Gatilhos em PostgreSQL

Banco de Dados II
Professor Fabiano Baldo

Sumário

- Introdução
- Gatilhos
  - Definição
  - Criação de gatilhos
- Funções
  - Diferenças
  - Variáveis especiais
  - Retorno
- Exemplos
- Extra
  - Tratamento de erros

2

Introdução | O que são gatilhos

- Um gatilho (em inglês, *trigger*) é uma especificação da ação que o banco de dados deve executar toda vez que determinado evento ocorrer
  - Ação = função
  - evento = executar certo tipo de operação
- É possível criar gatilhos usando diversas linguagens, entre elas, PL/pgSQL
- Não é possível criar gatilhos usando apenas SQL

3

Gatilhos | Definição

Tipos de gatilho

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

4

Gatilhos | Definição

Tipos de gatilho

Gatilhos podem disparar para modificações em TABELAS ou VIEWS...

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

5

Gatilhos | Definição

Tipos de gatilho

...ANTES, DEPOIS ou AO INVÉS...

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

6

Gatilhos | Definição

Tipos de gatilho

...dos comandos INSERT, UPDATE, DELETE ou TRUNCATE.

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

7

Gatilhos | Definição

Tipos de gatilho

Um gatilho pode ser configurado para que sua função seja executada uma vez por linha modificada (EACH ROW) ou uma vez por comando SQL (STATEMENT).

When	Event	Row-level	Statement-level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	—	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	—
	TRUNCATE	—	—

8

Gatilhos | Definição
Quando a função é executada

- Ordem de execução em tabelas:
  1. Gatilhos BEFORE STATEMENT são executados em ordem alfabética
  2. Mecanismo de execução verifica quais linhas serão modificadas pelo comando SQL
  3. Para cada linha que será modificada (ordem aleatória):
     i. Gatilhos BEFORE EACH ROW são executados em ordem alfabética
     ii. Comando SQL é executado
  4. Para cada linha que foi modificada
     i. Gatilhos AFTER EACH ROW são executados em ordem alfabética
  5. Gatilhos AFTER STATEMENT são executados em ordem alfabética

9

Gatilhos | Definição
Quando a função é executada

- Ordem de execução em views:
  1. Gatilhos BEFORE STATEMENT são executados em ordem alfabética
  2. Mecanismo de execução verifica quais linhas serão modificadas pelo comando SQL
  3. Para cada linha que será modificada (ordem aleatória):
     i. Gatilhos INSTEAD OF são executados em ordem alfabética
  4. Gatilhos AFTER STATEMENT são executados em ordem alfabética

10

Gatilhos | Definição
Quando a função é executada

- Ordem de execução em views:
  1. Gatilhos BEFORE STATEMENT são executados em ordem alfabética
  2. Mecanismo de execução verifica quais linhas serão modificadas pelo comando SQL
  3. Para cada linha que será modificada (ordem aleatória):
     i. Gatilhos INSTEAD OF são executados em ordem alfabética
  4. Gatilhos AFTER STATEMENT são executados em ordem alfabética

Gatilho INSTEAD OF descarta a operação que o disparou. Para que os dados sejam modificados, a FUNÇÃO é quem deve executar o comando SQL.

11

Gatilhos | Definição
Quando a função é executada

- Ordem de execução em views:
  1. Gatilhos BEFORE STATEMENT são executados em ordem alfabética
  2. Mecanismo de execução verifica quais linhas serão modificadas pelo comando SQL
  3. Para cada linha que será modificada (ordem aleatória):
     i. Gatilhos INSTEAD OF são executados em ordem alfabética
  4. Gatilhos AFTER STATEMENT são executados em ordem alfabética

Só existe no PostgreSQL versão 9.1 ou superior!

