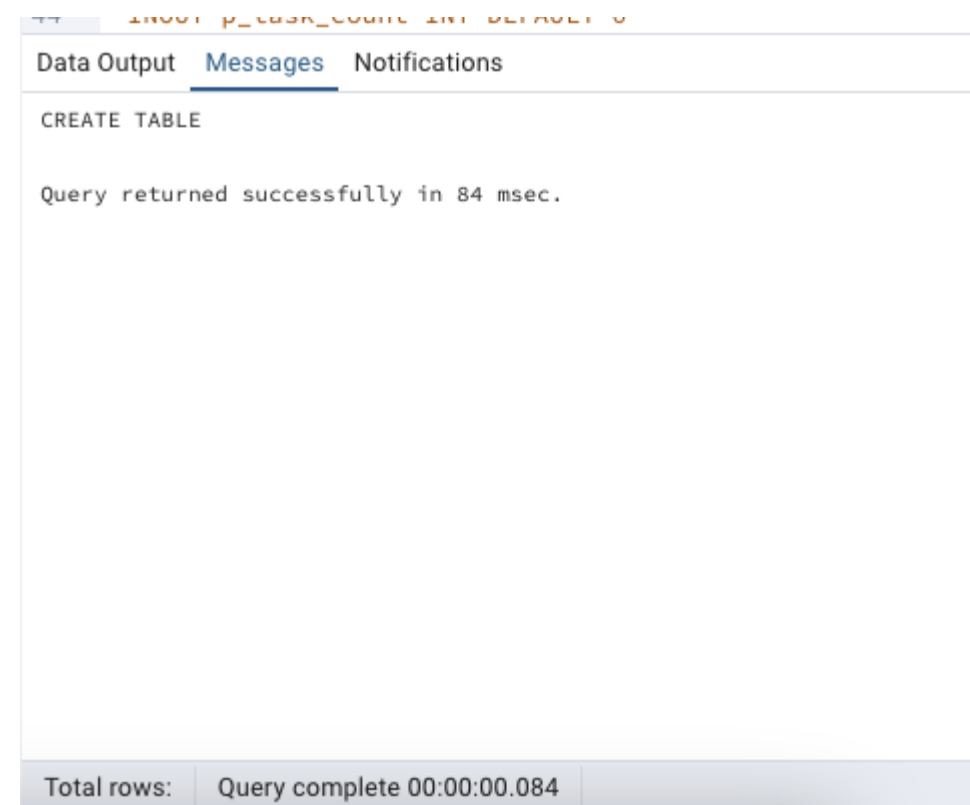


1. Create AFTER UPDATE trigger to track product price changes

- Create product_price_audit table with below columns:

```
audit_id SERIAL PRIMARY KEY,  
product_id INT,  
product_name VARCHAR(40),  
old_price DECIMAL(10,2),  
new_price DECIMAL(10,2),  
change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
user_name VARCHAR(50) DEFAULT CURRENT_USER
```

