

Funkcje i triggerzy

Funkcje SQL

Wewnątrz funkcji SQL można zamieszczać kod poleceń SQL.

Składnia:

```
CREATE OR REPLACE FUNCTION nazwa(parametry) RETURNS typ AS
'
    POLECENIA SQL
' LANGUAGE SQL
```

Funkcje SQL

Przykład:

```
CREATE OR REPLACE FUNCTION fsum(int,int)
RETURNS int AS
'
    SELECT $1 + $2;
' LANGUAGE SQL
```

\$x - parametr x

Funkcje SQL

Przykład:

```
CREATE OR REPLACE FUNCTION fsum(int,int)
RETURNS int AS
'
    SELECT $1 + $2;
' LANGUAGE SQL
```

\$x - parametr x

The screenshot shows a PostgreSQL Query Editor window with the following content:

- Top bar: `lecture/user@postgres`
- Tabs: `Query Editor` (selected), `Query History`
- Query text (line numbers 1-6):


```
1 CREATE OR REPLACE FUNCTION fsum(int,int)
2 RETURNS int AS
3 '
4     SELECT $1 + $2;
5 ' LANGUAGE SQL
6
```
- Bottom bar: `Data Output`, `Explain`, `Messages` (selected), `Notifications`
- Messages panel:


```
CREATE FUNCTION
Query returned successfully in 50 msec.
```

Funkcje SQL

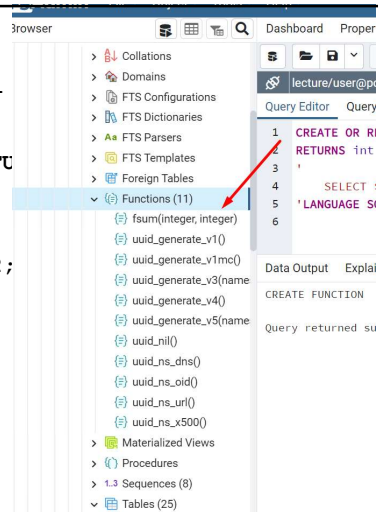
Przykład:

```
CREATE OR REPLACE FUNCTION fsum(int, int)
RETURNS int AS
```

```

    SELECT $1 + $2;
'LANGUAGE SQL
```

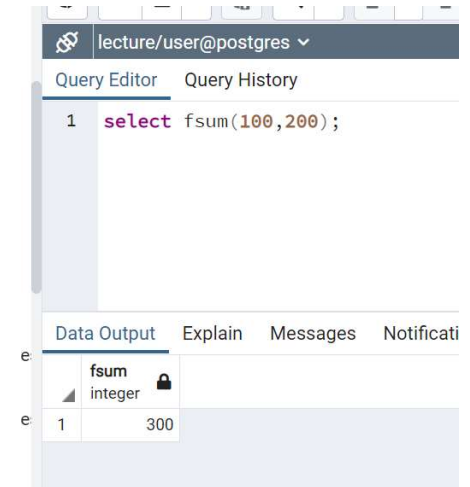
\$x - parametr x



Funkcje SQL

Wywoływanie:

```
Select fsum(1,2);
```



Funkcje SQL

Ze względu, na fakt, że apostrof często używany jest wewnątrz wyrażeń SQL, aby zamieścić kod funkcji można zamiennie użyć składni z "\$\$" zamiast apostrofu:

```
CREATE OR REPLACE FUNCTION fsum(int,int)
```

```
RETURNS int AS
```

```
$$
```

```
    SELECT $1 + $2;
```

```
$$ LANGUAGE SQL
```

Pomiędzy \$\$ można użyć \$body\$

Funkcje SQL

Funkcja, która nie zwraca wartości:

```
CREATE OR REPLACE FUNCTION nazwa(parametry)
```

```
RETURNS void AS
```

```
$$
```

```
    . . .
```

```
$$
```

```
LANGUAGE SQL
```

Funkcje SQL

Funkcja, która zwraca pojedynczą wartość - przykład:

```
CREATE OR REPLACE FUNCTION f_count_customers_rows()
RETURNS int AS
$$
    SELECT COUNT(*) from customers;
$$ LANGUAGE SQL
```

Query Editor		Query History
1	select	f_count_customers_rows()
2		
Data Output		Explain
f_count_customers_rows		integer
1		91

Funkcje SQL

Parametry - umieszczane w nawiasach okrągłych

param1 typ, param2 typ ...