12

Gatilhos | Definição
Visibilidade dos dados alterados

- A visibilidade das alterações depende do nível do evento e do momento de seu disparo:
  - Se o gatilho for BEFORE STATEMENT
    - O SQL ainda não foi executado, portanto nenhuma mudança é visível
  - Se o gatilho for AFTER STATEMENT
    - O SQL já foi executado, portanto todas as mudanças são visíveis
  - Se o gatilho for BEFORE EACH ROW ou INSTEAD OF
    - A modificação da linha que causou a execução da função não é visível; a modificação de todas as linhas anteriores é visível
      - Atenção: Ao executar um comando SQL, as linhas são modificadas em ordem aleatória
  - Se o gatilho for AFTER EACH ROW
    - É executado uma vez para cada linha mas só depois de o comando SQL já ter executado em todas as linhas, portanto todas as mudanças são visíveis

13

Gatilhos | Definição
Observações

- Se um gatilho altera os dados de entrada, o gatilho seguinte receberá os dados alterados, e não os originais
- Se um gatilho executa comandos SQL, esses comandos podem disparar gatilhos (“gatilhos em cascata“)
  - Não existe uma quantidade máxima de níveis da cascata
  - É possível que uma cascata dispare novamente um gatilho, causando uma invocação recursiva
    - e.g., um gatilho INSERT pode executar um comando SQL que insere linhas adicionais na mesma tabela, fazendo com que o mesmo gatilho INSERT seja disparado novamente
    - É reponsabilidade do programador evitar recursões infinitas!

14

Gatilhos | Criação de gatilhos
Estrutura do comando

- Sintaxe do comando:

```
CREATE TRIGGER name
{ BEFORE | AFTER } { event [ OR ... ] } ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )
```

15

Gatilhos | Criação de gatilhos
Estrutura do comando

- Sintaxe do comando:

```
CREATE TRIGGER name
{ BEFORE | AFTER } { event [ OR ... ] } ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )
```

INSERT
UPDATE
DELETE
INSERT OR UPDATE
.....

16

Gatilhos | Criação de gatilhos
Estrutura do comando

- Sintaxe do comando:

```
CREATE TRIGGER name
{ BEFORE | AFTER } { event [ OR ... ] } ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )
```

Ou nome da view

17

Gatilhos | Criação de gatilhos
Estrutura do comando

- Sintaxe do comando:

```
CREATE TRIGGER name
{ BEFORE | AFTER } { event [ OR ... ] } ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )
```

Vários gatilhos podem ser configurados com a mesma função!

18

Gatilhos | Criação de gatilhos
Estrutura do comando

- Sintaxe do comando:

```
CREATE TRIGGER name
{ BEFORE | AFTER } { event [ OR ... ] } ON table
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )
```

Primeiro criar função, depois criar gatilho!

19

Gatilhos | Criação de gatilhos
Extras

- Renomear gatilho:

```
ALTER TRIGGER name ON table RENAME TO new_name
```

- Apagar gatilho:

```
DROP TRIGGER [ IF EXISTS ] name ON table [ CASCADE | RESTRICT ]
```

20

Gatilhos | Criação de gatilhos
Extras

- Renomear gatilho:

ALTER TRIGGER name ON table RENAME TO new_name

- Apagar gatilho:

DROP TRIGGER [IF EXISTS] name ON table [CASCADE | RESTRICT]

Para que não ocorra erro caso o gatilho não exista.

21

Gatilhos | Criação de gatilhos
Extras

- Renomear gatilho:

ALTER TRIGGER name ON table RENAME TO new_name

- Apagar gatilho:

DROP TRIGGER [IF EXISTS] name ON table [CASCADE | RESTRICT]

CASCADE: Também remove objetos que dependam do gatilho
RESTRICT: Só remove gatilho se nenhum objeto depender dele (padrão)

22

Sumário

- Introdução
- Gatilhos
  - Definição
  - Criação de gatilhos
- **Funções**
  - **Diferenças**
  - Variáveis especiais
  - Retorno
- Exemplos
- Extra
  - Tratamento de erros

Funções | Diferenças

- Algumas diferenças na criação de funções para uso em gatilhos:
  - A função deve ser declarada sem parâmetros
    - Dados de entrada serão enviados através de uma estrutura de dados especial
  - O retorno da função deve ser do tipo “trigger”

```
CREATE FUNCTION primeira_funcao() RETURNS trigger AS
$$
BEGIN
        RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Funções | Diferenças

- Algumas diferenças na criação de funções para uso em gatilhos:
  - A função deve ser declarada sem parâmetros
    - Dados de entrada serão enviados através de uma estrutura de dados especial
  - O retorno da função deve ser do tipo “trigger”

