CREATE TRIGGER

CREATE TRIGGER — define um novo gatilho

Sinopse

```
CREATE [ OR REPLACE ] [ CONSTRAINT ] TRIGGER nome { BEFORE | AFTER | INSTEAD OF } { e
    ON nome_da_tabela
    [ FROM nome_da_tabela_referenciada ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] nome_da_relação_de_transição } [ ...
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condição ) ]
    EXECUTE { FUNCTION | PROCEDURE } nome_da_função ( argumentos )

onde evento pode ser um entre:

    INSERT
    UPDATE [ OF nome_da_coluna [, ... ] ]
    DELETE
    TRUNCATE
```

Descrição

O comando `CREATE TRIGGER` define um novo gatilho. O comando `CREATE OR REPLACE TRIGGER` define um novo gatilho, ou substitui um gatilho existente [212] [213]. O gatilho será associado à tabela, visão ou tabela estrangeira especificada, executando a função especificada por `nome_da_função` quando determinadas operações forem executadas na tabela.

Para substituir a definição corrente de um gatilho existente, deve ser usado `CREATE OR REPLACE TRIGGER`, especificando o nome do gatilho existente e a tabela ancestral. Todas as outras propriedades são substituídas.

O gatilho pode ser especificado para disparar antes que a operação seja tentada em uma linha (antes que as restrições sejam verificadas, e o comando `INSERT`, `UPDATE` ou `DELETE` seja tentado); ou após a operação terminar (depois que as restrições forem verificadas, e o comando `INSERT`, `UPDATE` ou `DELETE` estiver concluído); ou em vez da operação (no caso de inserções, atualizações ou exclusões em uma visão). Se o gatilho disparar antes ou em vez do evento, o gatilho pode ignorar a operação da linha corrente, ou alterar a linha que está sendo inserida (apenas para as operações `INSERT` e `UPDATE`). Se o gatilho disparar após o evento, todas as alterações, incluindo os efeitos de outros gatilhos, estarão “visíveis” ao gatilho.

Um gatilho marcado como FOR EACH ROW é chamado uma vez para cada linha que a operação modifica. Por exemplo, um comando DELETE que afeta 10 linhas fará com que qualquer gatilho ON DELETE na relação de destino seja chamado 10 vezes separadamente, uma vez para cada linha excluída. Por outro lado, um gatilho marcado como FOR EACH STATEMENT é executado apenas uma vez para qualquer operação, independentemente de quantas linhas são modificadas (em particular, uma operação que modifica zero linhas ainda resultará na execução de qualquer gatilho FOR EACH STATEMENT aplicável).

Os gatilhos especificados para disparar INSTEAD OF o evento de gatilho devem ser marcados como FOR EACH ROW, e só podem ser definidos em visões. Os gatilhos BEFORE e AFTER em uma visão devem ser marcados como FOR EACH STATEMENT.

Além disso, podem ser definidos gatilhos para disparar no comando TRUNCATE, embora apenas para FOR EACH STATEMENT.

A tabela a seguir resume quais tipos de gatilhos podem ser usados em tabelas, visões e tabelas estrangeiras:

Tipo	Evento	Nível de linha	Nível de declaração
BEFORE	INSERT/UPDATE/DELETE	Tabelas e tabelas estrangeiras	Tabelas, visões e tabelas estrangeiras
	TRUNCATE	—	Tabelas
AFTER	INSERT/UPDATE/DELETE	Tabelas e tabelas estrangeiras	Tabelas, visões e tabelas estrangeiras
	TRUNCATE	—	Tabelas
INSTEAD OF	INSERT/UPDATE/DELETE	Visões	—
	TRUNCATE	—	—

Além disso, a definição de gatilho pode especificar uma condição booleana WHEN, que será testada para ver se o gatilho deve ser disparado. Em gatilhos no nível de linha, a condição WHEN pode examinar os valores antigos e/ou novos das colunas da linha. Os gatilhos no nível de instrução também podem ter condições WHEN, embora o recurso não seja tão útil para eles, porque a condição não pode se referir a nenhum valor na tabela.

