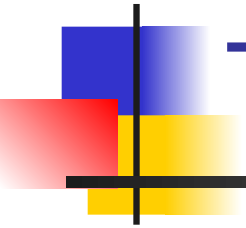
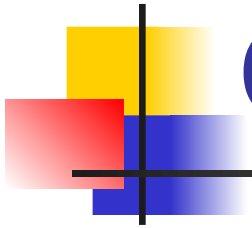


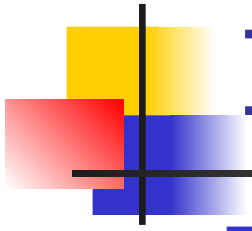
Triggers (Gatilhos)

Profa. Dra Andréia R. Casare
E-mail: casareandreia@gmail.com
andreia.casare01@fatec.sp.gov.br



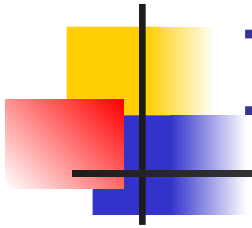
O que são Triggers (Gatilhos)?

- Um trigger é parecido com um Stored Procedure, mas com uma diferença principal: ele dispara automaticamente quando certos tipos de eventos acontecem.
- Esses eventos são baseados em tabelas e linhas; por exemplo, você pode definir um trigger que é invocado depois que uma linha é atualizada.



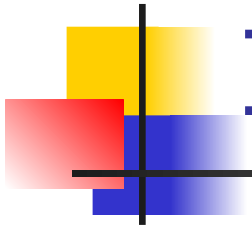
Introdução

- Uma função de gatilho pode ser criada para executar antes (BEFORE) ou após (AFTER) de um INSERT, UPDATE OU DELETE, uma vez para cada registro (linha) modificado ou por instrução SQL.
- Logo que ocorre um desses eventos do gatilho a função do gatilho é disparada automaticamente para tratar o evento.



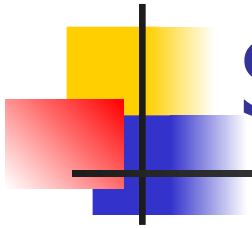
Introdução

- Após criar a função de gatilho, estabelecemos o gatilho pelo comando CREATE TRIGGER.
- Uma função de gatilho pode ser utilizada por vários gatilhos.
- O gatilho fica associado à tabela especificada e executa a função especificada nome_da_função quando determinados eventos ocorrerem.



Introdução

- O gatilho pode ser especificado para disparar antes de tentar realizar a operação na linha (antes das restrições serem verificadas e o comando INSERT, UPDATE ou DELETE ser tentado), ou após a operação estar completa (após as restrições serem verificadas e o INSERT, UPDATE ou DELETE ter completado).

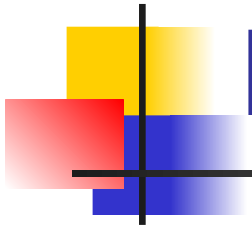


Sintaxe

- CREATE TRIGGER nome { BEFORE | AFTER } { evento [OR ...] } ON tabela [FOR [EACH] { ROW | STATEMENT }] EXECUTE PROCEDURE nome_da_função (argumentos);

Sintaxe - argumentos

- **nome -** nome da trigger
- **Before / after** determina se a função será chamada antes ou depois do evento.
- **evento** indica em que momento a trigger será disparada. Pode ser dispara antes ou depois de um evento de DELETE, UPDATE ou INSERT.
- **tabela** indica em qual tabela a trigger estará associada.
- **Row / statement** especifica se a trigger deve ser disparada uma vez para cada linha afetada pelo evento ou apenas uma vez por comando SQL. Se não for especificado nenhum dos dois, o padrão é FOR EACH STATEMENT.
- **nome_da_função** - função de trigger.



Exemplo 1

```
CREATE TABLE empregados (
    codigo integer NOT NULL PRIMARY KEY,
     nome varchar(50),
     salario numeric (12,2),
     departamento_cod integer,
     ultima_data timestamp,
     ultimo_usuario varchar(30)
)
```

Exemplo 1 (cont.)

```
CREATE FUNCTION empregados_gatilho() RETURNS trigger AS $$
BEGIN
 --verificar se foi fornecido o nome e o salário do empregado
IF NEW.nome IS NULL THEN
     RAISE EXCEPTION 'O nome do empregado não pode ser  nulo';
END IF;
IF NEW.salario IS NULL THEN
    RAISE EXCEPTION '% não pode ter um salário nulo',
    NEW.nome;
END IF;
```