Parametry mogą być widoczne po ich nazwach lub przy wykorzystaniu składni z dolarem - określenie porządku (od 1).

Przykład:

```
CREATE OR REPLACE FUNCTION f_count_names_in_empl(name varchar)
RETURNS int AS
$$
    SELECT COUNT(*) from employees where first_name=name;
$$ LANGUAGE SQL
```

Funkcje

Parametry - umieszczane w nawiasach okrągłych
param1 typ, param2 typ ...

Parametry mogą być widoczne po ich nazwach lub przy wykorzystaniu składni z dolarem - określenie porządku (od 1).

Przykład:

```
CREATE OR REPLACE FUNCTION f_count_names_in_empl(name varchar)
RETURNS int AS
$$
    SELECT COUNT(*) from employees where first_name=name;
$$ LANGUAGE SQL
```

Query Editor		Query History
1	select	f_count_names_in_empl('Nancy')
2		
3		
Data Output		Explain
f_count_names_in_empl		integer
1		1

Funkcje SQL

Zwracanie **pojedynczego** wiersza w postaci **tablicy**:

```
CREATE OR REPLACE FUNCTION nazwa(parametry)
RETURNS nazwaTabeliWyniku AS
$$
    ...
$$
LANGUAGE SQL
```

W celu zwrócenia danych **tablicowych** należy określić nazwę konkretnej tabeli po poleceniu RETURNS.

FUNKCJA ZWRÓCI TYLKO **PIERWSZY** WIERZS WYNIKU w postaci TABELI.

Funkcje SQL

Przykład:

```
CREATE OR REPLACE FUNCTION f_get_all_from_customers()
RETURNS customers AS
$$
```

```
    SELECT * FROM customers;
```

```
$$
```

```
LANGUAGE SQL
```

Dane są zwracane

w postaci

tablicy:

The screenshot shows a PostgreSQL query editor with the following content:

```
lecture/user@postgres
Query Editor Query History
1 SELECT f_get_all_from_customers()
2
3
```

Below the query editor, the 'Data Output' tab is active, displaying the result of the query as a table:

f_get_all_from_customers	
customers	
1	(ALFKI Alfreds Futterkiste Maria Anders Sales Representative Obere Str. 57 Berlin, 12209, Germany 030-0074321,030-0076545)

Funkcje SQL

Przykład:

Aby wyświetlić wiersz w postaci tabelowej:

```
SELECT (nazwa_funkcji()).*:
```

The screenshot shows a PostgreSQL query editor with the following content:

```
lecture/user@postgres
Query Editor Query History
1 SELECT (f_get_all_from_customers()).*
2
3
```

Below the query editor, the 'Data Output' tab is active, displaying the result of the query as a table:

customer_id	company_name	contact_name	contact_title	address	city	region	postal_code
1	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	12209

Funkcje SQL

Zwracanie **wielu** wierszy w postaci **tablicy**:

```
CREATE OR REPLACE FUNCTION nazwa(parametry)
```

```
RETURNS SETOF nazwaTabeliWyniku AS
```

```
$$
```

```
    ...
```

```
$$
```

```
LANGUAGE SQL
```

FUNKCJA ZWRÓCI WSZYSTKIE WYNIKI REZULTATU w postaci **TABLICY** (nie TABELI).