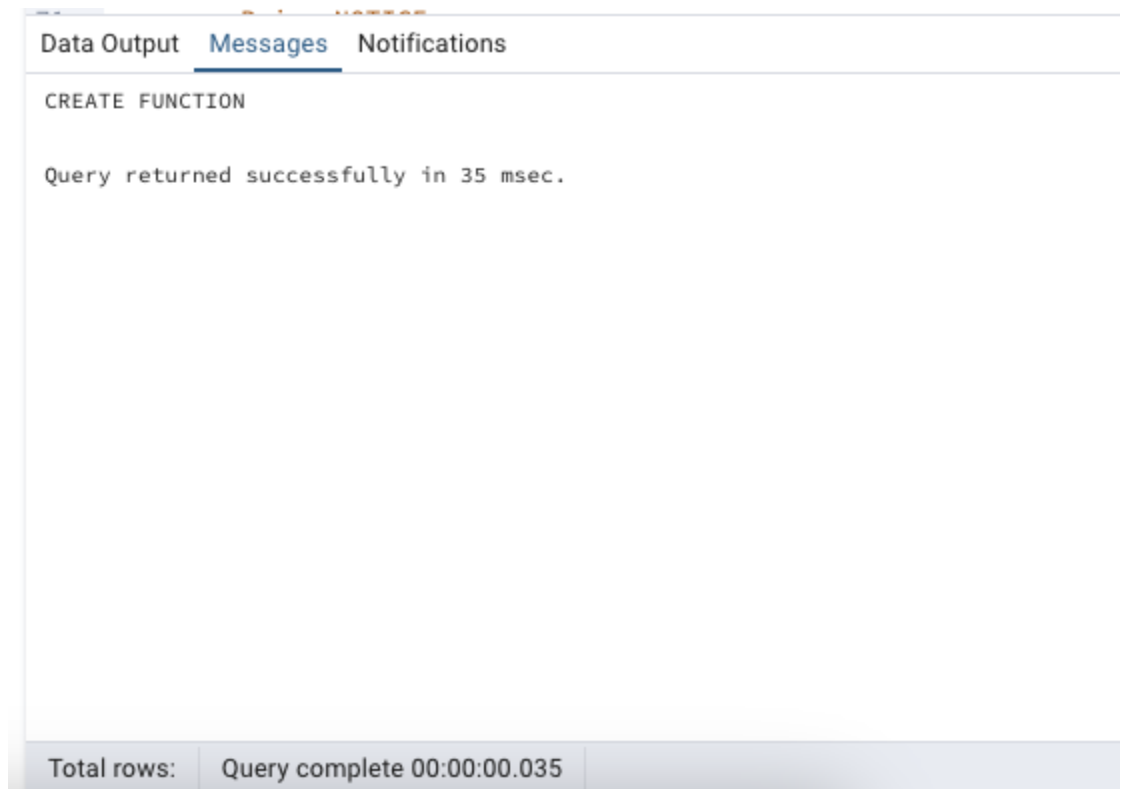


The screenshot shows a database management tool interface. At the top, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected. Below the tabs, the text 'CREATE TABLE' is visible. A message states 'Query returned successfully in 84 msec.' At the bottom, a status bar shows 'Total rows: 0' and 'Query complete 00:00:00.084'.

- Create a trigger function with the below logic:

```
INSERT INTO product_price_audit (  
    product_id,  
    product_name,
```

```
        old_price,  
        new_price  
    )  
VALUES (  
    OLD.product_id,  
    OLD.product_name,  
    OLD.unit_price,  
    NEW.unit_price  
);
```



The screenshot shows a database client window with three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected and displays the text 'CREATE FUNCTION' followed by 'Query returned successfully in 35 msec.'. At the bottom of the window, a status bar shows 'Total rows:' and 'Query complete 00:00:00.035'.

- Create a row level trigger for below event:

AFTER UPDATE OF unit_price ON products

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 35 msec.

Total rows: Query complete 00:00:00.035

- Test the trigger by updating the product price by 10% to any one product_id.

Data Output Messages Notifications

UPDATE 1

Query returned successfully in 29 msec.

Total rows: Query complete 00:00:00.029

Tras

Data Output Messages Notifications							
	audit_id	product_id	product_name	old_price	new_price	change_date	user_name
	[PK] integer	integer	character varying (40)	numeric (10,2)	numeric (10,2)	timestamp without time zone	character varying (50)
1	1	1	Chai	19.80	21.78	2025-05-05 09:44:10.388538	postgres

2. Create stored procedure using IN and INOUT parameters to assign tasks to employees

- Parameters:

IN p_employee_id INT,

IN p_task_name VARCHAR(50),

INOUT p_task_count INT DEFAULT 0

- Inside Logic: Create table employee_tasks:

```
CREATE TABLE IF NOT EXISTS employee_tasks (
    task_id SERIAL PRIMARY KEY,
    employee_id INT,
    task_name VARCHAR(50),
    assigned_date DATE DEFAULT CURRENT_DATE
);
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 52 msec.

Total rows: Query complete 00:00:00.052

- Insert employee_id, task_name into employee_tasks
- Count total tasks for employee and put the total count into p_task_count .
- Raise NOTICE message:
RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',
p_task_name, p_employee_id, p_task_count;

Data Output	Messages	Notifications
CREATE PROCEDURE		
Query returned successfully in 26 msec.		
Total rows:	Query complete 00:00:00.026	

After creating stored procedure test by calling it:

```
CALL assign_task(1, 'Review Reports');
```

Data OutputMessagesNotifications

SQL

	p_task_count integer
1	1

Total rows: 1Query complete 00:00:00.037

You should see the entry in employee_tasks table.

Data OutputMessagesNotifications

SQL

	task_id [PK] integer	employee_id integer	task_name character varying (50)	assigned_date date
1	1	1	Review Reports	2025-05-05

Total rows: 1Query complete 00:00:00.060