Se forem definidos vários gatilhos do mesmo tipo para o mesmo evento, eles serão disparados na ordem alfabética de seus nomes.

Quando é especificada a opção CONSTRAINT, o comando cria um *gatilho de restrição*. É o mesmo que um gatilho normal, exceto que o momento de disparo do gatilho pode ser ajustado usando SET CONSTRAINTS. Os gatilhos de restrição devem ser gatilhos AFTER ROW em tabelas simples (e não em tabelas estrangeiras). Eles podem ser disparados no final da instrução que causa o evento de disparo, ou no final da transação que os contém; nesse último caso, são ditos *postergados*. Um disparo de gatilho postergado pendente também pode ser forçado a ocorrer imediatamente usando SET CONSTRAINTS. Se espera que os gatilhos de restrição gerem uma exceção quando as restrições que eles implementam são violadas.

A opção REFERENCING ativa a coleta de *relações de transição*, que são conjuntos de linhas que incluem todas as linhas inseridas, excluídas ou modificadas pela instrução SQL corrente. Esse recurso permite que o gatilho tenha uma visão global do que a instrução fez, e não apenas uma linha por vez. Essa opção só é permitida para gatilhos AFTER, que não sejam gatilhos de restrição; além disso, se o gatilho for um gatilho para UPDATE, não deverá especificar uma lista de nome_da_coluna. A cláusula OLD TABLE só pode ser especificado uma vez, e apenas para um gatilho que pode disparar para UPDATE ou DELETE; ela cria uma relação de transição contendo as

imagens anteriores de todas as linhas atualizadas ou excluídas pela instrução. De forma semelhante, a cláusula `NEW TABLE` só pode ser especificado uma vez, e apenas para um gatilho que pode disparar para `UPDATE` ou `INSERT`; ela cria uma relação de transição contendo as *imagens posteriores* de todas as linhas atualizadas ou inseridas pela instrução.

O comando `SELECT` não modifica nenhuma linha, então não se pode criar gatilhos para `SELECT`. As regras e visões podem fornecer soluções viáveis para problemas que parecem necessitar de gatilhos para `SELECT`.

Veja Gatilhos para obter mais informações sobre gatilhos.

Parâmetros

`nome`

O nome a ser dado ao novo gatilho. Deve ser diferente do nome de qualquer outro gatilho para a mesma tabela. O nome não pode ser qualificado pelo esquema — o gatilho herda o esquema de sua tabela. Para um gatilho de restrição, esse também é o nome a ser usado ao modificar o comportamento do gatilho usando o comando `SET CONSTRAINTS`.

`BEFORE`
`AFTER`
`INSTEAD OF`

Determina se a função é chamada antes, depois ou em vez do evento. Um gatilho de restrição só pode ser especificado como `AFTER`.

`evento`

Um entre `INSERT`, `UPDATE`, `DELETE`, ou `TRUNCATE`; especifica o evento que irá disparar o gatilho. Podem ser especificados vários eventos usando `OR`, exceto quando são solicitadas as relações de transição.

Para eventos `UPDATE`, é possível especificar uma lista de colunas usando a seguinte sintaxe:

```
UPDATE OF nome_da_coluna1 [, nome_da_coluna2 ... ]
```

O gatilho só será disparado se pelo menos uma das colunas listadas for mencionada como destino do comando `UPDATE`, ou se uma das colunas listadas for uma coluna gerada que dependa de uma coluna que seja o destino do comando `UPDATE`.

Os eventos `INSTEAD OF UPDATE` não aceitam uma lista de colunas. A lista de colunas também não pode ser especificada ao solicitar relações de transição.

`nome_da_tabela`

O nome (opcionalmente qualificado pelo esquema) da tabela, visão ou tabela estrangeira para a qual o gatilho se destina.