```
CREATE FUNCTION primeira_funcao() RETURNS trigger AS
$$
BEGIN
        RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

25

Sumário

- Introdução
- Gatilhos
  - Definição
  - Criação de gatilhos
- Funções
  - Diferenças
  - **Variáveis especiais**
  - Retorno
- Exemplos
- Extra
  - Tratamento de erros

26

Funções | Variáveis especiais

Introdução

- Quando uma função PL/pgSQL é chamada como um gatilho, várias variáveis especiais são criadas automaticamente no bloco mais externo:

27

Funções | Variáveis especiais

Variáveis NEW e OLD

- NEW
  - Tipo de dado: RECORD
  - Funcionamento em gatilhos tipo EACH ROW:
    - armazena a NOVA tupla da tabela para operações de INSERT e UPDATE
  - Funcionamento em gatilhos tipo STATEMENT:
    - recebe NULL
- OLD
  - Tipo de dado: RECORD
  - Funcionamento em gatilhos tipo EACH ROW:
    - armazena a tupla ANTIGA da tabela para operações de UPDATE e DELETE
  - Funcionamento em gatilhos tipo STATEMENT:
    - recebe NULL

28

Funções | Variáveis especiais

Variáveis com informações sobre o gatilho

- TG_NAME
  - Tipo de dado: NAME
  - Contém o nome do gatilho disparado
- TG_WHEN
  - Tipo de dado: TEXT
  - Contém quando o gatilho executou: “BEFORE” ou “AFTER”
- TG_LEVEL
  - Tipo de dado: TEXT
  - Contém o tipo do gatilho: “ROW” ou “STATEMENT”
- TG_OP
  - Tipo de dado: TEXT
  - Contém a operação: “INSERT”, “UPDATE” ou “DELETE”

29

Funções | Variáveis especiais

Variáveis com informações sobre o gatilho

- TG_RELID
  - Tipo de dado: OID
  - Contém o OID da tabela que disparou o gatilho
- TG_TABLE_NAME
  - Tipo de dado: NAME
  - Contém o nome da tabela que disparou o gatilho
- TG_TABLE_SCHEMA
  - Tipo de dado: NAME
  - Contém o nome do esquema da tabela que disparou o gatilho

30

Funções | Variáveis especiais

Variáveis para parâmetros

- TG_NARGS
  - Tipo de dado: INTEGER
  - Contém o número de parâmetros que devem ser passados para a função, conforme declarado no comando CREATE TRIGGER
- TG_ARGV[]
  - Tipo de dado: vetor de TEXT
  - Contém os parâmetros passados para a função, conforme declarado no comando CREATE TRIGGER. Começa da posição 0.

31

Sumário

32

Funções | Retorno
Alterar comando SQL

- Se o gatilho for do tipo BEFORE EACH ROW:
  - Pode retornar NULL para ignorar a operação na linha atual
    - A requisição do comando SQL é ignorado e a linha não é alterada
    - Nenhum gatilho subsequente será disparado PARA ESTA LINHA
  - Pode retornar uma linha da tabela (válido apenas em operações INSERT e UPDATE)
    - Ao invés de alterar a linha com os valores enviados pelo comando SQL, irá alterá-la com os dados retornados pela função
    - Mais comum: NEW
    - Pode-se modificar e retornar NEW ou outra variável de mesmo tipo
- Se o gatilho for do tipo AFTER EACH ROW:
  - O retorno será ignorado, recomenda-se retornar NULL
- Se o gatilho for do tipo STATEMENT:
  - Deve retornar NULL

33

Funções | Retorno
Alterar comando SQL

- Se o gatilho for do tipo INSTEAD OF:
  - Pode retornar NULL para informar que nenhum dado foi modificado
  - Pode retornar qualquer coisa diferente de NULL para informar que a linha foi alterada
    - Isso faz com que o contador de linhas alteradas (GET DIAGNOSTICS ROW_COUNT) seja incrementado

