

## *Triggers (ou Gatilhos) no PostgreSQL*



Profa. Socorro Vânia

# [ Triggers ou Gatilhos ]

- Os triggers (ou **gatilhos**) são procedimentos armazenados que são acionados automaticamente por algum **evento** e em determinado **momento**. Ou seja, são disparados dependendo da ocorrência de um evento.

# [ Triggers ou Gatilhos ]

- Estes eventos podem ser: **INSERT**, **UPDATE**, **DELETE** e **TRUNCATE**.
- Logo que ocorre um desses eventos a **função do gatilho** é disparada automaticamente para tratar o evento.



# [Triggers ou Gatilhos

## ■ Utilização

- Quase sempre devemos **validar** ou **averiguar** informações antes de efetivar um ou mais comandos SQL.
- Ao termos uma grande quantidade de acessos ao banco de dados por múltiplas aplicações, a utilização de triggers é de grande utilidade, e com isso, podemos manter a integridade de dados complexos;
- Além disso, com triggers poderemos acompanhar as mudanças ou o log a cada modificação ocorrida nos dados presentes numa tabela.

# [ Triggers ou Gatilhos ]

- Os triggers são recursos dos SGBDs que auxiliam no reforço da integridade referencial de um BD, definindo operações que serão executadas quando um determinado **evento (inserção/alteração/exclusão)** ocorrer na tabela a qual está associado.
- Cada banco de dados implementa funções de gatilho de uma forma um pouco diferente uns dos outros.

# [ Triggers ou Gatilhos ]

- Os gatilhos podem ser disparados:
  - antes da execução do evento (**BEFORE**)
  - depois da execução do evento (**AFTER**)

# [Triggers no PostgreSQL]

- Um diferencial de triggers no PostgreSQL de outros SGBDs é que aqui **triggers são sempre associadas a funções de triggers** (veremos isso mais adiante), já nos demais SGBDs, criamos o corpo do trigger na declaração da própria trigger.



# [ Criando um trigger ]

- A sintaxe para a criação de um *trigger* no PostgreSQL é apresentada abaixo:

```
CREATE TRIGGER nomeDoTrigger { BEFORE | AFTER } { evento [ OR ... ] }  
ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]  
EXECUTE PROCEDURE nomeDaFunção ( argumentos )
```

```
CREATE TRIGGER trg_VerificaSalEmp BEFORE INSERT OR UPDATE  
ON empregados FOR EACH ROW  
EXECUTE PROCEDURE VerificaEmp();
```



# [ Criando um trigger ]

- Sempre devemos declarar quando o trigger deve ser disparado: antes (BEFORE) ou após (AFTER) um evento (INSERT, UPDATE ou DELETE) em determinada tabela, para cada linha (ROW) ou instrução (STATEMENT), e qual função (PROCEDURE) deve ser executada.

# [ Criando um trigger ]

- Onde:
  - **NomeDoTrigger**: define o nome do trigger
  - **Before** ou **After**: define se o trigger será executado antes ou depois da ação que o disparou.
  - **Evento**: indica qual ação dispara o trigger. As ações podem ser: insert, update ou delete. O mesmo trigger pode ser disparado por diferentes ações, exemplo: insert ou update.
  - **Tabela**: a tabela a qual o trigger está associado.

# [ Criando um trigger ]

- Onde:
  - **For each row**: indica que o trigger será executado para cada linha alterada pela ação que o disparou (row-level).
  - **For each statement**: indica que o trigger será executado uma única vez para a ação que o disparou.
  - **Nome da Função**: o nome da função que será executada quando o trigger for disparado.
  - **Argumentos**: argumentos a serem passados para a função trigger.