`nome_da_tabela_referenciada`

O nome (possivelmente qualificado pelo esquema) da outra tabela referenciada pela restrição. Essa opção é usada para restrições de chave estrangeira, não sendo recomendada para uso geral. Só pode ser especificado para gatilhos de restrição.

`DEFERRABLE`
`NOT DEFERRABLE`
`INITIALLY IMMEDIATE`
`INITIALLY DEFERRED`

O momento padrão do gatilho. Veja a documentação do comando CREATE TABLE para obter detalhes sobre essas opções de restrição. Só pode ser especificado para gatilhos de restrição.

`REFERENCING`

Essa palavra-chave precede imediatamente a declaração de um ou dois nomes de relação que fornecem acesso às relações de transição da instrução de disparo.

`OLD TABLE`
`NEW TABLE`

Essas cláusulas indicam se o nome da relação que as segue se refere a uma imagem de relação de transição anterior, ou a uma imagem de relação de transição posterior.

`nome_da_relação_de_transição`

O nome (não qualificado) a ser usado no gatilho para essa relação de transição.

`FOR EACH ROW`
`FOR EACH STATEMENT`

Especifica se a função de gatilho deve ser disparada uma vez para cada linha afetada pelo evento do gatilho, ou apenas uma vez por instrução SQL. Se não for especificada nenhuma dessas duas cláusulas, o padrão é `FOR EACH STATEMENT`. Os gatilhos de restrição só podem ser especificados como `FOR EACH ROW`.

`condição`

Uma expressão booleana que determina se a função de gatilho será realmente executada. Se for especificada a condição `WHEN`, a função só será chamada se a `condição` retornar `true`. Nos gatilhos `FOR EACH ROW`, a condição `WHEN` pode se referir a valores antigos e/ou novos das colunas da linha, escrevendo `OLD.nome_da_coluna` ou `NEW.nome_da_coluna`, respectivamente. Obviamente, os gatilhos `INSERT` não podem se referir a `OLD`, e os gatilhos `DELETE` não podem se referir a `NEW`.

Os gatilhos `INSTEAD OF` não dão suporte a condições `WHEN`.

No momento, as expressões `WHEN` não podem conter subconsultas.

Note que, para os gatilhos de restrição, a avaliação da condição `WHEN` não é postergada, ocorrendo imediatamente após a execução da operação de atualização de linha. Se a condição não for avaliada como verdade, o gatilho não será colocado na fila para execução postergada.

`nome_da_função`

Uma função definida pelo usuário declarada como sem argumentos, e retornando o tipo `trigger`, executada quando o gatilho é disparado.

Na sintaxe de `CREATE TRIGGER`, as palavras-chave `FUNCTION` e `PROCEDURE` são equivalentes, mas a função referenciada deve, em qualquer caso, ser uma função, não um procedimento. O uso da palavra-chave `PROCEDURE` aqui é histórico em obsolescência.

`argumentos`

Uma lista opcional de argumentos separados por vírgulas, a serem fornecidos à função quando o gatilho é executado. Os argumentos são constantes literal cadeia de caracteres. Nomes simples e constantes numéricas também podem ser escritos aqui, mas todos serão convertidos em cadeias de caracteres. Por favor, verifique a descrição da linguagem de implementação da função de gatilho para saber como esses argumentos podem ser acessados dentro da função; pode ser diferente dos argumentos normais de função.

Notas

Para criar ou substituir um gatilho em uma tabela, é necessário ter o privilégio `TRIGGER` na tabela. Também é necessário ter o privilégio `EXECUTE` na função de gatilho.

O comando DROP TRIGGER remove um gatilho.

