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

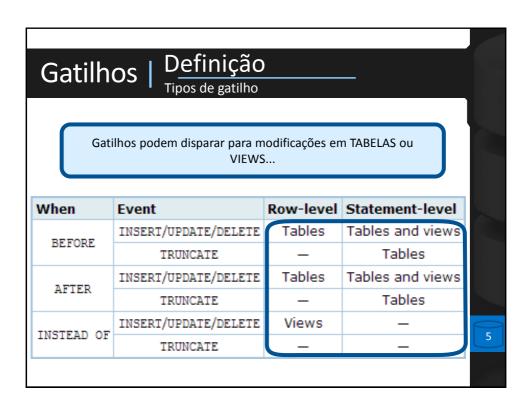
Introdução | O que são gatilhos

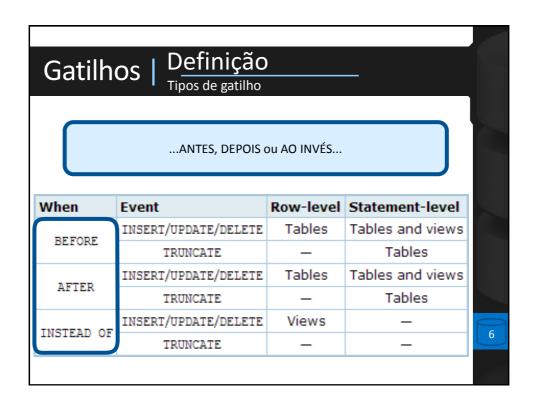
- Um gatilho (em inglês, trigger) é uma especificação da ação que o banco de dados deve executar toda vez que determinado evento ocorrer
 - Ação = função
 - evento = executar certo tipo de operação
- É possível criar gatilhos usando diversas linguagens, entre elas, PL/pgSQL
- Não é possível criar gatilhos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

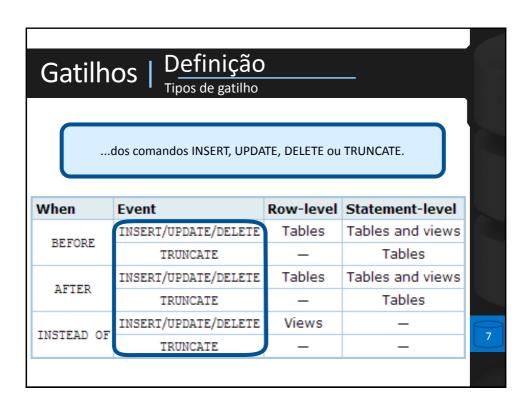
3

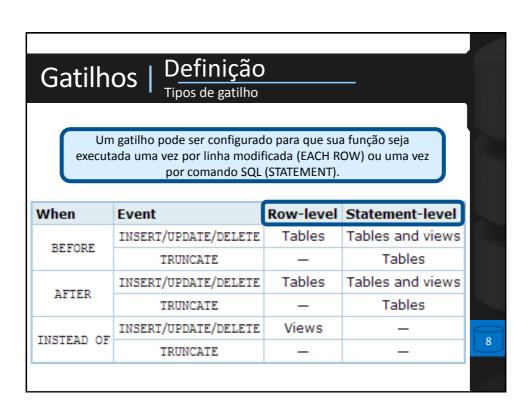
Gatilhos | Definição Tipos de gatilho

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	_	_









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
 - . 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
 - 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

Gatilhos | Definição

Quando a função é executada

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

Gatilhos | Definição

Quando a função é executada

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

Gatilhos | Definição

Visibilidade dos dados alterados

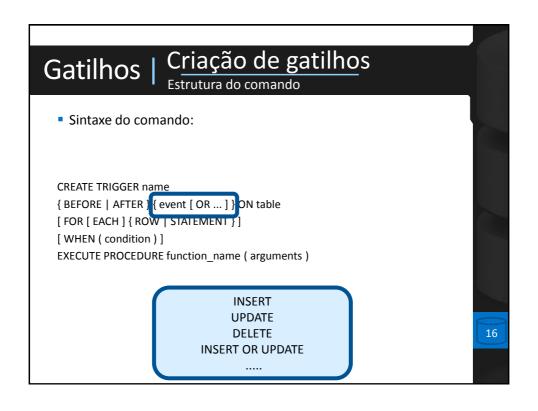
- A visibilidade das alterações depende do nível do evento e do momento de seu disparo:
 - Se o gatilho for BEFORE STATEMENT
 - O SQL ainda não foi executado, portanto nenhuma mudança é visível
 - Se o gatilho for AFTER STATEMENT
 - O SQL já foi executado, portanto todas as mudanças são visíveis
 - Se 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