# [ Tipos de Triggers ]

```
CREATE [ CONSTRAINT ] TRIGGER NAME { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }  
  ON table_NAME  
  [ FROM referenced_table_NAME ]  
  [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } ]  
  [ FOR [ EACH ] { ROW | STATEMENT } ]  
  [ WHEN ( condition ) ]  
  EXECUTE PROCEDURE function_NAME ( arguments )  
-- Quando um evento for declarado com:  
  INSERT  
  UPDATE [ OF column_NAME [, ... ] ]  
  DELETE  
  TRUNCATE
```

# [Tipos de Triggers]

- A expressão **condition** é uma expressão booleana que determina se a **trigger function** será executada.
- Se a condição WHEN for especificada, a função será chamada se a condição retornar true.

# [Tipos de Triggers]

- **Trigger-por-linha** é disparado uma vez para cada registro afetado pela instrução que disparou o trigger.
- **Trigger-por-instrução** é disparado somente uma vez quando uma instrução é executada.

# [Tipos de Triggers]

- **Por exemplo:**

- Se uma instrução UPDATE for executada, e esta afetar seis linhas, temos que a trigger de nível de linha será executada seis vezes, enquanto que a trigger a nível de instrução será chamada apenas uma vez por instrução SQL

# [ Funções de trigger ]

- No PostgreSQL, um trigger sempre está associado a uma função, chamada **Função de trigger**.
- Funções de trigger são funções que **não recebem nenhum parâmetro e retornam o tipo trigger**.



# [ Funções de trigger ]

- As triggers functions recebem, através de uma entrada especial, uma estrutura **TriggerData**, que possui um conjunto de variáveis locais que podemos usar nas nossas triggers functions.

# Funções de trigger e linguagens

- Uma função de trigger tem acesso a variáveis especiais, mas as mais utilizadas são **NEW** E **OLD**

Nome	Tipo	Descrição
NEW	RECORD	contém a nova linha criada por um insert ou update. Só existe em triggers do tipo row-level
OLD	RECORD	contém os valores antigos de uma linha apagada ou atualizada por um delete ou update. Só existem em triggers do tipo row-level
TG_NAME	name	nome do trigger disparado
TG_WHEN	text	indica se o trigger é do tipo before ou after
TG_LEVEL	text	contém o valor row ou statement
TG_OP	text	contém o evento que disparou o trigger: insert, update ou delete
TG_RELNAME	name	contém o nome da tabela associada ao trigger
TG_ARGV[]	vetor de texto	variável que contém os argumentos passados pelo comando CREATE TRIGGER. Vale ressaltar que o índice do vetor começa em 0 (zero).

# [ Variáveis NEW e OLD ]

- Como visto, o PostgreSQL disponibiliza duas variáveis importantes para serem usadas em conjunto com as triggers-por-linha: NEW e OLD.

# [ Variáveis NEW e OLD ]

- **A variável NEW**, no caso do **INSERT**, armazena o registro que está sendo inserido. No caso do **UPDATE**, armazena a nova versão do registro depois da atualização.
- **A variável OLD**, no caso do **DELETE**, armazena o registro que está sendo excluído. No caso do **UPDATE**, armazena a antiga versão do registro depois da atualização.

# [ Variáveis NEW e OLD ]

- OLD não é usada para **INSERTS**.
- NEW não é usada para **DELETE**.
- Os valores novos (NEW) de uma linha só podem ser alterados por gatilhos que disparam **BEFORE**.
- Atribuir um valor a NEW.coluna em um gatilho que dispara after INSERT ou UPDATE não fará efeito.

# [ Variáveis NEW e OLD ]

- A sintaxe das variáveis é:

**NEW.coluna**

**OLD.coluna**

- Onde coluna é qualquer coluna da tabela associada ao trigger.
- Essas variáveis podem ser utilizadas em qualquer lugar onde uma variável local ou parâmetro podem ser utilizados.

# [ Criando uma trigger ]

## Exemplo:

Vejamos o exemplo de um trigger que impede que sejam inseridos registros de empregados com nome nulo ou vazio, ou com o valor do salário menor que zero.