A criação de um gatilho no nível de linha em uma tabela particionada fará com que um gatilho "clone" idêntico seja criado em cada uma de suas partições existentes; e quaisquer partições criadas ou anexadas posteriormente também terão um gatilho idêntico. Se já houver um gatilho com nome conflitante em uma partição filha, ocorrerá um erro, a menos que seja usado o comando `CREATE OR REPLACE TRIGGER`, caso em que esse gatilho é substituído por um gatilho clone. Quando uma partição é desanexada de sua mãe, seus gatilhos clonados são removidos.

Um gatilho específico da coluna (definido usando a sintaxe `UPDATE OF nome_da_coluna`) irá disparar quando qualquer uma de suas colunas estiver listada como destino na lista `SET` do comando `UPDATE`. É possível que o valor de uma coluna mude mesmo que o gatilho não seja disparado, porque as alterações feitas no conteúdo da linha pelos gatilhos `BEFORE UPDATE` não são consideradas. Por outro lado, um comando como `UPDATE ... SET x = x ...` irá disparar um gatilho na coluna `x`, mesmo que o valor da coluna não seja alterado.

Em um gatilho `BEFORE`, a condição `WHEN` é avaliada logo antes da função ser (ou seria) executada, portanto, usar `WHEN` não é materialmente diferente de testar a mesma condição no início da função de gatilho. Note, em particular, que a linha `NEW` vista pela condição é o valor corrente, possivelmente modificado por gatilhos anteriores. Além disso, a condição `WHEN` de um gatilho `BEFORE` não tem permissão para examinar as colunas do sistema da linha `NEW` (tal como `ctid`), porque ainda não foram definidas.

Em um gatilho `AFTER`, a condição `WHEN` é avaliada logo após a ocorrência da atualização da linha, determinando se será enfileirado um evento para disparar o gatilho no final da instrução. Portanto, quando a condição `WHEN` de um gatilho `AFTER` não retorna `true`, não é necessário enfileirar um evento, nem buscar novamente a linha no final da instrução. Isso pode resultar em acelerações significativas em instruções que modificam muitas linhas, se o gatilho precisar ser disparado apenas para algumas das linhas.

Em alguns casos, é possível que um único comando SQL dispare mais de um tipo de gatilho. Por exemplo, um comando `INSERT` com a cláusula `ON CONFLICT DO UPDATE` pode causar operações de inserção e atualização, portanto, irá disparar os dois tipos de gatilhos conforme necessário. As relações de transição fornecidas aos gatilhos são específicas para seu tipo de evento; portanto, um gatilho `INSERT` verá apenas as linhas inseridas, enquanto um gatilho `UPDATE` verá apenas as linhas atualizadas.

Atualizações ou exclusões de linha causadas por ações de imposição de chave estrangeira, tais como `ON UPDATE CASCADE` ou `ON DELETE SET NULL`, são tratadas como parte do comando SQL que as causou (note-se que essas ações nunca são postergadas). Os gatilhos relevantes na tabela afetada serão disparados, de modo que isso forneça outra maneira pela qual um comando SQL pode disparar gatilhos que não correspondam diretamente ao seu tipo. Em casos simples, os gatilhos que solicitam relações de transição vão ver todas as alterações causadas em sua tabela por um único comando SQL original como uma única relação de transição. Entretanto, há

casos em que a presença de um gatilho `AFTER ROW`, que solicita relações de transição, fará com que as ações de imposição de chave estrangeira disparadas por um único comando SQL sejam divididas em várias etapas, cada uma com sua própria relação(ões) de transição. Nesses casos, quaisquer gatilhos no nível de instrução presentes serão disparados uma vez por criação de um conjunto de relações de transição, garantindo que os gatilhos vejam cada linha afetada em uma relação de transição uma vez, e apenas uma vez.

Os gatilhos de uma visão, no nível de instrução, são disparados apenas se a ação na visão for tratada por um gatilho `INSTEAD OF` no nível de linha. Se a ação for tratada por uma regra `INSTEAD`, quaisquer instruções emitidas pela regra serão executadas no lugar da instrução original especificando a visão, de modo que os gatilhos que serão disparados sejam aqueles nas tabelas especificadas nas declarações de substituição. Da mesma forma, se a visão for automaticamente atualizável, a ação será tratada reescrevendo automaticamente a instrução em uma ação na tabela base da visão, de modo que os gatilhos no nível da instrução da tabela base sejam os disparados.

