



# **Disciplina: Laboratório de Banco de Dados**

## **TRIGGERS**

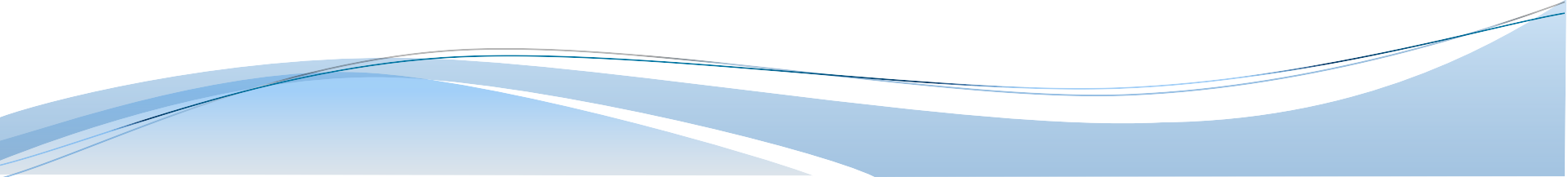


# O que é trigger

- Um “**trigger**” ou **gatilho** é um **objeto** criado no banco de dados com o objetivo de **disparar** uma **ação** em **conseqüência** de um **evento**.
- Ou seja, a cada **evento** está associado um **procedimento** que será executado.
- Podemos dizer que um “**trigger**” é um **bloco PL/pgSQL** do tipo **evento-condição-ação**.

# Para que serve um trigger



- Para **automatizar** algumas tarefas.
  - Para implementar **regras de negócio** complexas.
  - Para **replicar** tabelas, ou alguns registros, mantendo assim uma certa **redundância controlada** nos dados.
- 

# Para que serve um trigger - Exemplo

- O trigger serve para **automatizar** algumas tarefas.
- Por exemplo, quando inserimos um valor em uma tabela e desejamos que este valor seja debitado de outra tabela

Contrato

num_contr	vlr_total	vlr_devedor
00258	1200.00	1200.00

3 - AÇÃO

$\text{vlr\_devedor} = \text{vlr\_devedor} - \text{vlr\_prest}$

UPDATE

Prestação

num_contr	num_prest	Vlr_prest
00258	1	400.00

1 - EVENTO

INSERT INTO prestacao VALUES (00258, 1, 400.00)

2 - CONDIÇÃO

IF "INSERT"

# Vantagens

- **Segurança** – restringe acesso à algumas tabelas.
- **Auditoria** – permite gerar um log de acesso e modificação.
- **Replicação de Dados** – permite copiar um registro para outro lugar (tabela).
- **Integridade** – os dados continuam íntegros após modificações.
- **Controle de Dados** - Caso uma tabela tenha dados, cujo valor depende de outras tabelas, as Triggers pode atualizar automaticamente a coluna com os valores derivados.

# Componentes

- Um trigger é composto por quatro partes:
- 1. Momento** – define quando uma trigger será acionada.
    - BEFORE (ANTES) da efetivação do evento
    - AFTER (DEPOIS) da efetivação do evento
  - 2. Evento** – define qual instrução DML acionará a trigger.
    - INSERT
    - UPDATE
    - DELETE

# Componentes

- Um trigger é composto por quatro partes:

## 3. **Tipo** – define quantas vezes a ação será executada

- ROW
  - Para cada evento (INSERT, UPDATE ou DELETE) será disparada a TRIGGER
  - Pode-se utilizar as variáveis NEW, OLD para referenciar os campos da tabela
- STATEMENT
  - Será disparada apenas uma vez a TRIGGER, mesmo que ocorra eventos de atualizações de mais de uma tupla
  - Não é possível referenciar campos da tabela (variáveis NEW, OLD)

## 4. **Corpo (ação)** – define o procedimento que será executado, no caso uma function que retorna TRIGGERS

# Sintaxe

CREATE TRIGGER *name*

{ BEFORE | AFTER } → momento

{ *event* [ OR ... ] } ON *table* → evento

[ FOR [ EACH ] { ROW | STATEMENT } ] → tipo

EXECUTE PROCEDURE *funcname* ( *arguments* ) → ação



# Exemplo

- Criar um gatilho para atualizar o contrato toda vez que uma prestação por paga.

```
CREATE TRIGGER tr_exemplo
```

```
BEFORE
```

```
INSERT ON prestacao
```

```
FOR EACH ROW
```

```
EXECUTE PROCEDURE fn_update_contrato (num_contr )
```

momento

evento

tipo

ação

# Recursos de triggers

- Para criar um trigger primeiro devemos criar a função que depois será associada com o trigger.
- Uma função que será executada pela chamada de um trigger deve ter o tipo de retorno **TRIGGER**.
- Como toda função deve retornar um valor, uma função desenvolvida para um trigger deve retornar um valor também. Esse valor pode ser:
  - **NULL**.
  - Ou um **registro** da linha da tabela em que o evento ocorreu.

# Recursos de Trigger - variáveis especiais

- Algumas variáveis especiais:
- OLD – tipo de dado RECORD. Armazena os dados do registro que esta sendo removido (delete) ou atualizado (update).
- NEW – tipo de dado RECORD. Armazena os dados do registro que esta sendo inserido (insert) ou atualizado (update).
- TG\_OP – o tipo de operação realizada. (UPDATE, INSERT, DELETE)

# Exemplo 01 – Auditoria

**Empregado**

nome	salario
"Fulano"	1000.00

**Emp\_log**

Dat_update	userid	salario
05-05-2000	5	1000.00

```
CREATE OR REPLACE FUNCTION fn_audit_emp() RETURNS TRIGGER AS $$  
BEGIN  
    INSERT INTO emp_log VALUES (now(), userid, new.salario);  
    RETURN NEW;  
END;  
$$ language 'plpgsql';
```

```
CREATE TRIGGER tr_log_emp  
BEFORE INSERT OR UPDATE ON empregado  
FOR EACH ROW  
EXECUTE PROCEDURE fn_audit_emp()
```

# Exemplo 02 – Auditoria2

## Empregado

nome	salario
"Fulano"	1000.00

## Emp\_log

operation	Dat_update	userid	salario
INSERT	05-05-2009	5	1000.00

# Exemplo 02 – Auditoria2 (CONTINUAÇÃO)

```
CREATE OR REPLACE FUNCTION fn_audit_emp() RETURNS TRIGGER AS $$  
BEGIN  
    IF (TG_OP = 'DELETE') THEN  
        INSERT INTO emp_log VALUES ('D', now( ), user, OLD.*);  
        RETURN OLD;  
  
    ELSIF (TG_OP = 'UPDATE') THEN  
        INSERT INTO emp_log values ('U', now( ), user, NEW.*);  
        RETURN NEW;  
  
    ELSIF (TG_OP = 'INSERT') THEN  
        INSERT INTO emp_log VALUES( 'I', now( ), user, NEW.*)  
        RETURN NEW;  
  
    END IF;  
    RETURN NULL;  
END;  
$$language 'plpgsql';
```

## Exemplo 02 – Auditoria2

- Continuação...

```
CREATE TRIGGER tr_log_emp  
BEFORE INSERT OR UPDATE OR DELETE ON empregado  
FOR EACH ROW  
EXECUTE PROCEDURE fn_audit_emp()
```

# ATENÇÃO ! CUIDADO !

- O uso de trigger deve ser muito bem controlado e definido.
- O uso inadequado de trigger pode disparar uma série de triggers.
  - LOOP INFINITO
  - Recursões implícitas
- Portanto faça o uso da instrução **RAISE EXCEPTION** sempre que possível para identificar quaisquer problemas