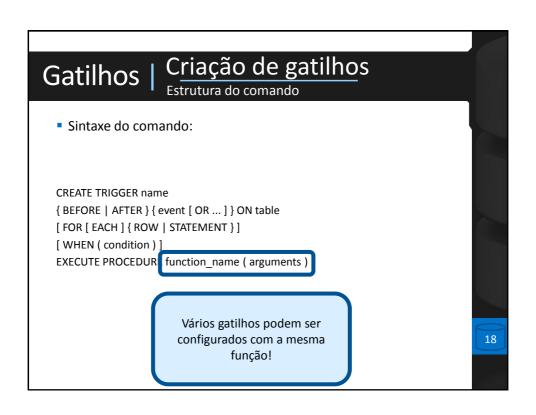
Gatilhos | Definição

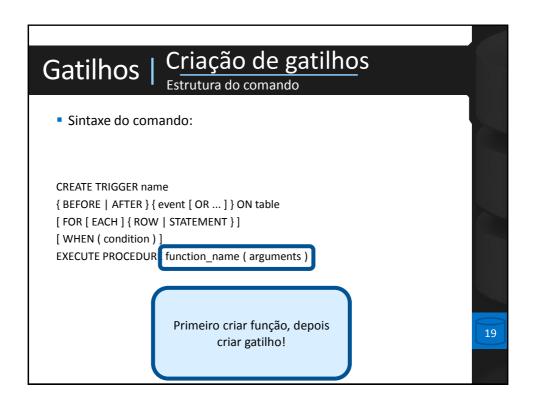
- Se um gatilho altera os dados de entrada, o gatilho seguinte receberá os dados alterados, e não os originais
- Se um gatilho executa comandos SQL, esses comandos podem disparar gatilhos ("gatilhos em cascata")
 - Não existe uma quantidade máxima de níveis da cascata
 - É possível que uma cascata dispare 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!



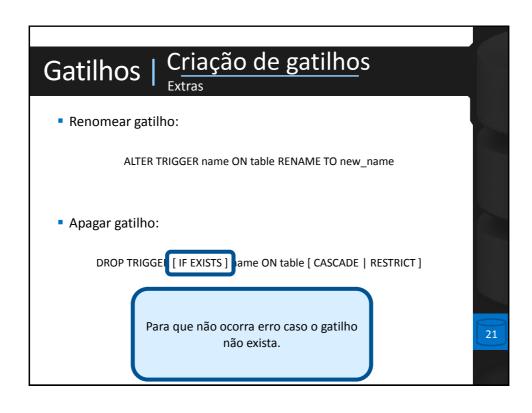


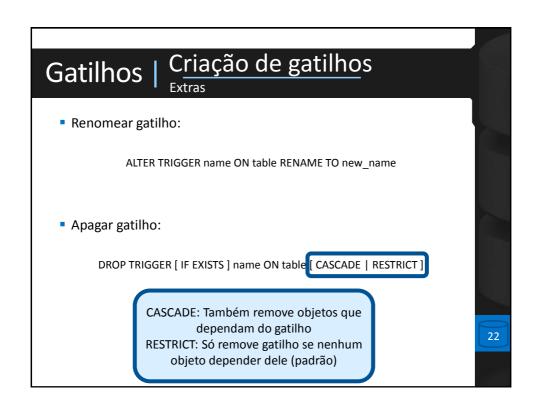












Sumário Introdução Gatilhos Definição Criação de gatilhos Funções Diferenças Variáveis especiais Retorno Exemplos Extra Tratamento de erros

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_funcap() ETURN trigger \(\) RETURN NULL; END; \$\$ LANGUAGE plpgsql;



Funções | Variáveis especiais

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

Funções | Variáveis especiais

- 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

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

Funções | Variáveis especiais

- 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.

