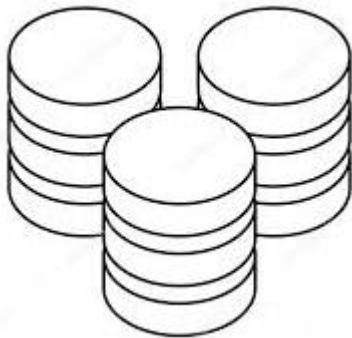


## Banco de Dados II

### Triggers

2ª parte



Profa. Damires Souza  
damires@ifpb.edu.br

# Trigger

---

- Um **trigger ou gatilho** no PostgreSQL consiste em duas partes:

- Uma função de gatilho
- O gatilho real que chama a função de gatilho

**\*\* Vantagem:** as funções de gatilho podem ser reutilizadas: gatilhos em tabelas diferentes podem usar a mesma função de gatilho.

```
CREATE OR REPLACE FUNCTION geralogemp()  
RETURNS trigger AS $$  
BEGIN  
    INSERT INTO empaudit (matemp, dataalter)  
    VALUES (new.matricula, current_timestamp);  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER logEmptrigger  
AFTER INSERT ON empregado  
FOR EACH ROW  
EXECUTE PROCEDURE geralogEmp();
```

# Níveis

---













- **Statement Level:** o trigger só dispara uma vez por comando
- **Row Level:** o trigger dispara uma vez por linha de dados que o comando envolve
- Exemplos:
  - **Row trigger + after update**
    - Executado uma vez para cada linha afetada pelo update. Se o update afeta 20 tuplas, o trigger é executado 20 vezes.
  - **Statement trigger + after update**
    - O trigger é executado apenas uma vez.

# Exemplo 3.1

```
Select * from departamento;  
CREATE OR REPLACE FUNCTION verificaop_Depto() RETURNS  
TRIGGER AS $$  
BEGIN  
    -- Utiliza a variável TG_OP para descobrir a operação sendo realizada.  
    IF (TG_OP = 'DELETE') THEN  
        raise notice 'Operação Delete sobre %', TG_TABLE_NAME;  
        RETURN null;  
    ELSIF (TG_OP = 'UPDATE') THEN  
        raise notice 'Operação Update sobre %', TG_TABLE_NAME;  
        RETURN null;  
    ELSIF (TG_OP = 'INSERT') THEN  
        raise notice 'Operação Insert sobre %', TG_TABLE_NAME;  
        RETURN null;  
    END IF;  
END;  
$$ language plpgsql;
```

# Exemplo 3.1

```
CREATE TRIGGER
  TestaDepto_audit
  AFTER INSERT OR UPDATE OR
  DELETE ON departamento
  FOR EACH ROW EXECUTE
  PROCEDURE verificaop_Depto();
```

- ▼  departamento
  - ▼  Columns (3)
    -  coddepto
    -  nome
    -  local
  - >  Constraints
  - >  Indexes
  - >  RLS Policies
  - >  Rules
  - ▼  Triggers (1)
    - ▼  testadepto\_audit
      -  verificaop\_depto()

# Exemplo 3.1

```
select * from departamento;
```

```
Insert into departamento values (8,'Compras','Sede');  
update departamento set local = 'Patos' where coddepto = 8;  
delete from departamento where coddepto = 8;
```

Data Output	Explain	<u>Messages</u>	Notifications
-------------	---------	-----------------	---------------

NOTICE: Operação Insert sobre departamento			
INSERT 0 1			

Query returned successfully in 101 msec.

