

TRIGGERS (GATILHOS)



Bruno A. N. Travençolo – FACOM

TRIGGERS

- ▶ Triggers são ***store procedures*** que são **executadas automaticamente** (disparam).
- ▶ “Disparam” quando ocorre um evento INSERT, UPDATE ou DELETE numa tabela.
 - ▶ Os *store procedures* que criamos até agora eram chamados explicitamente utilizando o comando SELECT. O trigger é chamado pelo SGBD.
- ▶ Geralmente são usados para reforçar restrições (*constraints*) de integridade que não podem ser tratadas pelos recursos mais simples, como valores *defaults*, restrições *pk* e *fk*, *NOT NULL*, *check()*, etc.



TRIGGERS

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



Definindo um Trigger Procedure

- ▶ O **trigger procedure** é muito similar a um *store procedure*, mas é um pouco mais restrito devido à maneira como é chamada
 - ▶ Não possui parâmetros de entrada na função
 - ▶ Mas pode receber dados de entrada (ver slides à frente)
 - ▶ Deve retornar o tipo especial 'trigger'
- ▶ Exemplo: criar um trigger que garanta que os salários dos empregados não possam ser reduzidos e sejam positivos

```
CREATE FUNCTION restringe_reducao() -- Sem parâmetros de entrada
RETURNS trigger AS -- Retorna tipo trigger
$$
BEGIN
    -- Aqui entra o código que não vai deixar reduzir o salário que vai chamar:
    RAISE EXCEPTION 'Salário não pode ser reduzido';
END; $$ language plpgsql;
```



TRIGGERS

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



TRIGGERS

- ▶ Sintaxe

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

- ▶ Deletando um trigger: deve-se informar o nome da tabela a qual ele está associado

```
DROP TRIGGER nome ON tabela;
```



TRIGGERS - Evento

- ▶ Sintaxe

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

<evento> pode ser INSERT, DELETE ou UPDATE

- ▶ 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 )
```



TRIGGERS – BEFORE ou AFTER

► Sintaxe

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

- 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 de eles serem inseridos ou atualizados



TRIGGERS – BEFORE ou AFTER

► Sintaxe

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

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



TRIGGERS – ROW ou STATEMENT

▶ Sintaxe

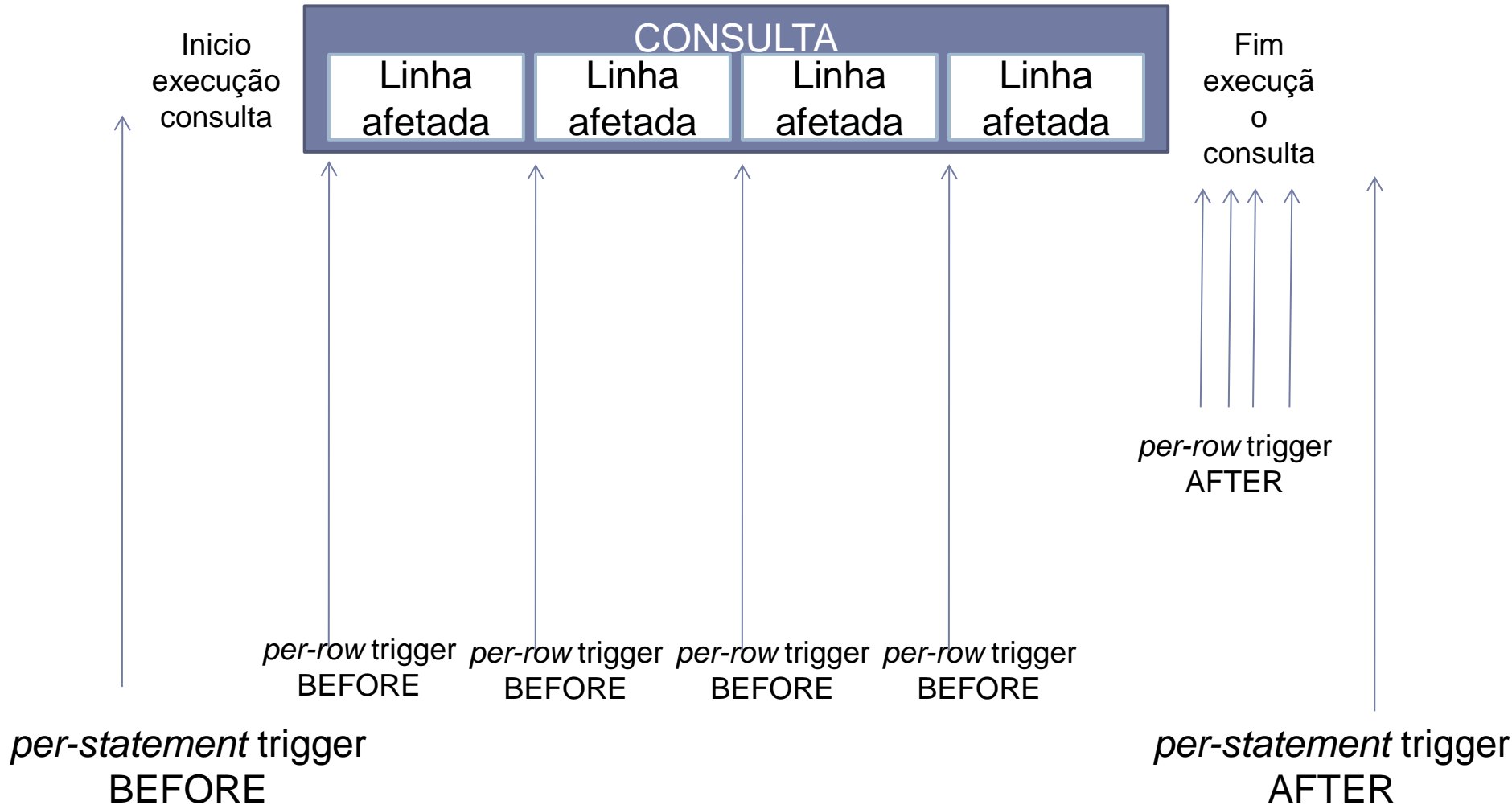
```
CREATE TRIGGER nome { BEFORE | AFTER }  
{ <evento> [OR ...] }  
ON tabela FOR EACH { ROW | STATEMENT }  
EXECUTE PROCEDURE função ( argumentos )
```

▶ 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
- ▶ 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



CHAMADA DO TRIGGER



Exemplo de sequência de chamada dos triggers

- ▶ Código disponível no moodle (seq_trigger.sql)

	tempo timestamp without time zone	swhen character varying	slevel character varying
1	2010-11-05 10:46:55.879	BEFORE	STATEMENT
2	2010-11-05 10:46:57.055	BEFORE	ROW
3	2010-11-05 10:46:58.249	BEFORE	ROW
4	2010-11-05 10:46:59.451	BEFORE	ROW
5	2010-11-05 10:47:00.611	AFTER	ROW
6	2010-11-05 10:47:01.747	AFTER	ROW
7	2010-11-05 10:47:02.882	AFTER	ROW
8	2010-11-05 10:47:04.015	AFTER	STATEMENT

Exemplo

CREATE FUNCTION restringe_reducao() -- Sem parâmetros de entrada

RETURNS trigger **AS** -- Retorna tipo trigger

\$\$

BEGIN

-- Aqui entra o código que não vai deixar reduzir o salário que vai chamar:

RAISE EXCEPTION 'Salário não pode ser reduzido';

END; **\$\$** language plpgsql;

CREATE TRIGGER *check_salario* **BEFORE**

INSERT OR UPDATE

ON *empregado* **FOR EACH ROW**

EXECUTE PROCEDURE restringe_reducao()



Disparo

- ▶ O **trigger procedure** terá acesso aos dados originais (no caso de UPDATE e DELETE) e ao novo dado (para INSERT e UPDATE) por meio de variáveis
- ▶ Variáveis
 - ▶ OLD para os dados originais
 - ▶ Ex: OLD.nome; OLD.salario
 - ▶ NEW para os novos dados
 - ▶ Ex: NEW.salario, NEW.idade
- ▶ Obs: Somente triggers *per-row* são associados às variáveis NEW e OLD



Variáveis para Triggers

Table 10-12. *PostgreSQL Trigger Procedure Variables*

Variable	Description
NEW	A record containing the new database row
OLD	A record containing the old database row
TG_NAME	A variable containing the name of the trigger that fired and caused the trigger procedure to run
TG_WHEN	A text variable containing the text 'BEFORE' or 'AFTER', depending on the type of the trigger
TG_LEVEL	A text variable containing 'ROW' or 'STATEMENT', depending on the trigger definition
TG_OP	A text variable containing 'INSERT', 'DELETE', or 'UPDATE', depending on the event that occurred resulting in this trigger being fired
TG_RELID	An object identifier representing the table the trigger has been activated upon
TG_RELNAME	The name of the table that the trigger has been fired upon
TG_NARGS	An integer variable containing the number of arguments specified in the trigger definition
TG_ARGV	An array of strings containing the procedure parameters, starting at zero; invalid indexes return NULL values



Exemplo

CREATE LANGUAGE plpgsql

/* Criar uma uma tabela de histórico de mudanças de edição em Hist_LIVRO */

```
CREATE TABLE HIST_LIVRO(  
    TITULO VARCHAR(30) NOT NULL,  
    COD_AUTOR NUMBER(3) NOT NULL,  
    COD_EDITORA NUMBER(3) NOT NULL,  
    VALOR NUMBER(7,2),  
    COMENTARIO TEXT,  
    PUBLICACAO DATE,  
    EDICAO NUMBER(2) NOT NULL,  
    CONSTRAINT CHAVEHIST_LIVRO PRIMARY KEY  
        (TITULO, COD_AUTOR, VOLUME),  
    CONSTRAINT HIST_LIVROINHERITAUTOR FOREIGN KEY  
        (COD_AUTOR) REFERENCES AUTOR,  
    CONSTRAINT HIST_LIVROINHERITEDITORA FOREIGN KEY  
        (COD_EDITORA) REFERENCES EDITORA);
```



Exemplo

```
CREATE LANGUAGE plpgsql
```

```
CREATE FUNCTION check_edicao() RETURNS trigger AS
```

```
$$
```

```
BEGIN
```

```
IF (OLD.EDICAO <> NEW.EDICAO) THEN
```

```
    INSERT INTO HIST_LIVRO
```

```
    (TITULO, COD_AUTOR, COD_EDITORA, VALOR,  
     COMENTARIO, PUBLICACAO, EDICAO)
```

```
    VALUES (OLD.TITULO, OLD.COD_AUTOR,  
             OLD.COD_EDITORA, OLD.VALOR,  
             OLD.COMENTARIO, OLD.PUBLICACAO,  
             OLD.EDICAO);
```

```
    RETURN NULL; // sempre deve ter um retorno.
```

```
END IF;
```

```
END; $$ language plpgsql;
```

```
CREATE TRIGGER TRIG_HIST_LIVRO AFTER UPDATE  
ON livro FOR EACH ROW  
EXECUTE PROCEDURE check_edicao()
```



Qual o tipo retornado em uma trigger procedure?

```
CREATE LANGUAGE plpgsql
```

```
CREATE FUNCTION check_edicao() RETURNS trigger AS
```

```
$$
```

```
BEGIN
```

```
IF (OLD.EDICAO <> NEW.EDICAO) THEN
```

```
    INSERT INTO HIST_LIVRO
```

```
    (TITULO, COD_AUTOR, COD_EDITORA, VALOR,  
     COMENTARIO, PUBLICACAO, EDICAO)
```

```
    VALUES (OLD.TITULO, OLD.COD_AUTOR,  
             OLD.COD_EDITORA, OLD.VALOR,  
             OLD.COMENTARIO, OLD.PUBLICACAO,  
             OLD.EDICAO);
```

```
    RETURN NULL; // sempre deve ter um retorno.
```

```
END IF;
```

```
END; $$ language plpgsql;
```

```
CREATE TRIGGER TRIG_HIST_LIVRO AFTER UPDATE  
ON livro FOR EACH ROW  
EXECUTE PROCEDURE check_edicao()
```

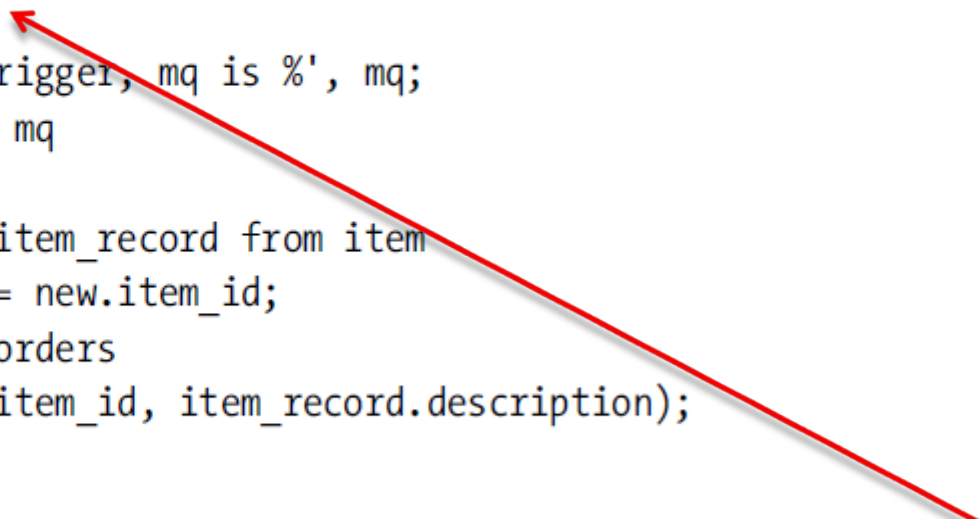


Tipo retornado na trigger procedure

- ▶ *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

Exemplo 2

```
create function reorder_trigger() returns trigger AS $$
declare
    mq integer;
    item_record record;
begin
    mq := tg_argv[0];
    raise notice 'in trigger, mq is %', mq;
    if new.quantity <= mq
    then
        select * into item_record from item
        where item_id = new.item_id;
        insert into reorders
            values (new.item_id, item_record.description);
    end if;
    return NULL;
end;
$$ language plpgsql;
```



```
CREATE TRIGGER trig_reorder
AFTER INSERT OR UPDATE ON stock
FOR EACH ROW EXECUTE PROCEDURE reorder_trigger(3);
```