3

Sumário

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

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

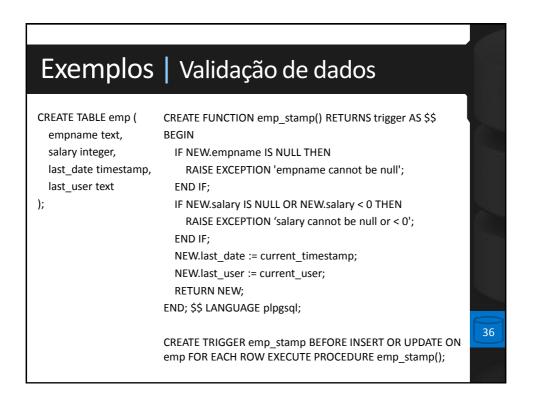
22

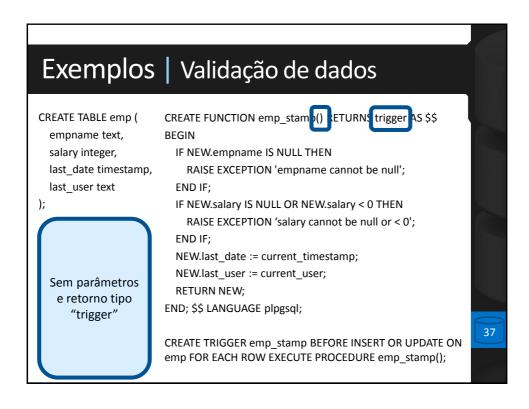
Funções | Retorno

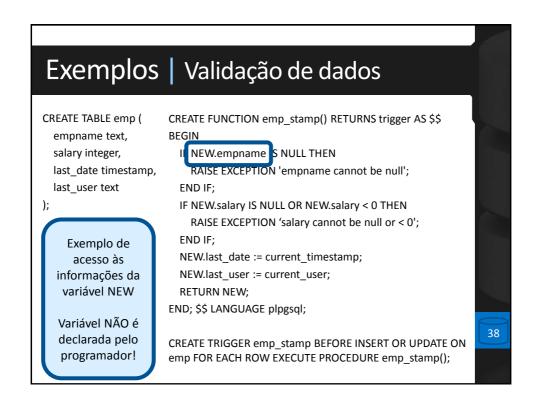
Alterar comando SQL

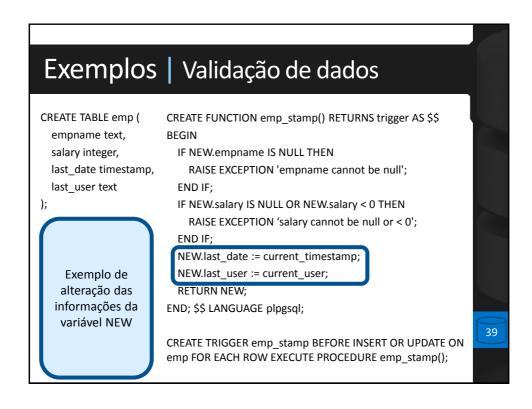
- Se o gatilho for do tipo INSTEAD OF:
 - Pode retornar NULL para informar que nenhum dado foi modificado
 - Pode retornar qualquer coisa diferente de NULL para informar que a linha foi alterada
 - Isso faz com que o contador de linhas alteradas (GET DIAGNOSTICS ROW_COUNT) seja incrementado

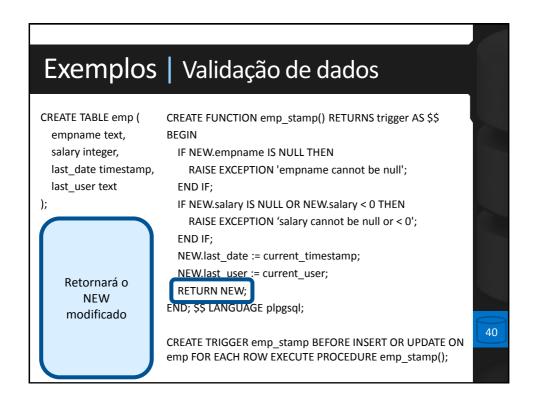
Sumário Introdução Gatilhos Definição Criação de gatilhos Funções Diferenças Variáveis especiais Retorno Exemplos Extra Tratamento de erros

