CEUB

EDUCAÇÃO SUPERIOR

ceub.br

BANCO DE DADOS II

AULA 15

TRIGGERS

Prof. Leonardo R. de Deus

1. TRIGGER

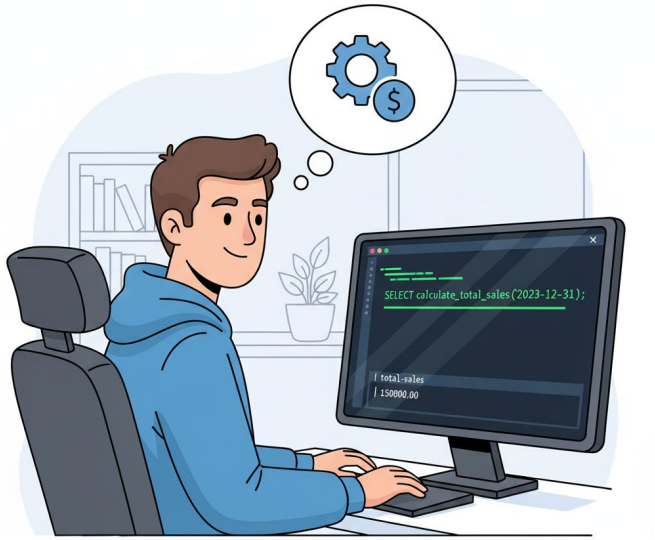
O que é uma TRIGGER?

É um **procedimento automático** executado pelo **próprio banco de dados** em resposta a um evento específico.

1. TRIGGER

O que é uma TRIGGER?

É um **procedimento automático** executado pelo **próprio banco de dados** em resposta a um evento específico.



1. TRIGGER

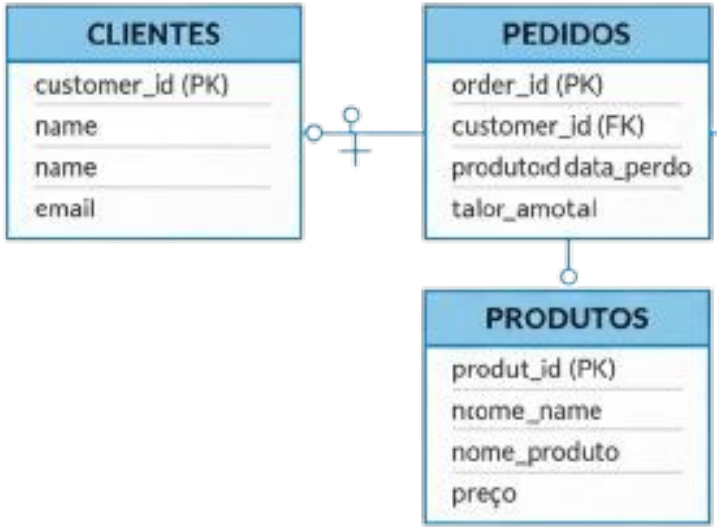
O que é uma TRIGGER?

É um **procedimento automático** executado pelo **próprio banco de dados** em resposta a um evento específico.

COMANDO DML

(Data Manipulation Language)

- **INSERT**
- **UPDATE**
- **DELETE**



Estrutura de uma Trigger no PostgreSQL

Passo 1: Criar uma Função

```
CREATE FUNCTION xxx()
RETURNS TRIGGER
AS $$
BEGIN
  xxxx
END;
$$ LANGUAGE plpgsql;
```

Passo 2: Criar a Trigger

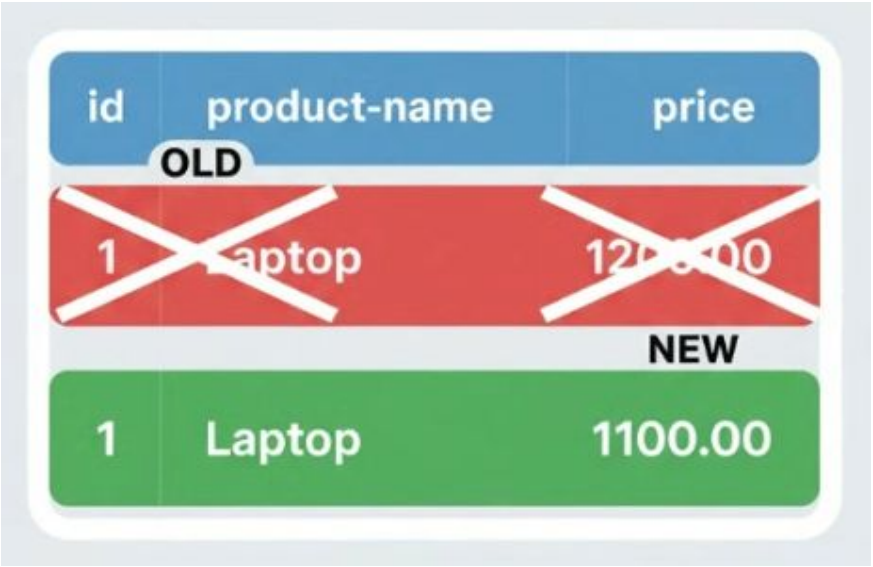
```
CREATE TRIGGER xxx
AFTER UPDATE ON tabela
FOR EACH ROW
EXECUTE FUNCTION xxx();
```