34

Sumário

- Introdução
- Gatilhos
  - Definição
  - Criação de gatilhos
- Funções
  - Diferenças
  - Variáveis especiais
  - Retorno
- **Exemplos**
- Extra
  - Tratamento de erros

35

Exemplos | Validação de dados

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL OR NEW.salary < 0 THEN
        RAISE EXCEPTION ‘salary cannot be null or < 0';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

36

Exemplos | Validação de dados

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

Sem parâmetros e retorno tipo "trigger"

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL OR NEW.salary < 0 THEN
        RAISE EXCEPTION 'salary cannot be null or < 0';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

37

Exemplos | Validação de dados

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

Exemplo de acesso às informações da variável NEW

Variável NÃO é declarada pelo programador!

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL OR NEW.salary < 0 THEN
        RAISE EXCEPTION 'salary cannot be null or < 0';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

38

Exemplos | Validação de dados

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

Exemplo de alteração das informações da variável NEW

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL OR NEW.salary < 0 THEN
        RAISE EXCEPTION ‘salary cannot be null or < 0';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

39

Exemplos | Validação de dados

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

Retornará o NEW modificado

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL OR NEW.salary < 0 THEN
        RAISE EXCEPTION ‘salary cannot be null or < 0';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

40

Exemplos | Validação de dados

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

PALAVRAS OBRIGATÓRIAS

QUANDO

OPERAÇÃO

TIPO

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL OR NEW.salary < 0 THEN
        RAISE EXCEPTION ‘salary cannot be null or < 0';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

41

Exemplos | Inserir dados em tabela

```
CREATE TABLE emp (
    empname text,
    salary integer
);
CREATE TABLE empHist(
    operation char(1),
    stamp timestamp,
    userid text,
    empname text,
    salary integer
);
```

```
CREATE FUNCTION procEmpHist() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'DELETE' THEN
        INSERT INTO empHist SELECT 'D', now(), user, OLD.*;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO empHist SELECT 'U', now(), user, NEW.*;
    ELSIF TG_OP = 'INSERT' THEN
        INSERT INTO empHist SELECT 'I', now(), user, NEW.*;
    END IF;
    RETURN NULL;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER empHist AFTER INSERT OR UPDATE OR
DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE
procEmpHist();
```

42

Exemplos | Inserir dados em tabela

```
CREATE TABLE emp (
    empname text,
    salary integer
);
CREATE TABLE empHist(
    operation char(1),
```

```
CREATE FUNCTION procEmpHist() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'DELETE' THEN
        INSERT INTO empHist SELECT 'D', now(), user, OLD.*;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO empHist SELECT 'U', now(), user, NEW.*;
    ELSIF TG_OP = 'INSERT' THEN
        INSERT INTO empHist SELECT 'I', now(), user, NEW.*;
    END IF;
    RETURN NULL;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER empHist AFTER INSERT OR UPDATE OR
DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE
procEmpHist();
```

TG_OP: Operação que disparou o gatilho

43

Exemplos | Inserir dados em tabela

```
CREATE TABLE emp (
    empname text,
    salary integer
);
CREATE TABLE empHist(
    operation char(1),
```

```
CREATE FUNCTION procEmpHist() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'DELETE' THEN
        INSERT INTO empHist SELECT 'D', now(), user, OLD.*;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO empHist SELECT 'U', now(), user, NEW.*;
    ELSIF TG_OP = 'INSERT' THEN
        INSERT INTO empHist SELECT 'I', now(), user, NEW.*;
    END IF;
    RETURN NULL;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER empHist AFTER INSERT OR UPDATE OR
DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE
procEmpHist();
```

Insere dados em outra tabela.

Outro gatilho poderia disparar...