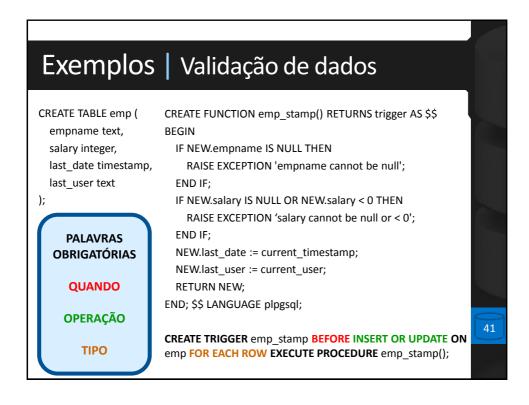






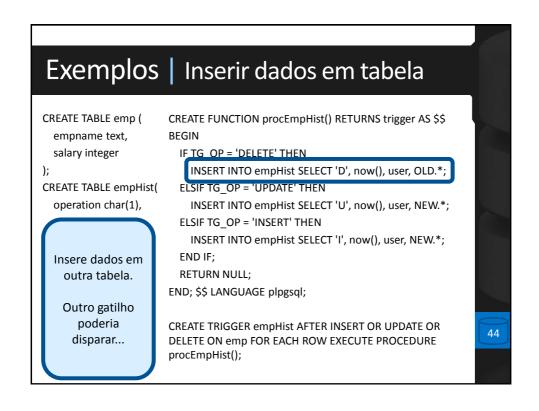






Exemplos | Inserir dados em tabela CREATE TABLE emp (CREATE FUNCTION procEmpHist() RETURNS trigger AS \$\$ empname text, **BEGIN** salary integer IF TG OP = 'DELETE' THEN INSERT INTO empHist SELECT 'D', now(), user, OLD.*; CREATE TABLE empHist(ELSIF TG OP = 'UPDATE' THEN operation char(1), INSERT INTO empHist SELECT 'U', now(), user, NEW.*; ELSIF TG_OP = 'INSERT' THEN stamp timestamp, userid text, INSERT INTO empHist SELECT 'I', now(), user, NEW.*; empname text, END IF; salary integer **RETURN NULL;**); END; \$\$ LANGUAGE plpgsql; CREATE TRIGGER empHist AFTER INSERT OR UPDATE OR 42 DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE procEmpHist();

```
Exemplos | Inserir dados em tabela
CREATE TABLE emp (
                      CREATE FUNCTION procEmpHist() RETURNS trigger AS $$
  empname text,
                      BEGIN
                        I TG_OP = 'DELETE' HEN
  salary integer
                           INS<u>ERT INTO empHist S</u>ELECT 'D', now(), user, OLD.*;
);
CREATE TABLE empHist(
                        ELSIF TG OP = 'UPDATE'
                                              HEN
  operation char(1),
                          INSERT INTO empHist SELECT 'U', now(), user, NEW.*;
                        ELSI TG OP = 'INSERT' 1 HEN
                          INSERT INTO empHist SELECT 'I', now(), user, NEW.*;
                        END IF;
                        RETURN NULL;
      TG OP:
   Operação que
                      END; $$ LANGUAGE plpgsql;
     disparou o
      gatilho
                      CREATE TRIGGER empHist AFTER INSERT OR UPDATE OR
                      DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE
                      procEmpHist();
```



```
Exemplos | Inserir dados em tabela
CREATE TABLE emp (
                      CREATE FUNCTION procEmpHist() RETURNS trigger AS $$
 empname text,
                      BEGIN
 salary integer
                       IF TG_OP = 'DELETE' THEN
);
                         INSERT INTO empHist SELECT 'D', now(), user, OLD.*;
CREATE TABLE empHist(
                       ELSIF TG OP = 'UPDATE' THEN
  operation char(1),
                         INSERT INTO empHist SELECT 'U', now(), user, NEW.*;
                        ELSIF TG OP = 'INSERT' THEN
                         INSERT INTO empHist SELECT 'I', now(), user, NEW.*;
    PALAVRAS
                       END IF;
  OBRIGATÓRIAS
                       RETURN NULL;
     QUANDO
                      END; $$ LANGUAGE plpgsql;
    OPERAÇÃO
                      CREATE TRIGGER empHist AFTER INSERT OR UPDATE OR
                      DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE
       TIPO
                      procEmpHist();
```

```
Exemplos | Manutenção de tabela
CREATE TABLE salesfact ( CREATE FUNCTION maintSummary() RETURNS trigger AS $$
  timekey integer,
                       DECLARE
  productkey integer,
                         delta_timekey integer;
  storekey integer,
                         delta amtsold numeric(9,2);
  amtsold numeric(9,2),
                         delta_unitssold numeric(12);
  unitssold integer,
                         delta_amtcost numeric(9,2);
  amtcost numeric(9,2) BEGIN
                         IF TG OP = 'DELETE' THEN
);
                           delta timekey := OLD.timekey;
CREATE TABLE summary(
                           delta_amtsold := -1 * OLD.amtsold;
  time_key integer,
                           delta_unitssold := -1 * OLD.unitssold;
  amtsold numeric(9,2),
                           delta_amtcost := -1 * OLD.amtcost; --...
  unitssold numeric(12),
                                                                                  46
  amtcost numeric(9,2)
```

```
Exemplos | Manutenção de tabela
CREATE TABLE salesfact ( CREATE FUNCTION maintSummary() RETURNS trigger AS $$
  timekey integer,
                       DECLARE
  productkey integer,
                         delta_timekey integer;
  storekey integer,
                         delta amtsold numeric(9,2);
  amtsold numeric(9,2),
                         delta unitssold numeric(12);
  unitssold integer,
                         delta amtcost numeric(9,2);
  amtcost numeric(9,2) BEGIN -- (...)
                         ELSIF TG OP = 'UPDATE' THEN
                           IF OLD.timekey != NEW.timekey THEN
CREATE TABLE summary(
                               RAISE EXCEPTION 'Update of timekey not allowed';
  time key integer,
  amtsold numeric(9,2),
                           delta timekey := OLD.timekey;
  unitssold numeric(12),
                           delta_amtsold := NEW.amtsold - OLD.amtsold;
  amtcost numeric(9,2)
                           delta unitssold := NEW.unitssold - OLD.unitssold;
```