Funkcje SQL

Zwracanie **wielu** wierszy w postaci **tablicy**:

```
CREATE OR REPLACE FUNCTION f_get_all_customers()
```

```
RETURNS SETOF customers AS
```

```
$$
```

```
    SELECT * FROM customers;
```

```
$$
```

```
LANGUAGE SQL
```

lecture/user@postgres

Query Editor Query History Scratch Pad

```
1 SELECT f_get_all_customers()
2
3
```

Data Output Explain Messages Notifications

f_get_all_customers
customers

1	(ALFKI;Alfreds Futterkiste;Maria Anders;Sales Representative;Obere Str. 57;Berlin,12209,Germany,030-0074321,030-0076545)
2	(ANATR;Ana Trujillo Emparedados y helados;Ana Trujillo;Owner;Avda. de la Constitución 2222;México D.F.;05021,Mexico;(5) 555-4729;{(5) 555-3745})
3	(ANTON;Antonio Moreno Taquería;Antonio Moreno;Owner;Mataderos 2312;México D.F.;05023,Mexico;(5) 555-3932;)
4	(AROUT;Around the Horn;Thomas Hardy;Sales Representative;120 Hanover Sq.;London,WA1 1DP;UK;(171) 555-7788;(171) 555-6750)
5	(BERGS;Berglunds snabbköp;Christina Berglund;Order Administrator;Berguvsvägen 8;Luleå,S-958 22;Sweden;0921-12 34 65;0921-12 34 67)
6	(BLAUS;Blauer See Delikatessen;Hanna Moos;Sales Representative;Forsterstr. 57;Mannheim,68306,Germany,0621-08460,0621-08924)
7	(BLONP;Blondesddsi père et fils;Frédérique Citeaux;Marketing Manager;24, place Kléber;Strasbourg,67000,France,88.60.15.31,88.60.15.32)
8	(BOLID;Bóldo Comidas preparadas;Martin Sommer;Owner;C/ Araquil, 67;Madrid,28023,Spain;(91) 555 22 82;(91) 555 91 99)
9	(BONAP;Bon app';Laurence Lebihan;Owner;12, rue des Bouchers;Marseille,13008,France,91.24.45.40,91.24.45.41)
10	(BOTTM;Bottom-Dollar Markets;Elizabeth Lincoln;Accounting Manager;23 Tsawassen Blvd.;Tsawassen,BC,T2F 8M4;Canada;(604) 555-4729;{(604) 555-3745})

Funkcje SQL

Query Editor Query History Scratch Pad

```
1 SELECT (f_get_all_customers()).*
2
3
```

Data Output Explain Messages Notifications

customer_id	company_name	contact_name	contact_title	address	city	region	
1	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	[null]
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	[null]
3	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	[null]
4	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	[null]
5	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå	[null]
6	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	[null]
7	BLONP	Blondesddsi père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	[null]
8	BOLID	Bóldo Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid	[null]
9	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille	[null]
10	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC
11	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London	[null]

Funkcje SQL

Używanie funkcji jako źródła danych:

```
SELECT lista_kolumn
FROM nazwa_funkcji();
```

Przykład:

Query Editor Query History

```
1 SELECT * from f_get_all_customers()
2
3
```

Data Output Explain Messages Notifications

customer_id	company_name	contact_name	contact_title	address	
1	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución
3	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312
4	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.
5	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8
6	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57

Funkcje SQL

Zwracanie **TABELI** (nie tablicy):

```
CREATE OR REPLACE FUNCTION nazwa(parametry)
RETURNS TABLE (col1 typ, col2 typ, ...) AS
$$
...
$$
LANGUAGE SQL
```

FUNKCJA ZWRÓCI WSZYSTKIE WYNIKI REZULTATU w postaci **TABELI** (nie tablicy).

Funkcje SQL

Zwracanie **TABELI** (nie tablicy):

```
CREATE OR REPLACE FUNCTION f_get_employees_data()
RETURNS TABLE(name varchar, surname varchar) AS
$$
    SELECT first_name as name, last_name as surname FROM
employees;
$$
LANGUAGE SQL
```

Funkcje SQL

Zwracanie **TABELI** (ni

```
CREATE OR REPLACE FU
RETURNS TABLE(name v
SELECT first_na
employees;
LANGUAGE SQL
```

lecture/user@postgres	
Query Editor Query History	
1	SELECT f_get_employees_data()
2	
3	
4	
5	
6	
Data Output Explain Messages Notifications	
f_get_employees_data record	
1	(Nancy,Davolio)
2	(Andrew,Fuller)
3	(Janet,Leverling)
4	(Margaret,Peacock)
5	(Steven,Buchanan)
6	(Michael,Suyama)
7	(Robert,King)

Funkcje SQL

Zwracanie **TABELI** (n

```
CREATE OR REPLACE F
RETURNS TABLE(name
SELECT first_n
employees;
LANGUAGE SQL
```

lecture/user@postgres	
Query Editor Query History	
1	SELECT (f_get_employees_data()).*
2	
3	
4	
5	
6	
Data Output Explain Messages Notifications	
name surname	
character varying character varying	
1	Nancy Davolio
2	Andrew Fuller
3	Janet Leverling
4	Margaret Peacock
5	Steven Buchanan
6	Michael Suyama

Funkcje SQL

Zwracanie **TABELI** (

```
CREATE OR REPLACE
RETURNS TABLE(name
SELECT first_
employees;
LANGUAGE SQL
```

lecture/user@postgres	
Query Editor Query History	
1	SELECT * from f_get_employees_data()
2	
3	
4	
5	
6	
Data Output Explain Messages Notifications	
name surname	
character varying character varying	
1	Nancy Davolio
2	Andrew Fuller
3	Janet Leverling
4	Margaret Peacock

Funkcje SQL

Wartości domyślne parametrów

Parametry z wartościami domyślnymi należy umieścić na końcu deklaracji listy parametrów.

Składnia:

```
CREATE OR REPLACE FUNCTION nazwa(p1 int, p2 varchar, p3 int
DEFAULT 1, p4 int DEFAULT 1000);
```

Funkcje SQL

```
CREATE OR REPLACE FUNCTION f_get_num(x int DEFAULT 3)
RETURNS TABLE(liczba int) AS
$$
    SELECT * FROM nums WHERE data >= x;
$$
LANGUAGE SQL
```

Funkcje SQL

```
CREATE OR REPLACE FUNCTION f_get_num(x int DEFAULT 3)
RETURNS TABLE(liczba int) AS
```

\$\$

```
    SELECT * FROM nums WHERE data >= x;
```

\$\$

LANGUAGE SQL

The screenshot shows the PostgreSQL Query Editor interface. The top bar indicates the user is 'lecture/user@postgres'. Below the editor, the 'Data Output' tab is active, displaying a table with two columns: 'liczba' (integer) and 'id' (integer). The table contains two rows of data.

liczba	id
4	1
5	2

The screenshot shows the PostgreSQL Query Editor interface. The top bar indicates the user is 'lecture/user@postgres'. Below the editor, the 'Data Output' tab is active, displaying a table with two columns: 'liczba' (integer) and 'id' (integer). The table contains four rows of data.

liczba	id
1	1
2	2
4	3
5	4

Funkcje SQL

Usuwanie funkcji:

```
DROP FUNCTION [IF EXISTS] nazwa(lista_parametrów) [cascade |
restrict];
```

cascade - usuwa też elementy zależne

restrict - odmawia usunięcia funkcji, gdy posiada obiekty zależne

Funkcje i Procedury

Funkcje nie wspierają transakcji, procedury wspierają.

Procedury nie zwracają wyniku - nie można używać wyrażenia:
return wartość;

ale można wyjść z procedury używając:
return;

Wywoływanie procedur:
CALL nazwa_procedury();

Procedury mogą używać parametrów.

Funkcje i Procedury

Do wywoływania procedury można też wykorzystać polecenie: EXECUTE

SKŁADNIA:

CREATE OR REPLACE PROCEDURE nazwa(parametry)

AS

\$\$

DECLARE

deklaracja zmiennych

BEGIN

ciało procedury

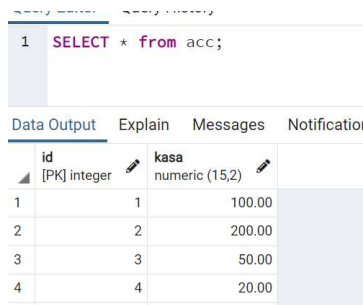
END

\$\$

LANGUAGE PLPGSQL;

Funkcje i Procedury

```
CREATE TABLE acc(id serial primary key, kasa dec(15,2) not null);
INSERT into acc(kasa) values (100), (200), (50), (20);
```



The screenshot shows a SQL query result in a web interface. The query is `SELECT * from acc;`. The result is displayed in a table with 4 rows and 2 columns: `id` (integer, primary key) and `kasa` (numeric(15,2)).

id	kasa
1	100.00
2	200.00
3	50.00
4	20.00

Funkcje i Procedury

```
CREATE OR REPLACE PROCEDURE p_transf (
src int, dest int, ile dec
) as
$$
BEGIN
UPDATE acc SET kasa = kasa - ile where id = src;
UPDATE acc SET kasa = kasa + ile where id = dest;
COMMIT;
END;
$$
LANGUAGE PLPGSQL;
```


Funkcje

```

1 CREATE OR REPLACE PROCEDURE p_transf (
2   src int, dest int, ile dec
3 ) as
4   $$
5 BEGIN
6   UPDATE acc SET kasa = kasa - ile where id = src;
7   UPDATE acc SET kasa = kasa + ile where id = dest;
8   COMMIT;
9 END;
10
11 UPDATE acc SET
12
13 UPDATE acc SET
14
15 COMMIT;
16
17 END;
18
19 $$
20
21 LANGUAGE PLPGSQL;

```

Query returned successfully in 40 msec.

Funkcje i Procedury

Wywołanie:

CALL p_transf(1,2,100):

Query Editor Query History

```

1 CALL p_transf(1,2,100)
2
3
4

```

Data Output Explain Messages Notifications

CALL

Query returned successfully in 39 msec.

1	select * from acc		
2			
3			

Data Output	Explain	Messages	Notifica
<div><div></div><div><div>id</div><div>[PK] integer</div></div></div>	<div><div></div><div>kasa</div></div> <div>numeric (15,2)</div>		
1	3	50.00	
2	4	20.00	
3	1	0.00	
4	2	300.00	

Funkcje i Procedury

Procedury nie zwracają bezpośrednio wartości na zewnątrz.

Można jednak użyć trybu INOUT dla parametru.

```

1 CREATE OR REPLACE PROCEDURE p_count(INOUT count integer default
2   0) AS
3   $$
4 BEGIN
5   SELECT COUNT(*) INTO count from acc;
6 END;
7
8 $$
9
10 LANGUAGE PLPGSQL;

```

Query Editor Query History

```

1 CALL p_count();
2
3

```

Data Output Explain Messages

count integer	
4	

Funkcje i Procedury

Usuwanie procedur:

DROP PROCEDURE [IF EXISTS] nazwa(parametry) [cascade | restrict]

Triggery

Trigger jest funkcją wywoływaną automatycznie gdy pojawia się zdarzenie powiązane z tabelą.

Do zdarzeń należą: INSERT, UPDATE, DELETE, TRUNCATE

Zdarzenie może być związane z:
tabelą, widokiem, tabelą obcą

Typy triggerów:

- BEFORE - wywoływany przed zdarzeniem,
- AFTER - wywoływany po zdarzeniu,
- INSTEAD - zamiast zdarzenia/operacji (inna obsługa).

Triggery

Jeżeli jest więcej niż jeden trigger w tabeli, będą one wywoływane w kolejności alfabetycznej.

Triggery mogą modyfikować dane przed i po modyfikacji.

Nie ma możliwości, żeby triggery otrzymywały parametry.

Nie można manualnie wywołać triggerów (wywoływane są one automatycznie).

Triggery

Typy triggerów

- triggery wierszy

FOR EACH ROW - funkcja zostanie wywołana dla każdego wiersza modyfikowanego w zdarzeniu

- triggery wyrażeń

FOR EACH STATEMENT - wywoływane jeden raz dla wyrażenia (bez względu na liczbę modyfikowanych wierszy).

