



# Programando com SQL Triggers

Profa. Maria Camila Nardini Barioni

[camila.barioni@ufu.br](mailto:camila.barioni@ufu.br)

Bloco B - sala 1B137

1º semestre de 2024

# Avisos

- ◆ Com o conteúdo da aula de hoje já é possível finalizar o desenvolvimento do projeto
- ◆ Prova 2: 17/10
  - ◆ Conteúdo:
    - ◆ Normalização
    - ◆ Álgebra Relacional
    - ◆ Tudo sobre SQL
    - ◆ Procedimentos armazenados e gatilhos

# Sumário

◆ Gatilhos

◆ Exercícios

◆ Agendamento das apresentações dos projetos

# Trigger

- ◆ Gatilho ou Disparador
- ◆ É uma sub-rotina, semelhante as stored procedures, que tem como característica operacional ser executada automaticamente quando uma determinada ação (INSERT/UPDATE/DELETE) for realizada no banco de dados

# Trigger

- ◆ Uma das maneiras mais práticas de implementar rotinas para garantir a integridade de dados ou de operações
- ◆ Principal diferença entre Trigger e Stored Procedure
  - Trigger é executado automaticamente
  - Stored Procedure precisa ser explicitamente invocada

# Trigger

- ◆ Podem ser utilizados para gerenciar informações do banco de dados
  - Automatizar a geração de dados
  - Fazer a auditoria das modificações
  - Implantar as restrições complexas de integridade
  - Personalizar as autorizações complexas de segurança

# Exemplo...

## Voltando ao exercício da aula anterior

Modelo Relacional:

Cliente = { CodCli, NomeCli, Endereco, Cidade, CEP, UF, CGC }

Vendedor = { CodVend, NomeVend, Salfixo, FaixaComis }

Produto = { CodProd, Unidade, Descri, ValoUnit }

Pedido = { NumPed, Prazoentr, CodCli(Cliente.CodCli),  
CodVend(Vendedor.CodVend) }

ItemPedido = { NumPed(Pedido.NumPed),  
CodProd(Produto.CodProduto), Quant }

Quais gerações de dados poderiam ser automatizadas?

# Trigger

- ◆ Aciona stored procedures para executar determinadas tarefas
- ◆ Está associado a uma tabela ou uma visão



# Componentes de um Trigger

## 1. Comando SQL que aciona o trigger

- O disparo do trigger pode ser ocasionado pelo comando SQL ou por um evento do usuário
  - Em uma tabela, pelos comandos INSERT, UPDATE ou DELETE
  - Em um objeto de esquema, por meio dos comandos CREATE, ALTER ou DROP
  - No carregamento/shutdown do BD por uma mensagem de erro
  - No logon/logoff de um usuário

# Componentes de um Trigger

## 2. Limitador de ação do trigger

- Representado pela cláusula WHEN
  - Especifica qual condição deve ser verdadeira para que o trigger seja disparado

# Componentes de um Trigger

## 3. Ação executada pelo trigger

- É o bloco de comandos que é executado pelo trigger
  - Chama uma stored procedure

# Trigger

- ◆ Para criar um trigger é preciso
  - Definir um trigger procedure
  - Criar o trigger propriamente dito, que definirá quando o trigger procedure será executado

# Trigger

- O trigger procedure é muito similar a um stored procedure, mas é um pouco mais restrito devido à maneira como é chamado
  - Não possui parâmetros de entrada na função
  - Mas pode receber dados de entrada (ver slides à frente)
  - Deve retornar o tipo especial **trigger**
  - **Exemplo:**

```
CREATE FUNCTION trigger_procedure_name()  
-- Sem parâmetros de entrada  
RETURNS trigger AS -- Retorna tipo trigger  
$$  
BEGIN  
-- Aqui entra o corpo da trigger_procedure  
END; $$ language plpgsql;
```

# Trigger

- ◆ Para criar um trigger é preciso
  - Definir um trigger procedure
  - Criar o trigger propriamente dito, que definirá quando o trigger procedure será executado

# Trigger: FORMA GERAL

```
CREATE TRIGGER trigger_name  
  
{BEFORE | AFTER} trigger_event ON table_name  
  
FOR EACH {ROW | STATEMENT}  
  
[WHEN trigger_condition) ]  
  
EXECUTE PROCEDURE function_name ( arguments )
```

<https://www.postgresql.org/docs/current/triggers.html>

# Trigger: Evento

```
CREATE TRIGGER trigger_name  
  
{BEFORE | AFTER} trigger_event ON table_name  
  
FOR EACH {ROW | STATEMENT}  
  
[WHEN trigger_condition) ]  
  
EXECUTE PROCEDURE function_name ( arguments )
```