);

delta_amtcost := NEW.amtcost - OLD.amtcost; --...

```
Exemplos | Manutenção de tabela
CREATE TABLE salesfact ( CREATE FUNCTION maintSummary() RETURNS trigger AS $$
  timekey integer,
                       DECLARE
  productkey integer,
                         delta timekey integer;
  storekey integer,
                         delta amtsold numeric(9,2);
  amtsold numeric(9,2),
                         delta_unitssold numeric(12);
  unitssold integer,
                         delta_amtcost numeric(9,2);
  amtcost numeric(9,2) BEGIN -- (...)
                         ELSIF TG OP = 'INSERT' THEN
);
                           delta timekey := NEW.timekey;
CREATE TABLE summary(
                           delta_amtsold := NEW.amtsold;
  time_key integer,
                           delta_unitssold := NEW.unitssold;
  amtsold numeric(9,2),
                           delta_amtcost := NEW.amtcost;
  unitssold numeric(12),
                         END IF; --...
                                                                                  48
  amtcost numeric(9,2)
```

```
Exemplos | Manutenção de tabela
CREATE TABLE salesfact ( CREATE FUNCTION maintSummary() RETURNS trigger AS $$
                      DECLARE
 timekey integer,
 productkey integer,
                        delta_timekey integer;
 storekey integer,
                        delta_amtsold numeric(9,2);
 amtsold numeric(9,2),
                        delta unitssold numeric(12);
 unitssold integer,
                        delta_amtcost numeric(9,2);
 amtcost numeric(9,2) BEGIN -- (...)
                        INSERT INTO summary VALUES (
                          delta_timekey, delta_amtsold,
CREATE TABLE summary(
                          delta_unitssold, delta_amtcost);
 time key integer,
                      END; $$ LANGUAGE plpgsql;
 amtsold numeric(9,2),
 unitssold numeric(12), CREATE TRIGGER maint_summary_bytime AFTER INSERT OR
                                                                                49
 amtcost numeric(9,2) UPDATE OR DELETE ON salesfact FOR EACH ROW EXECUTE
                      PROCEDURE maintSummary();
);
```