Triggery

Kiedy	Zdarzenie	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tabele	Tabele i widoki
	TRUNCATE	-	Tabele
AFTER	INSERT/UPDATE/DELETE	TABELE	TABELE i widoki
	TRUNCATE	-	TABELE
INSTEAD OF	INSERT/UPDATE/DELETE	Widoki	-
	TRUNCATE	-	-

Aby stworzyć trigger:

1. Stworzyć funkcję zwracającą trigger
2. Zbindować stworzoną funkcję do tabeli z wykorzystaniem CREATE TRIGGER

Składnia funkcji:

```
CREATE FUNCTION nazwa_funkcji()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS $$
BEGIN
    logika
END;
$$
```

Triggery

Tworzenie triggera:

```
CREATE TRIGGER nazwa_triggera
{BEFORE | AFTER} {event}
ON nazwa_tabeli
[FOR [EACH] {ROW|STATEMENT}]
EXECUTE PROCEDURE nazwa_funkcji;
```

Triggery

Funkcja triggera otrzymuje dane odnośnie środowiska uruchomieniowego, które będą dostępne przez zbiór zmiennych lokalnych - przykładowo:

- OLD - stan wiersza przed wywołaniem zdarzenia (na poziomie statement-level - NULL),
- NEW - stan po wywołaniu zdarzenia,
- Zbiór zmiennych TG_... (np. TG_WHEN, TG_TABLE_NAME, ...)
- CURRENT_USER - bieżący użytkownik

Triggery

```
CREATE TABLE users(user_id SERIAL PRIMARY KEY, login
VARCHAR(50));
```

```
CREATE TABLE modifications(user_mod_id SERIAL PRIMARY KEY,
user_id INT NOT NULL, login VARCHAR(50) NOT NULL, time TIMESTAMP
NOT NULL);
```

```

CREATE OR REPLACE FUNCTION fn_trigger1()
RETURNS TRIGGER LANGUAGE PLPGSQL
AS
$$
BEGIN
IF NEW.login <> OLD.login THEN
INSERT INTO modifications(user_id, login, time)
VALUES
(
    OLD.user_id, OLD.login, NOW()
);
END IF;
RETURN NEW;
END;
$$

```

```

CREATE OR REPLACE FUNCTION fn_trigger1()
RETURNS TRIGGER
AS
$$
BEGIN
IF NEW.login <> OLD.login THEN
INSERT INTO modifications(user_id, login, time)
VALUES
(
    OLD.user_id, OLD.login, NOW()
);
END IF;
RETURN NEW;
END;
$$

```

Query Editor Query History

1 IF NEW.login <> OLD.login THEN
2 INSERT INTO modifications(user_id, login, time)
3 VALUES
4 (
5 OLD.user_id, OLD.login, NOW()
6);
7 END IF;
8 RETURN NEW;
9 END;
10 \$\$

Data Output Explain Messages Notifications

CREATE FUNCTION

Query returned successfully in 40 msec.

Triggery

```

CREATE TRIGGER trg_users_login_change
BEFORE UPDATE
ON users
FOR EACH ROW
EXECUTE PROCEDURE fn_trigger1();

```

Trigger

```

CREATE TRIGGER trg_users_login_change
BEFORE UPDATE
ON users
FOR EACH ROW
EXECUTE PROCEDURE fn_trigger1();

```

lecture/user@postgres

Query Editor Query History

1 CREATE TRIGGER trg_users_login_change
2 BEFORE UPDATE
3 ON users
4 FOR EACH ROW
5 EXECUTE PROCEDURE fn_trigger1();

Data Output Explain Messages Notifications

CREATE TRIGGER

Query returned successfully in 44 msec.

Triggery

```
INSERT INTO users(login) values ('Adam'), ('Jan'), ('Anna');
UPDATE users SET login='Jan' where login='Adam';
```

The screenshot shows a PostgreSQL query editor with the following query:

```
1 SELECT * FROM public.modifications
2 ORDER BY user_mod_id ASC
```

The results are displayed in a table with the following columns: user_mod_id, user_id, login, and time.

