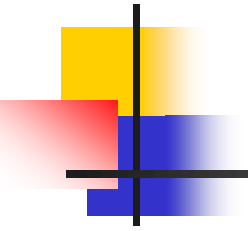
Universidade Tecnológica
Federal do Paraná
UTFPR - Campus Cornélio Procópio

AS33C
Banco de Dados 2

Professor: Eduardo Cotrin Teixeira

cotrin@utfpr.edu.br



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

- O PostgreSQL disponibiliza duas variáveis importantes para serem usadas com as triggers: **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.

SQL – Triggers

- **Exemplo** - Função para forçar limite de custo para projetos:

```
CREATE OR REPLACE FUNCTION LimitaCusto() RETURNS TRIGGER AS $$
BEGIN
   IF (NEW.PCusto > 50000) THEN NEW.PCusto := 50000; END IF;
   RETURN NEW; -- trigger BEFORE / FOR EACH ROW
   -- Se retorno for NULL, a inserção não é realizada!
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER LimitaCusto BEFORE INSERT ON Projeto
FOR EACH ROW EXECUTE PROCEDURE LimitaCusto();

insert into projeto values ('P10','Teste',10,60000);
-- será inserido P10 com custo de 50.000.
```

Depois (AFTER) não tem efeito !

SQL – Triggers

- **Exemplo** - Para forçar limite de custo na ATUALIZAÇÃO:

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

* 2 Triggers usam a mesma função. Poderia ser também uma
      trigger só disparada pelos dois eventos.

update Projeto set PCusto=60000 where PNro='P10';
-- P10 será atualizado com custo de 50.000.
```

SQL – Triggers

- **Exemplo** - Função para **rejeitar** atualização de custo:

```
CREATE OR REPLACE FUNCTION RejeitaCusto() RETURNS TRIGGER AS $$
BEGIN
   IF (NEW.PCusto > 50000) THEN NEW.PCusto := OLD.PCusto;END IF;
   RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER RejeitaCusto BEFORE UPDATE ON Projeto
FOR EACH ROW EXECUTE PROCEDURE RejeitaCusto();
--drop trigger limitacustoupdate on projeto; --apaga a trigger anterior que LIMITA o valor para substituir por essa que REJEITA
update Projeto set PCusto=30000 where PNro='P10'; --OK
update Projeto set PCusto=70000 where PNro='P10'; --Fica o anterior
```

SQL – Triggers

- **Exemplo** - Na inserção de um fornecedor, avisar se já existe na mesma cidade fornecedor de categoria igual ou mais alta:

```
CREATE OR REPLACE FUNCTION CategFornec()RETURNS TRIGGER AS $$
BEGIN
 IF EXISTS (SELECT * FROM Fornecedor Where FCidade=NEW.FCidade
             AND FCateg <= NEW.FCateg)
  THEN RAISE NOTICE 'Há fornecedor de categoria mais alta!';
 END IF;
 RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER CategFornec BEFORE INSERT ON Fornecedor
FOR EACH ROW EXECUTE PROCEDURE CategFornec();

insert into Fornecedor values ('F6','Teste','Campinas','C');
```