<https://www.postgresql.org/docs/current/triggers.html>



# Trigger: Evento

- ◆ O trigger dispara quando um evento específico ocorre (INSERT, DELETE ou UPDATE)
- ◆ Mais de um evento pode ser especificado na criação do trigger (separados por OR)

```
CREATE TRIGGER check_salario BEFORE  
INSERT OR UPDATE /* o trigger é chamado quando  
ocorre um INSERT ou UPDATE */  
ON empregado FOR EACH { ROW | STATEMENT }  
EXECUTE PROCEDURE função ( argumentos )
```

# Trigger: BEFORE ou AFTER

```
CREATE TRIGGER trigger_name  
  
{BEFORE | AFTER} trigger_event ON table_name  
  
FOR EACH {ROW | STATEMENT}  
  
[WHEN trigger_condition) ]  
  
EXECUTE PROCEDURE function_name ( arguments )
```

<https://www.postgresql.org/docs/current/triggers.html>

# Trigger: **BEFORE** ou **AFTER**

◆ O trigger dispara quando um evento específico ocorre (INSERT, DELETE ou UPDATE)

- Opção **BEFORE**: A chamada do trigger é feita **antes do evento ocorrer**
  - ◆ Tipicamente usada para verificação ou modificação dos dados antes deles serem inseridos ou atualizados
- Opção **AFTER**: É possível requisitar o disparo do trigger depois que o evento ocorreu
  - ◆ Tipicamente usada para propagar atualizações para outras tabelas ou fazer verificação de integridade com outras tabelas

# Trigger: ROW ou STATEMENT

```
CREATE TRIGGER trigger_name  
  
{BEFORE | AFTER} trigger_event ON table_name  
  
FOR EACH {ROW | STATEMENT}  
  
[WHEN trigger_condition) ]  
  
EXECUTE PROCEDURE function_name ( arguments )
```

<https://www.postgresql.org/docs/current/triggers.html>

# Trigger: ROW ou STATEMENT

- ◆ Algumas consultas SQL podem afetar várias linhas de dados. Nesses casos, o trigger pode ser chamado de duas formas:
  - Opção **ROW**: Nesse caso, o trigger procedure é executado múltiplas vezes, uma para cada linha afetada pelo evento (Insert / Delete / Update)
    - ◆ per-row trigger
    - ◆ Exemplo: uma atualização pelo comando UPDATE pode afetar diversas linhas da tabela

# Trigger: ROW ou STATEMENT

- ◆ Algumas consultas SQL podem afetar várias linhas de dados. Nesses casos, o trigger pode ser chamado de duas formas:
  - Opção **STATEMENT**: O trigger procedure é chamado somente uma vez independente do número de linhas afetadas na consulta
  - Particularmente, quando nenhuma tupla é afetada o trigger ainda é chamado
    - ◆ per-statement trigger
    - ◆ Exemplo: gerar um simples registro de auditoria proveniente de uma alteração

# Trigger: WHEN

```
CREATE TRIGGER trigger_name  
{BEFORE | AFTER} trigger_event ON table_name  
FOR EACH {ROW | STATEMENT}  
[WHEN trigger_condition) ]  
EXECUTE PROCEDURE function_name ( arguments )
```

<https://www.postgresql.org/docs/current/triggers.html>

# Trigger: WHEN

- ◆ Uma expressão booleana que determina se o trigger deve ser executado ou não
- ◆ Se a cláusula `WHEN` for especificada, a função só será chamada se a condição for verdadeira
- ◆ Nas triggers definidas com `FOR EACH ROW`, a condição especificada na cláusula `WHEN` pode referenciar colunas com valores antigos (`old`) ou novos (`new`) usando a sintaxe **`OLD.column_name`** ou **`NEW.column_name`** respectivamente



# Trigger

- ◆ Alteração
  - Recriar com o comando CREATE
- ◆ Desativação
  - ALTER TABLE table\_name DISABLE TRIGGER trigger\_name
- ◆ Ativação
  - ALTER TABLE table\_name ENABLE TRIGGER trigger\_name
- ◆ Exclusão
  - DROP TRIGGER trigger\_name ON table\_name

# Trigger - Exemplo

- ◆ Suponha que seja necessário auditar a alteração de preço quando o novo preço é inferior a 25% do preço antigo
- ◆ É necessário especificar uma condição do trigger para comparar o preço novo com o preço antigo

# Trigger - Exemplo

```
CREATE TABLE products (  
  product_id INTEGER  
    CONSTRAINT products_pk PRIMARY KEY,  
  product_type_id INTEGER ,  
  name VARCHAR(30) NOT NULL,  
  description VARCHAR(50),  
  price NUMERIC(5, 2)  
);
```