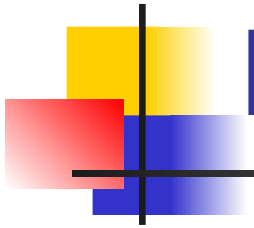
Exemplo 1 (cont.)

```
IF NEW.salario < 0 THEN
    RAISE EXCEPTION '% não pode ter um salário negativo',
    NEW.nome;
END IF;
--Registrar quem alterou a folha de pagamento e quando
NEW.ultima_data := 'now';
NEW.ultimo_usuario := current_user;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Exemplo 1 (cont.)

- CREATE TRIGGER empregados_gatilho BEFORE INSERT OR UPDATE ON empregados FOR EACH ROW EXECUTE PROCEDURE empregados_gatilho();

- INSERT INTO empregados (codigo,nome, salario) VALUES (5,'João',1000);
- INSERT INTO empregados (codigo,nome, salario) VALUES (6,'José',1500);
- INSERT INTO empregados (codigo,nome, salario) VALUES (7,'Maria',2500);
- SELECT * FROM empregados;



Exemplo 1 (cont.)

- INSERT INTO empregados (codigo,nome, salario) VALUES (5,NULL,1000);
- INSERT INTO empregados (codigo,nome, salario) VALUES (5,'Andréia',NULL);

Variável NEW e OLD

- O PostgreSQL disponibiliza duas variáveis importantes para serem usadas em conjunto com as triggers-por-linha: **NEW e OLD.**
- NEW - no caso do INSERT, armazena o registro que está sendo inserido e UPDATE, armazena a nova versão do registro depois da atualização.
- OLD - no caso do DELETE, armazena o registro que está sendo excluído e UPDATE, armazena a antiga versão do registro depois da atualização.

Exemplo 2

- DROP TABLE empregados

```
CREATE TABLE empregados (
    nome varchar NOT NULL,
    salario numeric
);
```

Exemplo 2

```
CREATE TABLE empregados_audit (
    operacao char(1) NOT NULL,
    usuario varchar(30) NOT NULL,
    data timestamp NOT NULL,
    nome varchar(30) NOT NULL,
    salario numeric
);
```

Exemplo 2 (cont.)

- O que esta função faz?
  - Cria uma linha na tabela empregados_audit para refletir a operação realizada na tabela empregados. Utiliza a variável especial TG_OP para descobrir a operação sendo realizada.

- CREATE FUNCTION processa_emp_audit()
  RETURNS TRIGGER AS $$

Exemplo 2 (cont.)

```
BEGIN
IF (TG_OP = 'DELETE') THEN
    INSERT INTO empregados_audit SELECT 'E', user, now(), OLD.*;
    RETURN OLD;
ELSIF (TG_OP = 'UPDATE') THEN
    INSERT INTO empregados_audit SELECT 'A', user, now(), NEW.*;
    RETURN NEW;
ELSIF (TG_OP = 'INSERT') THEN
    INSERT INTO empregados_audit SELECT 'I', user, now(), NEW.*;
    RETURN NEW;
END IF;
RETURN NULL;
END;
$$ language plpgsql;
```

Exemplo 2 (cont.)

- CREATE TRIGGER emp_audit AFTER INSERT OR UPDATE OR DELETE ON empregados FOR EACH ROW EXECUTE PROCEDURE processa_emp_audit();

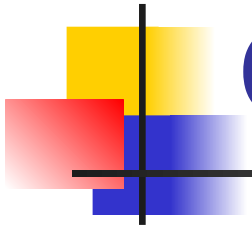
- INSERT INTO empregados (nome, salario) VALUES ('João',1000);
- INSERT INTO empregados (nome, salario) VALUES ('José',1500);
- INSERT INTO empregados (nome, salario) VALUES ('Maria',250);

Exemplo 2 (cont.)

- UPDATE empregados SET salario = 2500 WHERE nome = 'Maria';
- DELETE FROM empregados WHERE nome = 'João';
- SELECT * FROM empregados;
- SELECT * FROM empregados_audit;

Exemplo 3 (Função para validar data nascimento)

```
CREATE OR REPLACE FUNCTION validar_data_nascimento()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.data_nasc > CURRENT_DATE THEN
        RAISE EXCEPTION 'Data de nascimento inválida: %',
    NEW.data_nasc;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



Criando a trigger

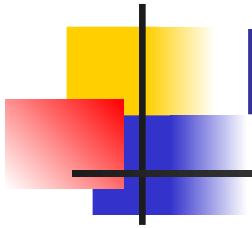
```
CREATE TRIGGER trg_validar_data_nascimento
BEFORE INSERT OR UPDATE ON aluno
FOR EACH ROW
EXECUTE FUNCTION validar_data_nascimento();
```

Exemplo 4

```
CREATE OR REPLACE FUNCTION validar_professor_disciplina()
    RETURNS TRIGGER AS $$
    DECLARE
        existe_professor INT;
    BEGIN
        -- Verifica se o professor existe
        SELECT COUNT(*) INTO existe_professor
        FROM professor
        WHERE matricula = NEW.cod_prof;

        -- Se não existir, lança erro
        IF existe_professor = 0 THEN
            RAISE EXCEPTION 'Código de professor inválido: %', NEW.cod_prof;
        END IF;

        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;
```

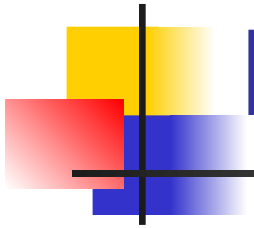


Exemplo 4 (cont.)

```
CREATE TRIGGER trg_validar_professor_disciplina
    BEFORE INSERT OR UPDATE ON disciplina
    FOR EACH ROW
    EXECUTE FUNCTION validar_professor_disciplina();
```

Apagando uma trigger

- DROP TRIGGER nome_trigger ON nome_tabela;



Referências

www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf