Gatilhos

O mecanismo de gatilho permite a execução de uma função antes ou depois de uma operação que altere o conteúdo de uma tabela ou visão, tanto uma vez para cada linha modificada quanto uma vez por instrução SQL.

Quando ocorre o evento do gatilho, a função de gatilho é chamada no momento apropriado para tratar o evento.

A função de gatilho deve ser definida antes do gatilho ser criado. Deve ser declarada como uma função que não recebe argumentos e que retorna o tipo TRIGGER. Os argumentos da função de gatilho são passados através de estruturas especiais e não na forma comum de argumentos de função.

Gatilhos

Gatilhos para tabelas são disparados antes ou depois da execução da operação que disparou o gatilho.

Na criação do gatilho deve ser especificado se o gatilho deve ser executado antes (BEFORE) ou depois (AFTER) da execução da operação.

Gatilhos

Existem dois tipos de gatilhos: gatilhos por linha e gatilhos por instrução. Em um gatilho por linha, a função é chamada uma vez para cada linha afetada pela instrução que disparou o gatilho. Em contraste, um gatilho por instrução é chamado somente uma vez quando a instrução apropriada é executada, a despeito do número de linhas afetadas pela instrução. Em particular, uma instrução que não afeta nenhuma linha ainda assim resulta na execução dos gatilhos por instrução aplicáveis. Este dois tipos de gatilho são algumas vezes chamados de "gatilhos no nível de linha" e "gatilhos no nível de instrução", respectivamente.

É possível criar gatilhos por linha para as operações de INSERT, UPDATE e DELETE.

Gatilhos por instrução podem ser criados para operações de INSERT, UPDATE, DELETE e TRUNCATE.

Gatilho para TRUNCATE não faz parte do padrão da linguagem SQL.

Gatilhos

Os gatilhos por instrução BEFORE disparam antes do início de qualquer operação causada pela instrução, enquanto os gatilhos por instrução AFTER disparam depois que todas as operações causadas pela instrução sejam executadas.

Os gatilhos por linha BEFORE disparam antes da operação em cada linha que seja afetada pela instrução, enquanto os gatilhos por linha AFTER disparam no fim da instrução, mas antes dos gatilhos por instrução AFTER.

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 para o executor da chamada.

Os gatilhos no nível-de-linha disparados antes de uma operação podem retornar **NULL** para cancelar a operação para a linha corrente. Isto instrui ao executor a não realizar a operação no nível-de-linha que chamou o gatilho sem gerar uma exceção.

Gatilhos no nível-de-linha disparados antes de uma operação de **INSERT** e **UPDATE** podem retornar uma linha da tabela para substituir a linha inserida ou atualizada. Isto permite à função de gatilho modificar os dados que serão armazenados. Para não interferir na operação, a função deve retornar a mesma linha que recebeu como argumento.

O valor de retorno é ignorado para gatilhos no nível-de-linha disparados após a operação.

Gatilhos

Se for definido mais de um gatilho para o mesmo evento na mesma relação, os gatilhos são disparados pela ordem alfabética de seus nomes. No padrão da linguagem SQL, os gatilhos são executados na ordem de criação.

No caso dos gatilhos para antes, a linha possivelmente modificada retornada por cada gatilho se torna a entrada do próximo gatilho. Se algum dos gatilhos para antes retornar NULL, a operação é abandonada e os gatilhos seguintes não são disparados.

CREATE TRIGGER

```
CREATE [ CONSTRAINT ] TRIGGER nome
   { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
   ON tabela
   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE |
INITIALLY DEFERRED } }
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condicao ) ]
   EXECUTE PROCEDURE nome_da_funcao ( argumentos )
```

A instrução **CREATE TRIGGER** cria um gatilho associado a tabela especificada que executa a função **nome_da_funcao** segundo os eventos especificados.

nome - nome do gatilho, deve ser distinto do nome de qualquer outro gatilho para a mesma tabela.

BEFORE | AFTER - determina se a função será chamada antes ou depois do evento. Gatilhos por linha para visões não podem utilizar **BEFORE** ou **AFTER**.

INSTEAD OF – utilizado para gatilhos por linha para visões.

CREATE TRIGGER

evento – especifica se o gatilho deve ser executado em uma operação de **INSERT**, **UPDATE**, **DELETE** ou **TRUNCATE** podem ser especificados vários eventos utilizando **OR**. O uso de **OR** para especificar vários eventos não é parte do padrão da linguagem SQL.

tabela - nome da tabela que o gatilho se destina.

