Exemplos de TRIGGER

Uma função de gatilho (TRIGGER) pode ser criada para executar antes (BEFORE) ou após (AFTER) as consultas INSERT, UPDATE OU DELETE, uma vez para cada registro (linha) modificado ou por instrução SQL. Logo que ocorre um desses eventos do gatilho a função do gatilho é disparada automaticamente para tratar o evento.

A função de gatilho deve ser declarada como uma função que não recebe argumentos e que retorna o tipo TRIGGER. Após criar a função de gatilho, estabelecemos o gatilho pelo comando CREATE TRIGGER. Uma função de gatilho pode ser utilizada por vários gatilhos.

As funções de gatilho chamadas por gatilhos-por-instrução devem sempre retornar NULL.

As funções de gatilho chamadas por gatilhos-por-linha podem retornar uma linha da tabela (um valor do tipo HeapTuple) para o executor da chamada, se assim o decidirem.

Sintaxe:

**CREATE TRIGGER nome { BEFORE | AFTER } { evento [ OR ... ] }**

**ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]**

**EXECUTE PROCEDURE nome\_da\_função ( argumentos )**

O gatilho fica associado à tabela especificada e executa a função especificada nome\_da\_função quando determinados eventos ocorrerem.

O gatilho pode ser especificado para disparar antes de tentar realizar a operação na linha (antes das restrições serem verificadas e o comando INSERT, UPDATE ou DELETE ser tentado), ou após a operação estar completa (após as restrições serem verificadas e o INSERT, UPDATE ou DELETE ter completado).

evento Um entre INSERT, UPDATE ou DELETE; especifica o evento que dispara o gatilho. Vários eventos podem ser especificados utilizando OR.

Exemplos:

/\*----------------------------------------

Trigger I

-----------------------------------------\*/

**CREATE TABLE empregados(**

**codigo int4 NOT NULL,**

**nome VARCHAR,**

**salario int4,**

**departamento\_cod int4,**

**ultima\_data TIMESTAMP,**

**ultimo\_usuario VARCHAR(50),**

**CONSTRAINT empregados\_pkey PRIMARY KEY (codigo) )**

**CREATE or REPLACE FUNCTION empregados\_gatilho()**

**RETURNS TRIGGER AS $empregados\_gatilho$**

**BEGIN**

**-- Verificar se foi fornecido o nome e o salário do empregado**

**IF NEW.nome IS NULL THEN**

**RAISE EXCEPTION 'O nome do empregado não pode ser NULO';**

**END IF;**

**IF NEW.salario IS NULL THEN**

**RAISE EXCEPTION ' O salário de % NÃO pode ser NULO', NEW.nome;**

**END IF;**

**--**

**-- Quem paga para trabalhar?**

**--**

**IF NEW.salario <= 0 THEN**

**RAISE EXCEPTION 'O Salário de % deve ser MAIOR que ZERO', NEW.nome;**

**END IF;**

**--**

**--**

**-- Registrar quem alterou a folha de pagamento e quando**

**--**

**NEW.ultima\_data := 'now';**

**NEW.ultimo\_usuario := CURRENT\_USER;**

**RETURN NEW;**

**END;**

**$empregados\_gatilho$ LANGUAGE plpgsql;**

**CREATE TRIGGER empregados\_gatilho BEFORE INSERT OR UPDATE ON empregados**

**FOR EACH ROW EXECUTE PROCEDURE empregados\_gatilho();**

**INSERT INTO empregados (codigo,nome, salario) VALUES (5,'João',1000);**

**INSERT INTO empregados (codigo,nome, salario) VALUES (6,'José',1500);**

**INSERT INTO empregados (codigo,nome, salario) VALUES (7,'Maria',2500);**

**INSERT INTO empregados (codigo,nome, salario) VALUES (8,'Maria Josefa',-2500);**

**INSERT INTO empregados (codigo,nome, salario) VALUES (8,NULL,2500);**

**INSERT INTO empregados (codigo,nome, salario) VALUES (8,'Carlos',NULL);**

**SELECT \* FROM empregados;**

/\* NEW – Para INSERT e UPDATE OLD – Para DELETE \*/

/\*----------------------------------------

Trigger II

------------------------------------------\*/

**CREATE TABLE emp (**

**nome VARCHAR NOT NULL,**

**salario INTEGER**

**);**

**CREATE TABLE empregados\_audit(**

**operacao CHAR(1) NOT NULL,**

**usuario VARCHAR NOT NULL,**

**DATA TIMESTAMP NOT NULL,**

**nome VARCHAR NOT NULL,**

**salario INTEGER**

**);**

CREATE OR REPLACE FUNCTION processa\_emp\_audit() RETURNS TRIGGER AS $emp\_audit$

BEGIN

--

-- Cria uma linha na tabela emp\_audit para refletir a operação

-- realizada na tabela emp. Utiliza a variável especial TG\_OP

-- para descobrir a operação sendo realizada.