# Trigger - Exemplo

```
CREATE TABLE product_price_audit (  
    product_id INTEGER  
    CONSTRAINT price_audit_fk_products  
    REFERENCES products(product_id), *  
    old_price NUMERIC(5, 2),  
    new_price NUMERIC(5, 2)  
);
```

\* Chave estrangeira que referencia a tabela que contém os produtos

# Trigger - Exemplo

```
CREATE OR REPLACE FUNCTION audit_product_price()  
RETURNS trigger AS $$ BEGIN  
  RAISE NOTICE 'product_id = %, old_price = %, new_price =  
  %', old.product_id, old.price, new.price ;  
  RAISE NOTICE 'A redução de preço é de mais de 25  
  por cento';  
  
  -- insert row into the product_price_audit table  
  INSERT INTO product_price_audit (product_id, old_price,  
  new_price) VALUES (old.product_id, old.price, new.price);  
  RETURN NULL; -- sempre deve ter um retorno  
END $$ language 'plpgsql';  
  
CREATE TRIGGER before_product_price_update  
BEFORE UPDATE OF price ON products  
FOR EACH ROW WHEN (new.price < old.price * 0.75)  
EXECUTE PROCEDURE audit_product_price();
```

# Trigger - Exemplo

## ◆ BEFORE UPDATE OF price

- Significa que o trigger é disparado antes da atualização de price

## ◆ FOR EACH ROW

- Significa que isso é um trigger em nível de tupla, isto é, o código do trigger é executado uma vez para cada tupla atualizada

## ◆ A condição do trigger é ( $\text{new.price} < \text{old.price} * 0.75$ )

- O trigger é disparado somente quando o novo preço é menor do que 25 % do preço antigo

## ◆ Os valores novos e antigos do atributo são acessados por meio dos apelidos `old` e `new` no trigger

# Trigger - Exemplo

- ◆ per-statement (por sentença): deve retornar sempre NULL
- ◆ row-level (por linha). Duas situações: BEFORE e AFTER
  - BEFORE
    - ◆ NEW/OLD para que a operação continue normalmente
      - RETURN NEW nos casos de INSERT e UPDATE
        - OBS: você pode alterar os atributos na variável NEW
      - RETURN OLD para DELETE
    - ◆ Para cancelar a operação deve retornar NULL
      - RETURN NULL;
  - AFTER
    - ◆ o tipo retornado é ignorado, podendo então ser NULL

# Trigger - Exemplo

## ◆ Disparando o trigger

- Para disparar o trigger do exemplo anterior é necessário reduzir o preço de um produto em mais de 25%

```
UPDATE products  
SET price = price * .7  
WHERE product_id IN (5, 10);
```



# Trigger - Exemplo

## ◆ Disparando o trigger

- Par
- nece
- de 2

The screenshot shows a PostgreSQL query editor window titled "Query - postgres on postgres@localhost:5432". The window has a menu bar (File, Edit, Query, Favourites, Macros, View, Help) and a toolbar with various icons. The main area is the "SQL Editor" tab, which contains the following SQL statement:

```
UPDATE products
SET price = price * .7
WHERE product_id IN (5, 10);
```

Below the SQL editor is the "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Messages" tab is selected, showing the following output:

```
NOTA: product_id = 5, old_price = 49.99, new_price = 34.99
NOTA: A redução de preço é de mais de 25 por cento
NOTA: product_id = 10, old_price = 15.99, new_price = 11.19
NOTA: A redução de preço é de mais de 25 por cento
```

At the bottom of the output pane, it states: "Query returned successfully: 0 rows affected, 15 ms execution time."

# Trigger - Exemplo

## ◆ Disparando o trigger

- Para ver o conteúdo da tabela que guarda as informações de auditoria

```
SELECT *  
FROM product_price_audit  
ORDER BY product_id;
```

PRODUCT_ID	OLD_PRICE	NEW_PRICE
-----	-----	-----
5	49.99	34.99
10	15.99	11.19

# Trigger – Exemplo 2

- ◆ Trigger “*for each row*” para armazenar data/hora e nome do usuário que inseriu ou modificou empregado

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

# Trigger – Exemplo 2

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
BEGIN
    -- Check that empname and salary are given
    IF NEW.empname IS NULL THEN
        RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL THEN
        RAISE EXCEPTION '% cannot have null salary', NEW.empname;
    END IF;

    -- Who works for us when she must pay for it?
    IF NEW.salary < 0 THEN
        RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
    END IF;

    -- Remember who changed the payroll when
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
END;
$emp_stamp$ LANGUAGE plpgsql;