FOR [EACH] { ROW | STATEMENT } - especifica se o gatilho deve ser disparado por linha ou por instrução, o padrão é **FOR EACH STATEMENT**.

nome_da_função – nome da função que será executada pelo gatilho.

argumentos - lista opcional de argumentos, separados por vírgula, a serem fornecidos para a função quando o gatilho for executado. Os argumentos são literais constantes que serão sempre convertidos em cadeias de caracteres.

CREATE TRIGGER

WHEN – especifica uma condição para a execução do gatilho, o gatilho será executado apenas se a condição retornar **TRUE**. Para gatilhos por linha, a condição pode referenciar a nova linha que será gravada por **NEW** (nos gatilhos para **INSERT** e **UPDATE**) e a antiga linha existente por **OLD** (nos gatilhos para **UPDATE** e **DELETE**). Gatilhos **INSTEAD OF** não suportam a cláusula **WHEN**.

CONSTRAINT – o uso dessa cláusula cria um gatilho constraint que é similar a um gatilho regular, porém o tempo de execução pode ser ajustado usando **SET CONSTRAINTS**, segundo a definição do tempo de execução do gatilho. Gatilhos constraints não são parte do padrão da linguagem SQL.

CREATE TRIGGER

É possível especificar gatilhos para UPDATE por coluna, que são executados apenas quando ao menos uma das colunas especificadas constar da lista de alvos do comando UPDATE. Para especificar gatilhos por coluna, deve ser especificado como evento:

```
UPDATE OF coluna [, ... ]
```

O gatilho por coluna não é executado se a coluna não consta da lista de alvos do comando UPDATE mas é alterada como consequência da execução de algum gatilho por linha BEFORE.

CREATE TRIGGER

Exemplo:

```
CREATE FUNCTION alunos_restantes() RETURNS TRIGGER AS $$
  DECLARE
    n INTEGER;
  BEGIN
    SELECT INTO n COUNT(*) FROM aluno;
    RAISE NOTICE 'Existem % alunos cadastrados', n;
    RETURN NULL;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER alunos_count AFTER INSERT OR DELETE
  ON aluno FOR EACH ROW
  EXECUTE PROCEDURE alunos_restantes();
```

CREATE TRIGGER

```
SELECT * FROM aluno;
 matricula |      nome        |    rg    | curso | serie | turma
-----------+------------------+----------+-------+-------+-------
         1 | Ana Lucia        | 20143531 | 0001  | 1     | A
         2 | Luis Claudio     | 22336362 | 0001  | 1     | A
         3 | Marcelo          | 25343256 | 0001  | 1     | A
         4 | Debora           | 20356328 | 0001  | 1     | B
         5 | Fernanda         | 26344325 | 0001  | 1     | B
         6 | Alvaro           | 21764527 | 0001  | 1     | B
         7 | Claudio          | 23336368 | 0002  | 1     | A
         8 | Andrea           | 28456474 | 0002  | 1     | A
         9 | Carla            | 23636731 | 0002  | 2     | A
        10 | Fernanda         | 29563735 |       |       |
DELETE FROM aluno WHERE curso='0002';
NOTICE:  Existem 7 alunos cadastrados
NOTICE:  Existem 7 alunos cadastrados
NOTICE:  Existem 7 alunos cadastrados
DELETE 3
```

DROP TRIGGER

DROP TRIGGER [IF EXISTS] nome ON tabela [CASCADE | RESTRICT]

A instrução DROP TRIGGER remove o gatilho nome associado a tabela.

CASCADE – remove os objetos que dependem do gatilho.

RESTRICT – não remove o gatilho se existir algum objeto que dependa do gatilho.

Exemplo:

```
DROP TRIGGER alunos_count ON aluno;
```

Variáveis para Gatilhos

Quando uma função em PL/pgSQL é disparada como gatilho, são criadas variáveis especiais para receber os argumentos do gatinho.

NEW - variável do tipo **RECORD** contendo a nova linha do banco de dados, para as operações de **INSERT/UPDATE** nos gatilhos no nível de linha. O valor desta variável é **NULL** nos gatilhos no nível de instrução.

OLD - variável do tipo **RECORD** contendo a antiga linha do banco de dados, para as operações de **UPDATE/DELETE** nos gatilhos no nível de linha. O valor desta variável é **NULL** nos gatilhos no nível de instrução.

TG_NAME - variável do tipo **NAME** contendo o nome do gatilho disparado.

TG_WHEN - variável do tipo **TEXT** contendo "**BEFORE**" ou "**AFTER**", dependendo da definição do gatilho.