# Exemplo 3.2

---

```
update departamento  
set local = 'Outro';
```



Data Output	Explain	<u>Messages</u>	Notifications
NOTICE:	Operação Update sobre departamento		
NOTICE:	Operação Update sobre departamento		
NOTICE:	Operação Update sobre departamento		
NOTICE:	Operação Update sobre departamento		
UPDATE	4		

# Exemplo 3.3

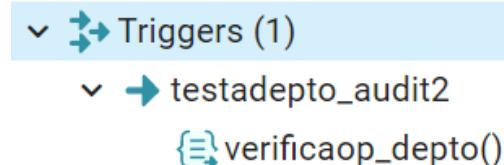
**drop trigger TestaDepto\_audit on departamento;**

**-- Retire o “for each row”**

**CREATE TRIGGER TestaDepto\_audit2 AFTER INSERT OR  
UPDATE OR DELETE ON Departamento  
EXECUTE PROCEDURE verificaop\_Depto();**

update departamento  
set local = 'teste';

**O que aconteceu?**



Data Output	Explain	Messages	Notifications
NOTICE: Operação Update sobre departamento			
UPDATE 4			



# Trigger

---

- Valor de retorno de uma função de trigger
  - Funções são declaradas como “RETURNS Trigger”
  - Retorno será:
    - para gatilhos de nível de instrução (statement), o valor **NULL**
    - para gatilhos de nível de linha (row), uma linha da tabela na qual o gatilho é definido
- O valor de retorno é ignorado para gatilhos AFTER de nível de linha => pode ser **NULL**

# Trigger

---

- No nível de linha (row) BEFORE:
  - Se o gatilho retornar NULL, a operação de disparo será abortada e a linha não será modificada
  - Para gatilhos INSERT e UPDATE, a linha retornada é a entrada para a instrução DML de gatilho

# Instead Of Trigger

---

- Uma **view** pode se tornar **atualizável** por meio do **INSTEAD OF trigger**
  - Esses triggers provêem, de forma transparente, a execução da atualização



# Exemplo 4

Create or replace view **vPessoas** as  
select nome as nome, 'c' as tipo  
from cliente

Union

Select nome, 'v' from vendedor;

Select \* from vPessoas;

	Data Output	Explain	Messages	Not
	<b>nome</b> character varying (30)		<b>tipo</b> text	
1	Marcia Araruna2		c	
2	Amelia Paulo		v	
3	Maria Portela		c	
4	Carla Gomes		v	
5	Augusto Paulo		v	
6	Alicia Silva		v	
7	Joao Peregrino		v	

# Exemplo 4

```
CREATE or replace FUNCTION insere_view_vPessoas()
  RETURNS trigger AS $$
  Declare v_cod_vend integer;
          v_cod_cli integer;
Begin
  Select max(codvend)+1 into v_cod_vend from vendedor;
  Select max(codcli)+1 into v_cod_cli from cliente;
  If new.tipo = 'c' then
    Insert into cliente(codcli, nome) values (v_cod_cli, new.nome);
  Else
    Insert into vendedor (codvend,nome) values
(v_cod_vend,new.nome);
  End if;
  Return null;
END;
$$ language plpgsql;
```

# Exemplo 4

Create trigger **insViewVPessoas**  
**Instead of insert on vPessoas**  
for each row  
execute procedure insere\_view\_vPessoas();

select \* from vendedor;  
select \* from cliente;  
select \* from vPessoas order by tipo;

insert into vPessoas values('Mercia','v');  
insert into vPessoas values('Catarina','c');

6	Joaquim Moraes	c
7	Catarina	c
13	Mercia	v
14	Juan Gomes	v
15	Amelia2 Paulo	v

# Exemplo 5

```
create table tabClienteaudit(atualizacao integer, ultimadata date, quem  
varchar);
```

```
select * from tabClienteaudit;
```

```
CREATE OR REPLACE FUNCTION registra_upd_cliente() RETURNS  
TRIGGER AS $$
```

```
    declare qtd_linhas integer;
```

```
Begin
```

```
    select count(*) into qtd_linhas from tabClienteaudit;
```

```
    if qtd_linhas = 0 then insert into tabclienteAudit values(1,current_date,  
current_user);
```

```
    else Update tabClienteAudit
```

```
        Set atualizacao = atualizacao + 1, ultimadata = current_date, quem =  
current_user;
```

```
    end if;
```

```
    return null;
```

```
End;
```

```
$$ language plpgsql;
```