44

Exemplos | Inserir dados em tabela

```
CREATE TABLE emp (
    empname text,
    salary integer
);
CREATE TABLE empHist(
    operation char(1),
```

PALAVRAS OBRIGATÓRIAS

QUANDO

OPERAÇÃO

TIPO

```
CREATE FUNCTION procEmpHist() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'DELETE' THEN
        INSERT INTO empHist SELECT 'D', now(), user, OLD.*;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO empHist SELECT 'U', now(), user, NEW.*;
    ELSIF TG_OP = 'INSERT' THEN
        INSERT INTO empHist SELECT 'I', now(), user, NEW.*;
    END IF;
    RETURN NULL;
END; $$ LANGUAGE plpgsql;
```

CREATE TRIGGER empHist **AFTER INSERT OR UPDATE OR DELETE ON** emp **FOR EACH ROW EXECUTE PROCEDURE** procEmpHist();

45

Exemplos | Manutenção de tabela

```
CREATE TABLE salesfact (
    timekey integer,
    productkey integer,
    storekey integer,
    amtsold numeric(9,2),
    unitssold integer,
    amtcost numeric(9,2)
);

CREATE TABLE summary(
    time_key integer,
    amtsold numeric(9,2),
    unitssold numeric(12),
    amtcost numeric(9,2)
);
```

```
CREATE FUNCTION maintSummary() RETURNS trigger AS $$
DECLARE
    delta_timekey integer;
    delta_amtsold numeric(9,2);
    delta_unitssold numeric(12);
    delta_amtcost numeric(9,2);
BEGIN
    IF TG_OP = 'DELETE' THEN
        delta_timekey := OLD.timekey;
        delta_amtsold := -1 * OLD.amtsold;
        delta_unitssold := -1 * OLD.unitssold;
        delta_amtcost := -1 * OLD.amtcost;   -- . . .
```

46

Exemplos | Manutenção de tabela

```
CREATE TABLE salesfact (
   timekey integer,
   productkey integer,
   storekey integer,
   amtsold numeric(9,2),
   unitssold integer,
   amtcost numeric(9,2)
);

CREATE TABLE summary(
   time_key integer,
   amtsold numeric(9,2),
   unitssold numeric(12),
   amtcost numeric(9,2)
);
```

```
CREATE FUNCTION maintSummary() RETURNS trigger AS $$
DECLARE
   delta_timekey integer;
   delta_amtsold numeric(9,2);
   delta_unitssold numeric(12);
   delta_amtcost numeric(9,2);
BEGIN  -- (. . .)
   ELSIF TG_OP = 'UPDATE' THEN
      IF OLD.timekey != NEW.timekey THEN
            RAISE EXCEPTION 'Update of timekey not allowed';
      END IF;
      delta_timekey := OLD.timekey;
      delta_amtsold := NEW.amtsold - OLD.amtsold;
      delta_unitssold := NEW.unitssold - OLD.unitssold;
      delta_amtcost := NEW.amtcost - OLD.amtcost;  -- . . .
```

47

Exemplos | Manutenção de tabela

```
CREATE TABLE salesfact (
   timekey integer,
   productkey integer,
   storekey integer,
   amtsold numeric(9,2),
   unitssold integer,
   amtcost numeric(9,2)
);

CREATE TABLE summary(
   time_key integer,
   amtsold numeric(9,2),
   unitssold numeric(12),
   amtcost numeric(9,2)
);
```

```
CREATE FUNCTION maintSummary() RETURNS trigger AS $$
DECLARE
   delta_timekey integer;
   delta_amtsold numeric(9,2);
   delta_unitssold numeric(12);
   delta_amtcost numeric(9,2);
BEGIN  -- (. . .)
   ELSIF TG_OP = 'INSERT' THEN
      delta_timekey := NEW.timekey;
      delta_amtsold := NEW.amtsold;
      delta_unitssold := NEW.unitssold;
      delta_amtcost := NEW.amtcost;
   END IF;  -- . . .
```

48

Exemplos | Manutenção de tabela

```
CREATE TABLE salesfact (
   timekey integer,
   productkey integer,
   storekey integer,
   amtsold numeric(9,2),
   unitssold integer,
   amtcost numeric(9,2)
);

CREATE TABLE summary(
   time_key integer,
   amtsold numeric(9,2),
   unitssold numeric(12),
   amtcost numeric(9,2)
);
```

```
CREATE FUNCTION maintSummary() RETURNS trigger AS $$
DECLARE
   delta_timekey integer;
   delta_amtsold numeric(9,2);
   delta_unitssold numeric(12);
   delta_amtcost numeric(9,2);
BEGIN  -- (. . .)
   INSERT INTO summary VALUES (
      delta_timekey, delta_amtsold,
      delta_unitssold, delta_amtcost);
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER maint_summary_bytime AFTER INSERT OR
UPDATE OR DELETE ON salesfact FOR EACH ROW EXECUTE
PROCEDURE maintSummary();
```