TG_LEVEL - variável do tipo **TEXT** contendo "**ROW**" ou "**STATEMENT**", dependendo da definição do gatilho.

Variáveis para Gatilhos

TG_OP - variável do tipo TEXT contendo “INSERT”, “UPDATE”, ou “DELETE”, informando para qual operação o gatilho foi disparado.

TG_RELID - variável do tipo **OID** com **OID** de objeto da tabela que causou o disparo do gatilho.

TG_TABLE_NAME - variável do tipo **NAME** com o nome da tabela que causou o disparo do gatilho, anteriormente era utilizada a variável **TG_RELNAME**, essa variável ainda existe mas será descontinuada futuramente.

TG_TABLE_SCHEMA - variável do tipo **NAME** com o nme do schema da tabela que causou o disparo do gatilho.

TG_NARGS - variável inteira com o número de argumentos fornecidos ao procedimento de gatilho na instrução **CREATE TRIGGER**.

TG_ARGV[] - vetor de variáveis do tipo **TEXT** com os argumentos da instrução **CREATE TRIGGER**. O contador do índice começa por 0, índices inválidos (menor que 0 ou maior ou igual a **TG_NARGS)** resultam em um valor nulo.

Variáveis para Gatilhos

Exemplo:

```
ALTER TABLE funcionario ADD COLUMN contratacao DATE;

CREATE FUNCTION funcionario_contratacao() RETURNS TRIGGER
AS $$
  BEGIN
    IF NEW.contratacao IS NULL THEN
      NEW.contratacao=current_date;
    ELSE
      IF NEW.contratacao > current_date THEN
        RAISE NOTICE 'Data de contratacao invalida';
        RETURN NULL;
      END IF;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER contratacao BEFORE INSERT ON funcionario
FOR EACH ROW EXECUTE PROCEDURE funcionario_contratacao();
```

Variáveis para Gatilhos

```
INSERT INTO funcionario ( matricula, nome ) VALUES
( '42512', 'Julia' );
SELECT matricula, nome, contratacao FROM funcionario;
```

matricula	nome	contratacao
12342	Claudia	
45739	Andre	
34840	Marta	
38283	Sandro	
26360	Luis	
15730	Ana	
46373	Sandra	
36392	Marcos	
29847	Luana	
19463	Pedro	
28912	Nanci	
30472	Paula	
40528	Fabio	
22189	Andreia	
30352	Raquel	
48026	Luis	
42512	Julia	2013-10-22

Variáveis para Gatilhos

```
INSERT INTO funcionario ( matricula, nome, contratacao )
VALUES ( '25410', 'Pedro', '2016/11/01' );
NOTICE:  Data de contratacao invalida
INSERT 0 0
```

Gatilhos para Visões

Visões complexas não podem ser atualizadas, apenas visões mais simples que atendam as especificações correspondentes podem ser atualizadas.

Porém, é possível utilizar gatilhos para visões para simular a alteração de visões, independente da visão ser ou não uma visão atualizável.

Visões podem ter gatilhos por linha ou por instrução.

Gatilhos por instrução para visões pode ser especificados com AFTER ou BEFORE.

Gatilhos por linha para visões tem que ser especificados como INSTEAD OF e deve retornar NULL para indicar que não foi executada nenhuma alteração nas tabelas ou retornar uma linha com dados indicando que foram executadas as mudanças necessárias nas tabelas para simular a alteração visão.

Exercícios

1) Criar a tabela abaixo e criar um gatilho que registre nessa tabela todas as alterações realizadas na tabela “aluno”.

Tabela “alunolog”

```
  Column   |            Type
-----------+-----------------------------
 data      | timestamp without time zone
 usuario   | name
 operacao  | character(1)
 matricula | integer
 nome      | character(15)
```

2) Inclua a coluna “salario_maximo” na tabela “funcao”, com o mesmo tipo do campo “salario” da tabela “funcionario” e escreva o gatilho que não permita qualquer alteração na tabela “funcionario” que ultrapasse esse limite.

3) Inclua a coluna “salario_minimo” na tabela “funcao”, com o mesmo tipo do campo “salario” da tabela “funcionario” e escreva o gatilho que atribua esse mínimo em qualquer operação na tabela “funcionario” que atribua um salário menor que esse mínimo.

Exercícios

4) Criar um gatilho para as tabelas de entrada e saída que não permitam que o saldo do produto fique negativo após alguma operação nessas tabelas. Também não deve ser permitida a alteração do código do produto nos registros dessas tabelas.