# Exemplo 5

```
CREATE TRIGGER cliente_audit AFTER UPDATE ON
cliente FOR EACH ROW
EXECUTE PROCEDURE registra_upd_cliente();
```

```
select * from cliente;
update cliente set cidade = 'João Pessoa' Where codcli = 1;
Select * from tabclienteAudit;
```

	Data Output	Explain	Messages	Notificat
	atualizacao integer	ultimadata date	quem character varying	
1	1	2022-10-24	postgres	



```
CREATE OR REPLACE FUNCTION impedeAlteração() RETURNS  
  TRIGGER AS $$  
Declare  
  v_hoje integer;  
  v_agora integer;  
Begin  
  v_hoje := to_number(to_char(current_timestamp,'d'),'99');  
  v_agora := to_number(to_char(current_timestamp,'hh24mi'),'9999');  
  If v_agora > 1330 then  
    RAISE EXCEPTION '%', 'Hora proibida para atualizações' USING  
    ERRCODE = 45000;  
  End if;  
  If v_hoje = 1 then  
    RAISE EXCEPTION '%', 'Dia proibido para atualizações' USING  
    ERRCODE = 45001;  
  End if;  
  return new;  
End;  
$$ language plpgsql;
```

# Exemplo 6

---

```
CREATE TRIGGER empnotifTrig  
BEFORE INSERT OR UPDATE ON empregado  
FOR EACH ROW EXECUTE PROCEDURE  
impedeAlteração();
```

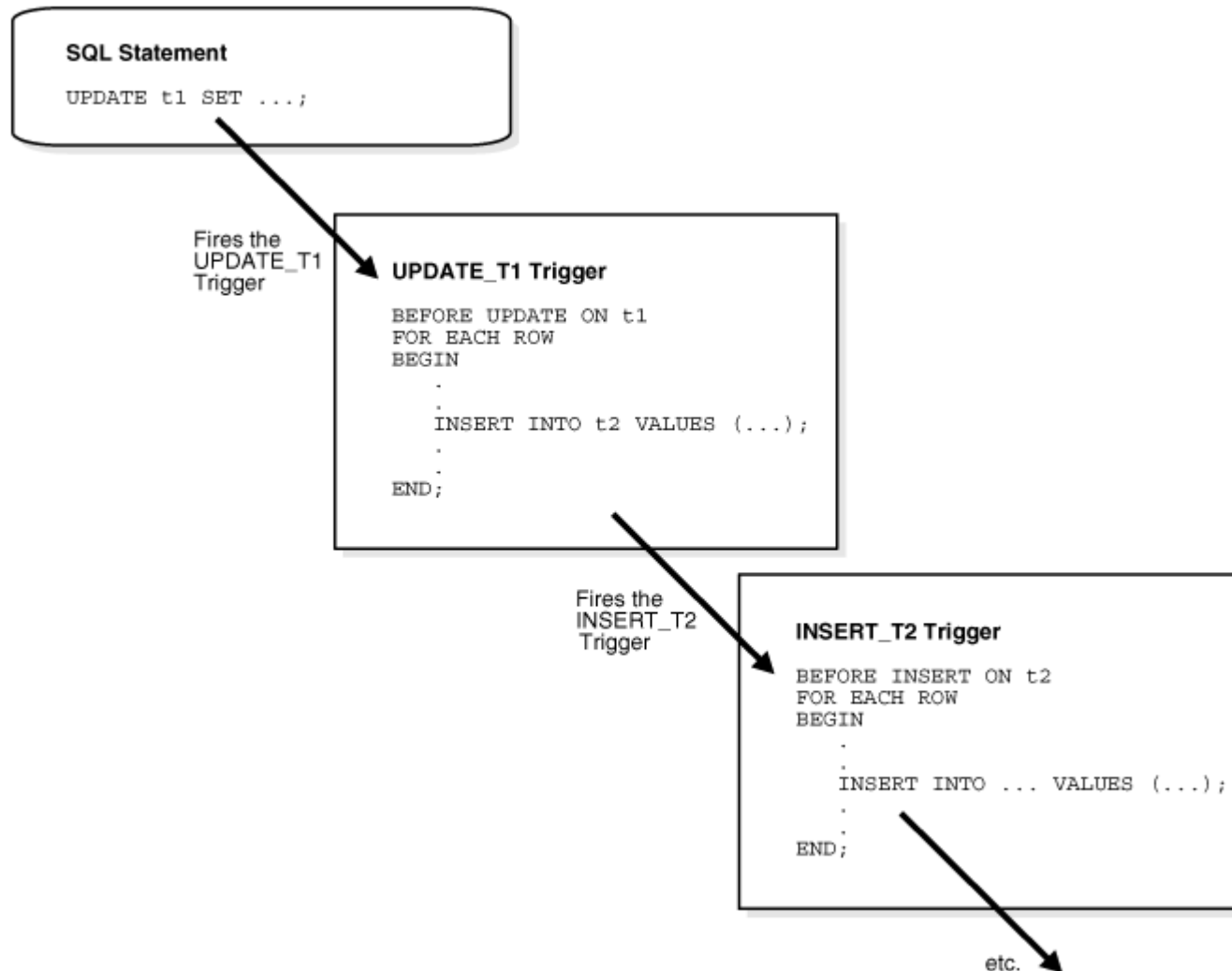
```
INSERT INTO empregado VALUES (8,'TESTE','Morais','12-03-  
2020','Gerente',10000,1,1);
```

```
select * from empregado;  
select current_timestamp;
```

```
select * from empregado order by primeironome;
```