49

Sumário

- Introdução
- Gatilhos
  - Definição
  - Criação de gatilhos
- Funções
  - Diferenças
  - Variáveis especiais
  - Retorno
- Exemplos
- **Extra**
  - **Tratamento de erros**

50

Extra | Tratamento de erros

Cláusula EXCEPTION

- Quando ocorre um erro em uma função PL/pgSQL, o comportamento padrão é abortá-la
  - E o comando SQL que a disparou também
- Isso pode ser evitado com o uso da cláusula EXCEPTION

```
[ DECLARE declarations ]
BEGIN
        statements
EXCEPTION
        WHEN condition [ OR condition ... ] THEN
                handler_statements
        [ WHEN condition [ OR condition ... ] THEN
                handler_statements ... ]
END;
```

51

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

52

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

ERRO: divisão por zero

53

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

Existe uma exceção configurada para este erro, portanto o código a seguir é executado

54

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

Constante definida pelo PostgreSQL.
http://www.postgresql.org/docs/9.0/static/errcodes-appendix.html

55

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

Variáveis mantém o mesmo valor que possuíam no momento que o erro ocorreu, e podem ser acessadas dentro do EXCEPTION

56

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

Qualquer alteração no banco de dados feita dentro do bloco é anulada (rollback)

57

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

INSERT está fora do bloco, portanto é executado

58

Extra | Tratamento de erros

Exemplo de uso de EXCEPTION

```
. . .

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
BEGIN
        UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
        x := x + 1;
        y := x / 0;
EXCEPTION
        WHEN division_by_zero THEN
                RAISE NOTICE 'caught division_by_zero';
                RETURN x;
END;

. . .
```

59

Extra | Tratamento de erros

Cláusula RAISE EXCEPTION

- É possível provocar exceções usando a cláusula RAISE EXCEPTION

+

- Exceções anulam a execução de comandos SQL

+

- RAISE EXCEPTION pode ser usado em gatilhos

=

- Com gatilhos, é possível fazer verificações mais apuradas dos comandos SQL, corrigir erros e, se necessário, abortar transações

60

Extra | Tratamento de erros

Exemplo de uso de RAISE EXCEPTION (1)

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'UPDATE' AND NEW.empname IS NULL THEN
        NEW.empname := OLD.empname;
    ELSEIF TG_OP = 'INSERT' AND NEW.empname IS NULL THEN
        RAISE EXCEPTION 'Proibido empregado sem nome!';
    END IF;
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON
emp FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

61

Extra | Tratamento de erros

Exemplo de uso de RAISE EXCEPTION (2)

```
CREATE FUNCTION trg_check_max_4_updated_records() RETURNS trigger AS $$
DECLARE
    counter_ integer := 0;
BEGIN
    PERFORM true FROM pg_class
    WHERE relname='check_max_4' AND relnamespace=pg_my_temp_schema();
    IF NOT FOUND THEN
        CREATE TEMPORARY TABLE check_max_4 (counter integer) ON COMMIT DROP;
        INSERT INTO check_max_4 VALUES (1);
    ELSE
        --...
```

62

Extra | Tratamento de erros

Exemplo de uso de RAISE EXCEPTION (2)

```
-- (. . .)
ELSE
   SELECT counter FROM check_max_4 INTO counter_;
   counter_ = counter_+1;
   UPDATE check_max_4 SET counter = counter_;
   RAISE NOTICE 'Actual value for counter=%', counter_;
   IF counter_ > 4 THEN
      RAISE EXCEPTION 'Cannot change more than 4 rows in one transaction';
   END IF;
END IF;
RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER trg_bu_test BEFORE UPDATE ON test FOR EACH ROW EXECUTE
PROCEDURE trg_check_max_4_updated_records();
```

63

Dúvidas?

64