Empregados	
codigo	integer
nome	dm_nome
Datanascimento	date
salario	numeric(10,2)
cod_departamento	integer
cod_chefe	integer

Estrutura da tabela de Empregados

# Criando uma função de trigger

## Exemplo:

Criando a função de trigger VerificaEmp()

```
CREATE OR REPLACE FUNCTION VerificaEmp()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.nome IS NULL OR NEW.nome="" THEN
        RAISE EXCEPTION 'Nome não pode ser nulo ou vazio.';
    END IF;

    IF NEW.salario IS NULL OR NEW.salario < 0 THEN
        RAISE EXCEPTION 'Salario não pode ser nulo e deve ser maior do que zero.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

**RAISE EXCEPTION** gera um erro de exceção, provocando o cancelamento da transação corrente.



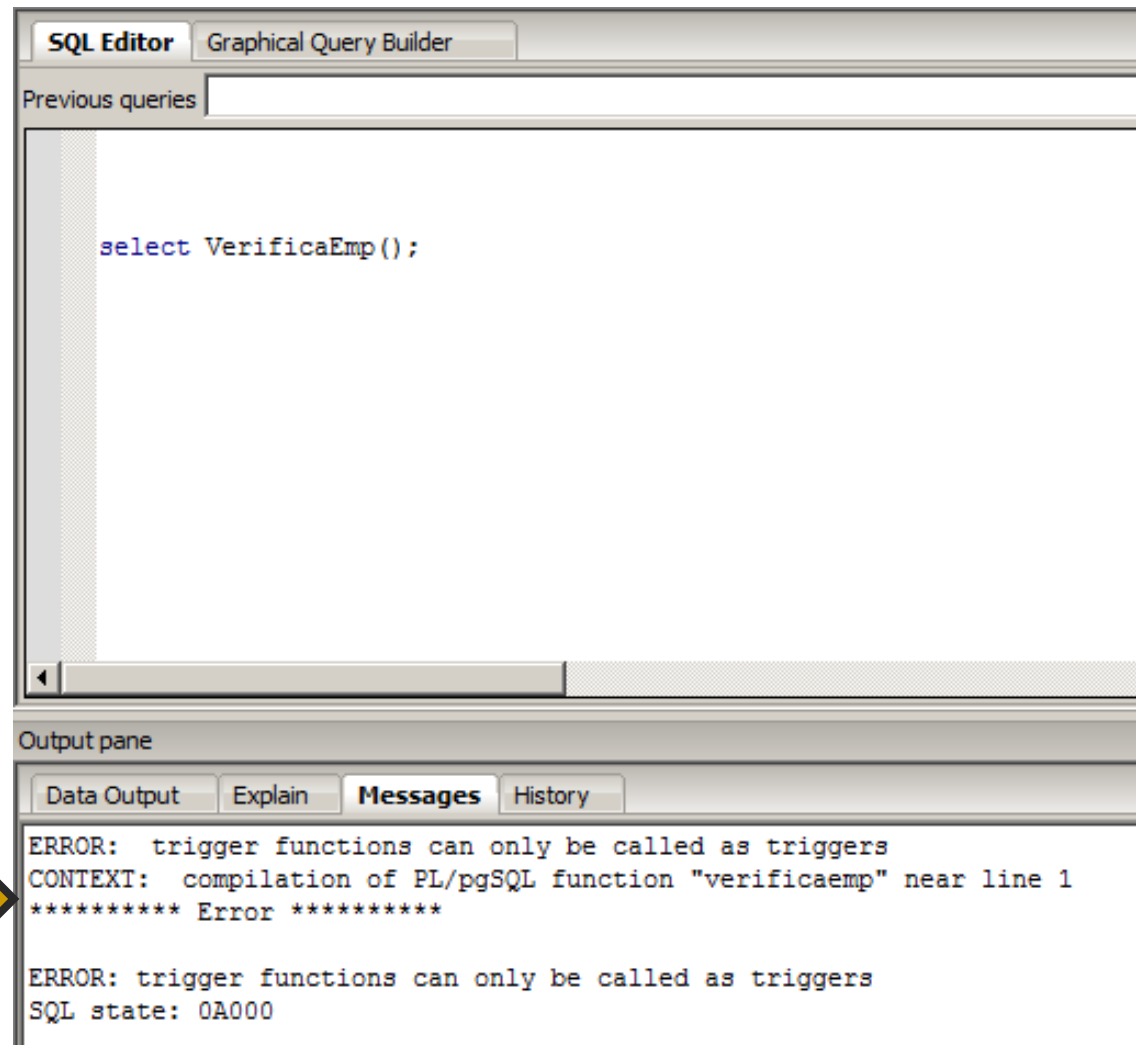
# Criando uma função de trigger

## Exemplo:

- ❑ A primeira mudança notável em relação as demais funções em PL/pgSQL é o retorno do tipo trigger, indicando que se se trata exclusivamente de uma função desse tipo.
- ❑ Se a função for chamada isoladamente ao invés de ser disparada por um trigger, o PostgreSQL acusará erro e a transação corrente será abortada, como exemplificado no próximo slide.

# Criando uma função de trigger

**Exemplo:**



The screenshot shows a SQL Editor window with two tabs: "SQL Editor" and "Graphical Query Builder". The "SQL Editor" tab is active, and the query editor contains the text `select VerificaEmp();`. Below the query editor is a "Previous queries" section. At the bottom of the window is an "Output pane" with four tabs: "Data Output", "Explain", "Messages", and "History". The "Messages" tab is selected, displaying two error messages. A large yellow arrow points from the word "Erro" to the first error message.

```
select VerificaEmp();
```

Output pane

Data Output Explain **Messages** History

ERROR: trigger functions can only be called as triggers  
CONTEXT: compilation of PL/pgSQL function "verificaemp" near line 1  
\*\*\*\*\* Error \*\*\*\*\*

ERROR: trigger functions can only be called as triggers  
SQL state: 0A000

**Erro**

# [ Criando uma trigger ]

## Exemplo:

Criando um trigger VerificaSalEmp:

```
CREATE TRIGGER trg_VerificaSalEmp BEFORE INSERT OR UPDATE  
ON empregados FOR EACH ROW  
EXECUTE PROCEDURE VerificaEmp();
```

Execute os códigos abaixo para verificar o resultado da execução do trigger

```
INSERT INTO empregados(codigo, nome, data_nasc, cod_departamento, cod_chefe)  
VALUES (4, 'Andre Gonçalves', '01-01-1975', 1, 3);
```

```
UPDATE empregados  
SET salario=-100.00  
WHERE codigo=3;
```

```
INSERT INTO empregados(codigo, data_nasc, salario, cod_departamento, cod_chefe)  
VALUES (5, '01-09-1977', 1050.00, 3, 3);
```

# [Passando argumentos para um trigger]

- ❑ É possível passar argumentos para uma função de trigger. **No entanto, esses argumentos só devem ser especificados no comando CREATE TRIGGER e não na criação da função (CREATE FUNCTION).**
- ❑ O acesso aos argumentos é feito através da variável especial **TG\_ARGV**, que é um vetor do tipo text, onde os índices do vetor correspondem à ordem em que os argumentos foram passados.
- ❑ A passagem de argumentos pode ser muito útil quando temos dois triggers diferentes que disparam a mesma função.

# [Passando argumentos para um trigger]

## **Exemplo:**

Utilizando argumentos:

```
CREATE TRIGGER argumentoExemplo1 BEFORE DELETE  
ON empregados FOR EACH ROW  
EXECUTE PROCEDURE argFunc('1');
```

```
CREATE TRIGGER argumentoExemplo2 BEFORE INSERT  
ON empregados FOR EACH ROW  
EXECUTE PROCEDURE argFunc('2');
```

# [Passando argumentos para um trigger]

## Exemplo:

Utilizando argumentos:

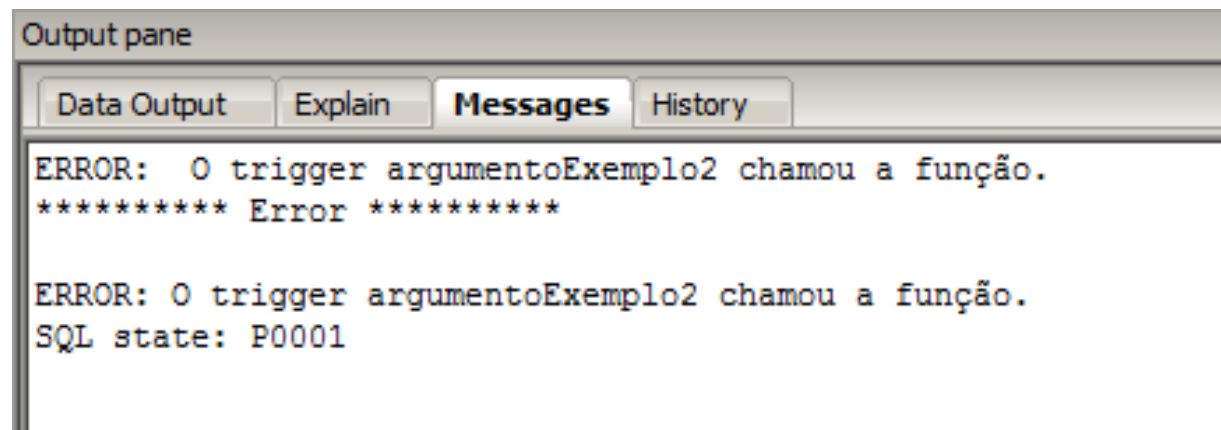
```
CREATE OR REPLACE FUNCTION argFunc()  
RETURNS TRIGGER AS $$  
  BEGIN  
    IF TG_ARGV[0] = '1' THEN  
      RAISE EXCEPTION 'O trigger argumentoExemplo1 chamou a função.';  
    ELSE  
      RAISE EXCEPTION 'O trigger argumentoExemplo2 chamou a função.';  
    END IF;  
    RETURN NEW;  
  END;  
$$ LANGUAGE plpgsql;
```

# [Passando argumentos para um trigger]

## Exemplo:

Agora teste os triggers criados anteriormente:

```
INSERT INTO empregados(codigo, nome, data_nasc, cod_departamento, cod_chefe)
VALUES (8, 'Maria Lima', '01-10-1989', 1, 3);
```



# [ Ignorando a sentença que disparou o trigger ]

- ❑ Digamos que diante de uma falha na validação dos valores não queiramos abortar a transação, mas simplesmente ignorar a operação que disparou o trigger (lembrando que o código da função de trigger é executado por completo).
- ❑ Para isso basta retirarmos os trechos com RAISE EXCEPTION da função verificaEmp() **e retornamos um valor nulo.**
- ❑ Isso fará com que nenhuma mensagem de erro seja exibida, e o comando SQL de insert ou update seja ignorado.



# [ Ignorando a sentença que disparou o trigger ]

## Exemplo:

Criando a 2ª. Versão da função de trigger VerificaEmp()

```
CREATE OR REPLACE FUNCTION VerificaEmp()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.nome IS NULL OR NEW.salario IS NULL THEN
        RETURN NULL;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

# [ Exclusão de Triggers ]

- Para remover um gatilho do banco de dados o comando DROP TRIGGER deve ser utilizado.

- **Sintaxe:**

**DROP TRIGGER nome ON tabela [ CASCADE | RESTRICT ];**

- Drop trigger verificaEmp ON empregados RESTRICT;
- Cascade indica que havendo objetos dependentes, estes serão apagados.
- Restrict impede que o trigger seja apagado se houver objetos dependentes (padrão)

# [ Exclusão de Triggers ]

- Vale lembrar que a **exclusão** de um trigger não implica na exclusão da função de trigger associada a ele. Dessa forma, não é necessário reescrever a função correspondente ao recriar um trigger.

# [ Ordem de execução de triggers ]

- Podemos ter mais de uma trigger associada ao mesmo evento e momento, neste caso a ordem de execução das triggers é definida pela ordem alfabética de seus nomes.

[

]

**Obrigada!**