# Cuidado com Efeito Cascata

---



# Triggers x Constraints

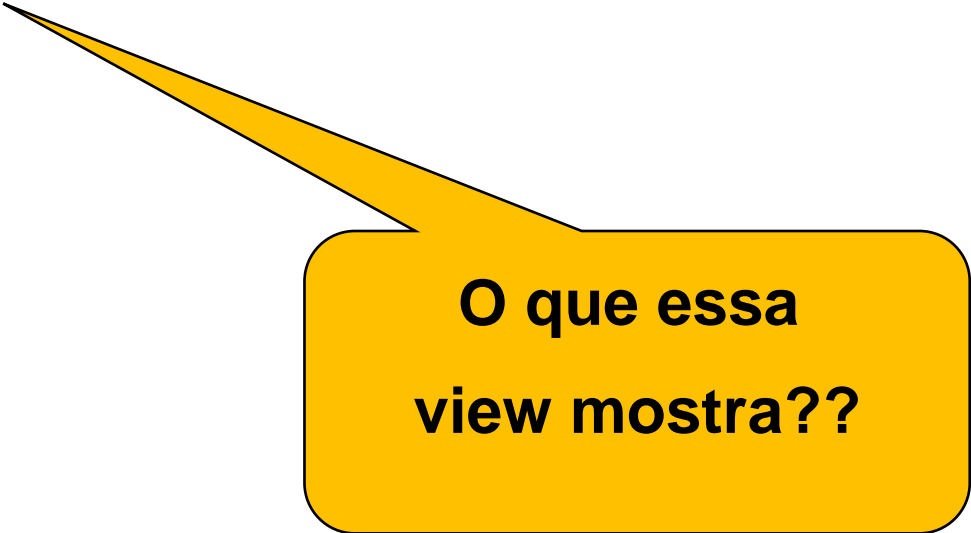
---

- Use triggers para realizar restrições sobre dados quando:
  - For necessário **reforçar integridade referencial**
    - Ex: quando tabelas mãe e filhas estão em nós diferentes (**bd distribuídos**)
  - Para otimizar **regras de negócio mais complexas** impossíveis de serem realizadas por meio das seguintes constraints:
    - NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY
    - CHECK, DELETE CASCADE, DELETE SET NULL
  - **Algumas operações de administração e de suporte**

# Verifique

---

```
SELECT * FROM  
INFORMATION_SCHEMA.TRIGGERS
```



**O que essa  
view mostra??**

<https://www.postgresql.org/docs/current/plpgsql-trigger.html>