Modificar uma tabela particionada, ou uma tabela com filhas de herança, dispara os gatilhos no nível de instrução anexados à tabela especificada explicitamente, mas não os gatilhos no nível de instrução para suas partições ou tabelas filhas. Por outro lado, os gatilhos no nível de linha são disparados nas linhas em partições afetadas ou tabelas filhas, mesmo que não sejam explicitamente especificados na consulta. Se um gatilho no nível de instrução tiver sido definido com relações de transição especificadas pela cláusula `REFERENCING`, as imagens de antes e depois das linhas serão visíveis em todas as partições afetadas ou tabelas filhas. No caso de filhos de herança, as imagens de linha incluem apenas as colunas que estão presentes na tabela à qual o gatilho está anexado.

No momento, os gatilhos no nível de linha com relações de transição não podem ser definidos em partições ou tabelas filhas de herança. Além disso, gatilhos em tabelas particionadas não podem ser `INSTEAD OF`.

No momento, a opção `OR REPLACE` não tem suporte para gatilhos de restrição.

A substituição de um gatilho existente em uma transação que já realizou ações de atualização na tabela do gatilho não é recomendada. As decisões de disparo de gatilho, ou partes de decisões de disparo, que já foram feitas, não serão reconsideradas, então os efeitos poderão ser surpreendentes.

Existem algumas funções de gatilho nativas que podem ser usadas para resolver problemas comuns, sem ter que escrever seu próprio código de gatilho; veja Funções de gatilho.

Exemplos

Executar a função `check_account_update` sempre que uma linha da tabela `accounts` estiver prestes a ser atualizada:

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
    FOR EACH ROW
    EXECUTE FUNCTION check_account_update();
```

Modificar essa definição de gatilho para executar a função apenas se a coluna `balance` for especificada como destino no comando `UPDATE`:

```
CREATE OR REPLACE TRIGGER check_update
    BEFORE UPDATE OF balance ON accounts
    FOR EACH ROW
    EXECUTE FUNCTION check_account_update();
```

A forma a seguir só executa a função, se a coluna `balance` tiver de fato mudado de valor:

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
    FOR EACH ROW
    WHEN (OLD.balance IS DISTINCT FROM NEW.balance)
    EXECUTE FUNCTION check_account_update();
```

Chamar uma função para registrar as atualizações na tabela `accounts`, mas somente se algo mudou:

```
CREATE TRIGGER log_update
    AFTER UPDATE ON accounts
    FOR EACH ROW
    WHEN (OLD.* IS DISTINCT FROM NEW.*)
    EXECUTE FUNCTION log_account_update();
```

Executar a função `view_insert_row` para cada linha, para inserir linhas nas tabelas subjacentes a uma visão:

```
CREATE TRIGGER view_insert
    INSTEAD OF INSERT ON my_view
    FOR EACH ROW
    EXECUTE FUNCTION view_insert_row();
```

Executar a função `check_transfer_balances_to_zero` para cada instrução, para confirmar que os saldos das transferências sejam zero.

```
CREATE TRIGGER transfer_insert
    AFTER INSERT ON transfer
    REFERENCING NEW TABLE AS inserted
    FOR EACH STATEMENT
    EXECUTE FUNCTION check_transfer_balances_to_zero();
```

Executar a função `check_matching_pairs` para cada linha, para confirmar que as alterações são feitas nos pares correspondentes ao mesmo tempo (pela mesma instrução):

```
CREATE TRIGGER paired_items_update
    AFTER UPDATE ON paired_items
    REFERENCING NEW TABLE AS newtab OLD TABLE AS oldtab
    FOR EACH ROW
    EXECUTE FUNCTION check_matching_pairs();
```

O Exemplo completo de gatilho contém um exemplo completo de uma função de gatilho escrita em C.