--

IF (TG\_OP = 'DELETE') THEN

INSERT INTO emp\_audit SELECT 'E', USER, now(), OLD.\*;

RETURN OLD;

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO emp\_audit SELECT 'A', USER, now(), NEW.\*;

RETURN NEW;

ELSIF (TG\_OP = 'INSERT') THEN

INSERT INTO emp\_audit SELECT 'I', USER, now(), NEW.\*;

RETURN NEW;

END IF;

RETURN NULL; -- o resultado é ignorado uma vez que este é um gatilho AFTER

END;

$emp\_audit$ LANGUAGE plpgsql;

CREATE TRIGGER emp\_audit

AFTER INSERT OR UPDATE OR DELETE ON emp

FOR EACH ROW EXECUTE PROCEDURE processa\_emp\_audit();

INSERT INTO emp (nome, salario) VALUES ('João',1000);

INSERT INTO emp (nome, salario) VALUES ('José',1500);

INSERT INTO emp (nome, salario) VALUES ('Maria',250);

UPDATE emp SET salario = 2500 WHERE nome = 'Maria';

DELETE FROM emp WHERE nome = 'João';

SELECT \* FROM emp;

SELECT \* FROM empregados\_audit;

Outro exemplo:

/\*--------------------------------------------------------

TRIGGER III

---------------------------------------------------------\*/

CREATE TABLE funcionarios (

codigo serial PRIMARY KEY,

nome VARCHAR NOT NULL,

salario INTEGER

);

CREATE TABLE funcionarios\_audit(

usuario VARCHAR NOT NULL,

DATA TIMESTAMP NOT NULL,

id INTEGER NOT NULL,

coluna text NOT NULL,

valor\_antigo text NOT NULL,

valor\_novo text NOT NULL

);

CREATE OR REPLACE FUNCTION processa\_func\_audit()

RETURNS TRIGGER AS $funcionarios\_audit$

BEGIN

--

-- Não permitir atualizar a chave primária

--

IF (NEW.codigo <> OLD.codigo) THEN

RAISE EXCEPTION 'Não é permitido atualizar o campo codigo';

END IF;

--

-- Inserir linhas na tabela funcionarios\_audit para refletir as alterações

-- realizada na tabela emp.

--

IF (NEW.nome <> OLD.nome) THEN

INSERT INTO funcionarios\_audit SELECT CURRENT\_USER, CURRENT\_TIMESTAMP,

NEW.codigo, 'nome', OLD.nome, NEW.nome;

END IF;

IF (NEW.salario <> OLD.salario) THEN

INSERT INTO funcionarios\_audit SELECT CURRENT\_USER, CURRENT\_TIMESTAMP,

NEW.codigo, 'salario', OLD.salario, NEW.salario;

END IF;

RETURN NULL; -- o resultado é ignorado uma vez que este é um gatilho AFTER

END;

$funcionarios\_audit$ LANGUAGE plpgsql;

--drop trigger funcionarios\_audit

CREATE TRIGGER funcionarios\_audit

AFTER UPDATE ON funcionarios

FOR EACH ROW EXECUTE PROCEDURE processa\_func\_audit();

INSERT INTO funcionarios (nome, salario) VALUES ('João',1000);

INSERT INTO funcionarios (nome, salario) VALUES ('José',1500);

INSERT INTO funcionarios (nome, salario) VALUES ('Maria',2500);

UPDATE funcionarios SET salario = 2500 WHERE codigo = 2;

UPDATE funcionarios SET nome = 'Maria Cecília' WHERE codigo = 3;

UPDATE funcionarios SET codigo=100 WHERE codigo=1;

--ERRO: Não é permitido atualizar o campo codigo

SELECT \* FROM funcionarios;

SELECT \* FROM funcionarios\_audit;