



AS33C

Banco de Dados 2

Professor: Eduardo Cotrin Teixeira



cotrin@utfpr.edu.br



SQL – Triggers

- A **linguagem procedural** que permite a definição de **procedimentos armazenados**, também permite a criação de **regras ativas**.
- Regras ativas são úteis para:
 - Definição de regras de negócio
 - Especificação de restrições de integridade não possíveis no modelo relacional
 - Cálculo de atributos derivados
 - Auditoria
 - Adição de funcionalidades ao banco



SQL – Triggers

- Regras ativas são implementadas por meio de **gatilhos (*triggers*)**.
- *Triggers* seguem o paradigma E-C-A:

ECA = Evento-Condição-Ação

*quando o **evento** ocorre,
se a **condição** é satisfeita,
Então a **ação** é executada*



SQL – Triggers

- Regras ECA reagem de forma autônoma a eventos que ocorrem sobre os dados.
- Gatilhos disparam execuções em função de *eventos* que ocorrem.
- Um evento ocorre:
 - ✓ Em uma tabela
 - ✓ De acordo com uma operação (INSERT, UPDATE ou DELETE)
 - ✓ Antes ou depois da operação (AFTER ou BEFORE)



SQL – Triggers

CREATE TRIGGER <nome>{BEFORE|AFTER|INSTEAD OF}{evento [OR...] }
ON tabela [FOR EACH {ROW|STATEMENT}] **EXECUTE PROCEDURE** função();

before | after | instead of determina se a função será chamada antes, depois ou em substituição ao evento.

evento indica a que evento o disparo da trigger está vinculado, e pode ser DELETE, UPDATE ou INSERT.

tabela indica a qual tabela a trigger estará associada.

row | statement especifica se a trigger deve ser disparada uma vez para cada linha afetada (row) ou uma vez por comando SQL (statement - padrão).



SQL – Triggers

■ Exemplos

```
CREATE TRIGGER Exemplo1 BEFORE UPDATE ON Projeto  
FOR EACH ROW EXECUTE PROCEDURE Exemplo1();
```

```
CREATE TRIGGER Exemplo2 AFTER INSERT OR UPDATE OF  
Quant ON Fornece_Para FOR EACH ROW EXECUTE PROCEDURE  
Exemplo2();
```

```
CREATE TRIGGER Exemplo3 AFTER INSERT OR DELETE OR  
UPDATE ON Projeto FOR EACH STATEMENT EXECUTE  
PROCEDURE Exemplo3();
```

Obs.: No PostgreSQL não é possível usar '**OR REPLACE**' na criação de uma trigger (ela é criada como uma *constraint*). Para refazer a trigger ela deve ser excluída: **DROP TRIGGER <nome_trigger> ON <tabela>;**



SQL – Triggers

- Antes ou depois da operação :

BEGIN *--TRANSAÇÃO*

TRIGGER BEFORE

INSERT / UPDATE / DELETE

TRIGGER AFTER

COMMIT/ROLLBACK *--TRANSAÇÃO*

***Se a trigger causar o disparo de uma exceção a operação é cancelada (ROLLBACK),
mesmo para triggers AFTER***



SQL – Triggers

- **Funções de triggers (...EXECUTE PROCEDURE...)**
 - Uma função de trigger deve ser declarada como uma função que **não recebe argumentos** e que **retorna o tipo *trigger***.
 - O **valor de retorno** de uma função de trigger deve ser **null** ou um registro NEW ou OLD que veremos em breve.
 - * As funções de trigger chamadas com FOR EACH STATEMENT devem sempre retornar NULL.



SQL – Triggers

*** Informar limite de custo dos projetos:**

```
CREATE TRIGGER CustProjeto AFTER INSERT OR UPDATE ON  
Projeto FOR EACH STATEMENT EXECUTE PROCEDURE CustProjeto();
```

*** Função:**

```
CREATE OR REPLACE FUNCTION CustProjeto() RETURNS TRIGGER AS $$  
BEGIN  
    IF EXISTS (SELECT PCusto FROM Projeto WHERE Pcusto > 50000) THEN  
        RAISE NOTICE 'Há projeto(s) com custo maior que 50000 !';  
    END IF;  
    RETURN NULL;  
END; $$ LANGUAGE plpgsql;
```



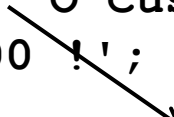
SQL – Triggers

* Manter média de custo para projetos:

```
CREATE TRIGGER MediaProjeto AFTER INSERT OR DELETE OR UPDATE ON
Projeto FOR EACH STATEMENT EXECUTE PROCEDURE MediaProjeto();
```

* Função:

```
CREATE OR REPLACE FUNCTION MediaProjeto() RETURNS TRIGGER AS $$
DECLARE Media FLOAT;
BEGIN
    Media := (SELECT AVG(PCusto) FROM Projeto);
    IF Media > 35000 THEN RAISE EXCEPTION 'O Custo médio dos
        projetos não pode ultrapassar 35.000 !';
    END IF;
    RETURN NULL;
END; $$ LANGUAGE plpgsql;
```



**Com a exceção, a operação
(INSERT, UPDATE ou
DELETE) não é realizada !**



SQL – Triggers

* Manter LOG (histórico) - Diferença entre ROW e STATEMENT:

```
CREATE TRIGGER LogProjeto AFTER UPDATE ON Projeto FOR EACH  
STATEMENT EXECUTE PROCEDURE LogProjeto(); --1 LOG POR COMANDO  
--OU
```

```
CREATE TRIGGER LogProjeto AFTER UPDATE ON Projeto FOR EACH ROW  
EXECUTE PROCEDURE LogProjeto(); --1 LOG POR LINHA
```

* Função:

```
CREATE TABLE LOGPROJETO (ID SERIAL PRIMARY KEY, DATA TIMESTAMP);  
CREATE OR REPLACE FUNCTION LogProjeto() RETURNS TRIGGER AS $$  
BEGIN  
    INSERT INTO LOGPROJETO (DATA) VALUES (CURRENT_TIMESTAMP);  
    RETURN NULL;  
END; $$ LANGUAGE plpgsql;  
--TESTE: UPDATE PROJETO SET PCUSTO=PCUSTO*1.1;
```



SQL – Triggers

*** Manter custo total dos projetos:**

```
CREATE TRIGGER CustoTotal AFTER INSERT OR UPDATE OR DELETE ON  
Projeto FOR EACH STATEMENT EXECUTE PROCEDURE CustoTotal();
```

*** Função:**

```
CREATE TABLE CUSTOTOTAL(TOTAL FLOAT PRIMARY KEY);  
CREATE OR REPLACE FUNCTION CustoTotal() RETURNS TRIGGER AS $$  
BEGIN  
    IF EXISTS (SELECT * FROM CUSTOTOTAL) THEN  
        UPDATE CUSTOTOTAL SET TOTAL=(SELECT SUM(PCUSTO) FROM PROJETO);  
    ELSE  
        INSERT INTO CUSTOTOTAL SELECT SUM(PCUSTO) FROM PROJETO;  
    END IF;  
    RETURN NULL;  
END; $$ LANGUAGE plpgsql;
```