user_mod_id	user_id	login	time
1	1	Adam	2021-11-21 20:14:39.766978

Modyfikacja danych przy tworzeniu:

```
CREATE OR REPLACE FUNCTION tr_trigger2()
RETURNS TRIGGER LANGUAGE PLPGSQL
AS
$$
BEGIN
    IF NEW.login = 'Zenon' THEN
        NEW.login = 'Pan Zenon';
    END IF;
    RETURN NEW;
END;
$$
```

Modyfikacja danych przy tworzeniu:

```
CREATE TRIGGER tr_zmiana_loginu
BEFORE INSERT
ON users
FOR EACH ROW
EXECUTE PROCEDURE tr_trigger2();
```

Modyfikacja

```
CREATE TRIG
BEFORE INSE
ON users
FOR EACH RO
EXECUTE PRO
```

The screenshot shows a PostgreSQL query editor with the following query:

```
1 select * from users
```

The results are displayed in a table with the following columns: user_id, login, and time.

user_id	login	time
1	Jan	
2	Jan	
3	Juliusz	
4	xyz	
5	Anna	
6	Anna	

Przykład

Trigger logujący użytkownika dokonującego zmian

```
CREATE TABLE history(usr VARCHAR(50), time TIMESTAMP )
```

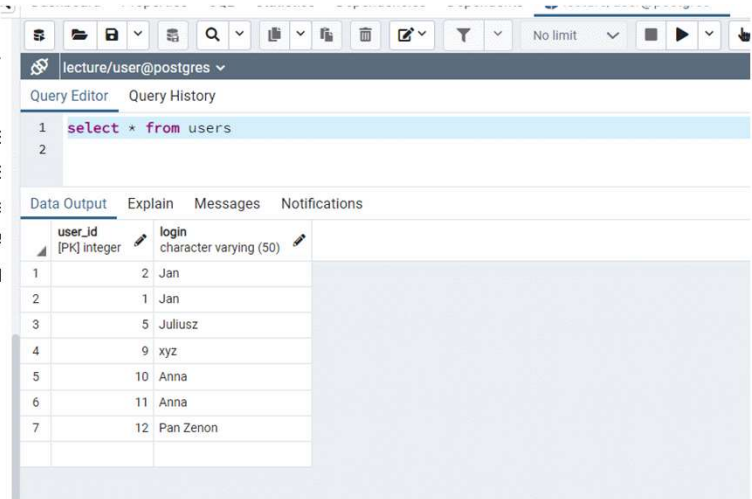
```
CREATE OR REPLACE FUNCTION fn_trigger_history()
RETURNS TRIGGER LANGUAGE PLPGSQL
AS
$$
BEGIN
IF NEW.login <> OLD.login THEN
INSERT INTO history(usr, time)
VALUES
(
CURRENT_USER, NOW()
);
END IF;
RETURN NEW;
END;
$$
```

Triggery

```
CREATE TRIGGER history_log
BEFORE UPDATE
ON users
FOR EACH ROW
EXECUTE FUNCTION fn_trigger_history();
```

T

CREATE TRIGGER history_log
BEFORE UPDATE
ON users
FOR EACH ROW
EXECUTE FUNCTION fn_trigger_history();



The screenshot shows a PostgreSQL Query Editor interface. The query editor has a toolbar at the top with icons for saving, running, and other actions. Below the toolbar, the query editor shows a query: `select * from users`. The query is executed, and the results are displayed in a table. The table has two columns: `user_id` (integer) and `login` (character varying (50)). The results are as follows:

user_id	login
1	Jan
2	Jan
3	Juliusz
4	xyz
5	Anna
6	Anna
7	Pan Zenon

Zadanie

Stwórz:

1. 2 funkcje - wymagania:

a) przetwarzającą daną liczbową (np. przy cenie netto liczy cenę brutto)

b) przetwarzającą daną tekstową (np. konkatencja 3 kolumn w tabeli i zwrócenie wyniku)

Wykorzystaj funkcje w swojej aplikacji

2. Stwórz trigger, który umożliwi logowanie historii operacji dla 2 wybranych tabel z bazy. Poszczególne wpisy w tabeli historii mają zawierać:

informacje na temat użytkownika wykonującego modyfikację, stare wartości z tabeli, nowe wartości w tabeli, datę wykonania operacji.