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

Extra | Tratamento de erros

- 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 \dots ] THEN
                  handler_statements
        [ WHEN condition [ OR condition ... ] THEN
                  handler_statements ... ]
END;
```

Tratamento de erros Extra |

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;
. . .
```

```
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

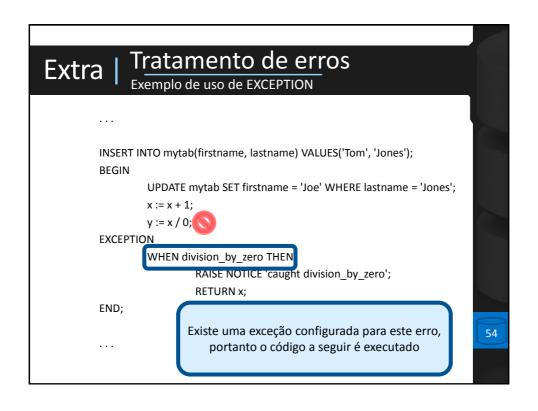
WHEN division_by_zero THEN

RAISE NOTICE 'caught division_by_zero';
RETURN x;

END;

...

ERRO: divisão por zero
```



```
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 HEN

RAISE NOTICE 'caught division_by_zero';

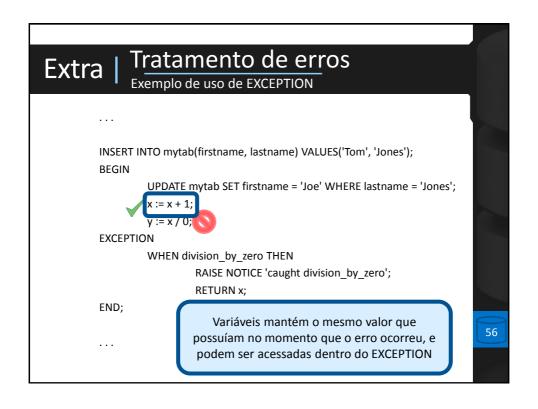
RETURN x;

END;

Constante definida pelo PostgreSQL.

http://www.postgresql.org/docs/9.0/static/err

codes-appendix.html
```



```
INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');

BEGIN

UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';

x := x + 1;

y := x / 0;

EXCEPTION

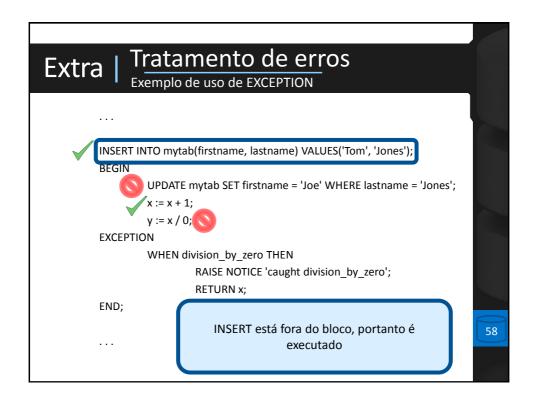
WHEN division_by_zero THEN

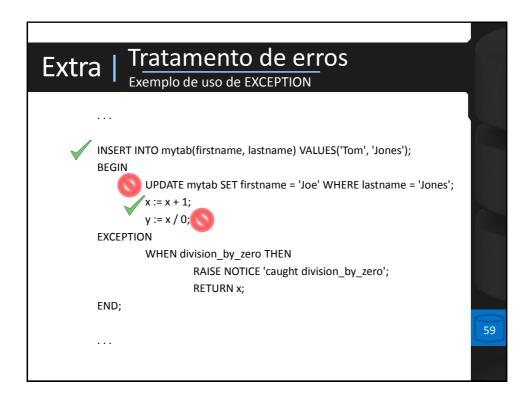
RAISE NOTICE 'caught division_by_zero';

RETURN x;

END;

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





Extra | Tratamento de erros Cláusula RAISE EXCEPTION

Claasala IVIISE EXCEL LIGIT

• 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

Tratamento de erros Extra | Exemplo de uso de RAISE EXCEPTION (1) CREATE TABLE emp (CREATE FUNCTION emp stamp() RETURNS trigger AS \$\$ empname text, **BEGIN** IF TG_OP = 'UPDATE' AND NEW.empname IS NULL THEN salary integer, last_date timestamp, NEW.empname := OLD.empname; last user text 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();

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 --...

```
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;
                                                                              63
CREATE TRIGGER trg_bu_test BEFORE UPDATE ON test FOR EACH ROW EXECUTE
PROCEDURE trg_check_max_4_updated_records();
```