Compatibilidade

O comando `CREATE TRIGGER` no PostgreSQL implementa um subconjunto do padrão SQL. As seguintes funcionalidades estão ausentes no momento:

- Enquanto os nomes das tabelas de transição para os gatilhos `AFTER` são especificados usando a cláusula `REFERENCING` da maneira padrão, as variáveis de linha usadas nos gatilhos `FOR EACH ROW` não podem ser especificadas na cláusula `REFERENCING`. Eles estão disponíveis de uma maneira que depende da linguagem na qual a função de gatilho é

escrita, mas é fixo para qualquer linguagem. Algumas linguagens efetivamente se comportam como se houvesse uma cláusula `REFERENCING` contendo `OLD ROW AS OLD NEW ROW AS NEW`.

- O padrão permite que as tabelas de transição sejam usadas com gatilhos `UPDATE` específicos da coluna, mas o conjunto de linhas que devem ser visíveis nas tabelas de transição depende da lista de colunas do gatilho. Isso não está implementado atualmente no PostgreSQL.

- O PostgreSQL permite apenas a execução de uma função definida pelo usuário para a ação disparada. O padrão permite a execução de vários outros comandos SQL, como `CREATE TABLE`, como a ação disparada. Essa limitação não é difícil de ser contornada criando uma função definida pelo usuário que executa os comandos desejados.

O padrão SQL especifica que os vários gatilhos devem ser acionados na ordem de momento da criação. O PostgreSQL usa a ordem de nome, que foi julgada ser mais conveniente.

O padrão SQL especifica que os gatilhos `BEFORE DELETE` em exclusões em cascata disparam *após* o comando `DELETE` em cascata estar concluído. O comportamento do PostgreSQL é para `BEFORE DELETE` disparar sempre antes da ação de exclusão, mesmo que seja em cascata. Isso é considerado mais consistente. Também haverá um comportamento fora do padrão se os gatilhos `BEFORE` modificarem linhas, ou impedirem atualizações durante uma atualização causada por uma ação referencial. Isso pode levar a violações de restrição ou dados armazenados que não respeitam a restrição referencial.

A capacidade de especificar várias ações para um único gatilho usando `OR` é uma extensão do PostgreSQL ao padrão SQL.

A capacidade de disparar gatilhos para o comando `TRUNCATE` é uma extensão do PostgreSQL do padrão SQL, assim como a capacidade de definir gatilhos no nível de instrução em visões.

O comando `CREATE CONSTRAINT TRIGGER` é uma extensão do PostgreSQL do padrão SQL. Assim como a opção `OR REPLACE`.

Veja também

ALTER TRIGGER, DROP TRIGGER, CREATE FUNCTION, SET CONSTRAINTS

---

[212] O comando `CREATE TRIGGER` define um gatilho no banco de dados. Os gatilhos podem ser criados para dar suporte a formas gerais de integridade ou regras de negócios. O gatilho define um conjunto de ações que são executadas com, ou disparadas por, um comando `INSERT`, `UPDATE` ou `DELETE`. IBM DB2 11.5 – CREATE TRIGGER statement (N. T.)

[213] Um gatilho no nível de instrução é especificado usando `FOR EACH STATEMENT`, enquanto um gatilho no nível de linha é especificado usando `FOR EACH ROW`. Um gatilho não pode ser tanto um gatilho no nível de instrução quanto um gatilho de nível de linha. A *ordem de execução* de um conjunto de gatilhos é crescente pelo valor de seu registro de data e hora de criação em seus descritores, de modo que o gatilho mais antigo seja executado primeiro. ISO/IEC 9075-2:1999 (E) – 4.35 Triggers (N. T.)

Anterior: CREATE TRANSFORM | Acima: Principal | Próxima: CREATE TYPE