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

# Trigger – Exemplo 2

```
insert into emp values ('John Doe',1000);
```

```
select * from emp;
```

Output pane

Data Output

Explain

Messages

History

	empname text	salary integer	last_date timestamp without time zone	last_user text
1	John Doe	1000	2015-05-13 16:58:23.703	DBA

```
update emp set salary = 10000 where empname = 'John Doe';
```

```
select * from emp;
```

Output pane

Data Output

Explain

Messages

History

	empname text	salary integer	last_date timestamp without time zone	last_user text
1	John Doe	10000	2015-05-13 16:59:32.598	DBA

# Exercícios

## Modelo Relacional

FUNCIONARIO (idFunc, nome, idade, salario)

TRABALHA (idFunc (FUNCIONARIO.idFunc), idDepto (DEPARTAMENTO.idDepto), nro\_horas)

DEPARTAMENTO (idDepto, nome, orçamento, idGerente (FUNCIONARIO.idFunc), qtdeFuncionario, custoFPagamento)

# Exercícios

- ◆ Analise cada uma das restrições de integridade e verifique se
  - elas podem ser expressas em SQL (restrição de domínio, chave primária, chave estrangeira ou restrições CHECK)
  - elas precisam ser impostas por um gatilho
    - ◆ Nesse caso, indique quais operações (inserções, exclusões ou atualizações em relações específicas) devem ser monitoradas para forçar a restrição e explique

# Exercícios

1. O número total de horas/semanais que um funcionário trabalha na empresa não pode ser inferior a 20 horas e não pode ultrapassar 44 horas
2. Os funcionários devem ter um salário mínimo de R\$ 954,00
3. Todo gerente também deve ser um funcionário
4. A contratação de funcionários por um departamento não pode fazer com que o custo com a folha de pagamento exceda o orçamento do departamento



# Leitura Adicional para Casa

- ◆ **ELMASRI-NAVATHE – Sistemas de Banco de Dados**
  - **Capítulo 5 (Seção 5.2)**
- ◆ **Manual de SQL do PostgreSQL**
  - <https://www.postgresql.org/docs/current/triggers.html>

# Agenda – Apresentações projetos

Data/Horário	Grupo
01/11 – 14:50h	Vinicius Lima Carvalho (12221BCC018), Osvaldo Pagioli de Lollo Silva (12221BCC047) e Gustavo Marques Oliveira (12221BCC021)
01/11 – 15:20h	
01/11 – 15:50h	Enzo Faria de Lacerda - 12221BCC010 Bernardo Hipólito Mundim Porto 12311BCC047 Adriano Ferro de Oliveira Filho - 12221BCC026 Andreas de Castro -12311BCC067
01/11 – 16:20h	Fernanda Ferreira de Melo – 12211BCC043 Odelmo Ferreira Neto – 12211BCC006 Sthephanny Caroline da Silva Santos – 12211BCC044

Todas as apresentações serão feitas no Campus Santa Mônica Lab04  
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

# Agenda – Apresentações projetos

Data/Horário	Grupo
01/11 – 17:00h	
01/11 – 17:30h	João Gabriel Ribeiro Viana - 12411BCC099 Amanda Gomes - 12221BCC016
01/11 – 18:00h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04  
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para  
logar na máquina e deixar tudo preparado para a apresentação.

# Agenda – Apresentações projetos

Data/Horário	Grupo
07/11 – 13:20h	Marcos Paulo Gomes Pires, Lazáro Joabe, Matheus Matos
07/11 – 13:50h	Lucas Panonko Assis Belchior 12311BCC068 Lucas Cardoso Jabur 12311BCC045 Leandro Marques Gontijo Jersé 2311BCC044 Valter Fellype Ferreira Netto 12111BCC038

Todas as apresentações serão feitas no Campus Santa Mônica. **Nessa data as apresentações deverão ser feitas com computador próprio do grupo na minha sala 1B137.** Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

# Agenda – Apresentações projetos

Data/Horário	Grupo
08/11 – 14:50h	João Vitor Feijó, Jean Souto, Eduardo Cipriano
08/11 – 15:20h	
08/11 – 15:50h	
08/11 – 16:20h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04  
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para  
logar na máquina e deixar tudo preparado para a apresentação.

# Agenda – Apresentações projetos

Data/Horário	Grupo
08/11 – 17:00h	
08/11 – 17:30h	
08/11 – 18:00h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04  
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para  
logar na máquina e deixar tudo preparado para a apresentação.

# Agenda – Apresentações projetos

Data/Horário	Grupo
14/11 – 13:20h	
14/11 – 13:50h	
14/11 – 14:20h	

Todas as apresentações serão feitas no Campus Santa Mônica. **Nessa data as apresentações deverão ser feitas com computador próprio do grupo na minha sala 1B137.** Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.