Estrutura de uma Trigger no PostgreSQL

Dentro de uma função de trigger, temos duas "variáveis" especiais que nos permitem ver os dados:

OLD: Contém a linha como ela era antes da alteração **(disponível em UPDATE e DELETE)**.

NEW: Contém a linha como ela será depois da alteração **(disponível em INSERT e UPDATE)**.



Vamos criar uma trigger para registrar as alterações de preço dos produtos.

1. Criar uma tabela que vai armazenar o registro de alterações de preço

```
CREATE TABLE loja.tb_log_precos (
    id_log SERIAL PRIMARY KEY,
    id_produto INT NOT NULL,
    data_alteracao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario_alteracao VARCHAR(100) DEFAULT CURRENT_USER,
    preco_antigo DECIMAL(10, 2),
    preco_novo DECIMAL(10, 2)
);
```

Vamos criar uma trigger para registrar as alterações de preço dos produtos.

2. Criar a função da TRIGGER

```
CREATE OR REPLACE FUNCTION loja.fn_log_mudanca_preco()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.preco <> OLD.preco THEN
        INSERT INTO loja.tb_log_precos (id_produto, preco_antigo, preco_novo)
        VALUES (OLD.id_produto, OLD.preco, NEW.preco);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Vamos criar uma trigger para registrar as alterações de preço dos produtos.

3. Criar a definição da TRIGGER

```
CREATE OR REPLACE TRIGGER trg_log_preco
    AFTER UPDATE
    ON loja.tb02_produto
    FOR EACH ROW
    EXECUTE FUNCTION loja.fn_log_mudanca_preco();
```

3. O MOMENTO DE DISPARO DA TRIGGER

	BEFORE	**AFTER**
Quando usar?	para ações que devem acontecer antes que uma alteração seja permitida no banco	para ações que devem acontecer depois que uma alteração foi concluída com sucesso
Para que usar?	● validar regras de negócio; ● padronizar dados antes que sejam registrados	● criar registros para auditorias ou logs do banco; ● disparar ações em outras tabelas
Observação	● permite cancelar uma operação antes que ela ocorra; ● pode alterar o valor de uma variável antes de inserir no banco	● não permite cancelar a operação que está associada; ● não permite alterar os dados, pois já foram inseridos no banco

4. CRIANDO UMA TRIGGER DE VALIDAÇÃO

Foi definido como regra para o banco que:

- **o nome do cliente deve ser armazenado sempre em letra maiúscula;**
- **o endereço de email não pode ser inválido.**

1. Criar um função trigger para fazer as validações

```
CREATE OR REPLACE FUNCTION loja.fn_validar_cliente()
RETURNS TRIGGER AS $$
BEGIN
    NEW.nome_cliente := UPPER(NEW.nome_cliente);
    IF NEW.email NOT LIKE '%@%' THEN
        RAISE EXCEPTION 'E-mail inválido. O e-mail deve conter um "@".';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

4. CRIANDO UMA TRIGGER DE VALIDAÇÃO

Foi definido como regra para o banco que:

- **o nome do cliente deve ser armazenado sempre em letra maiúscula;**
- **o endereço de email não pode ser inválido.**

2. Criar a definição da trigger

```
CREATE OR REPLACE TRIGGER trg_validar_cliente
    BEFORE INSERT OR UPDATE
    ON loja.tb01_cliente
    FOR EACH ROW
    EXECUTE FUNCTION loja.fn_validar_cliente();
```

5. CRIANDO UMA TRIGGER PARA SINCRONIZAR DADOS

Sempre que um cliente se cadastrar na loja, se o email já existir na tabela de leads, devemos alterar o status desse “lead” para convertido

1. **Criação da função da trigger**

```
CREATE OR REPLACE FUNCTION loja.fn_marcar_lead_convertido()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE loja.tb06_leads_newsletter
    SET convertido = TRUE
    WHERE email = NEW.email; -- Usamos o e-mail do NOVO cliente
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Sempre que um cliente se cadastrar na loja, se o email já existir na tabela de leads, devemos alterar o status desse “lead” para convertido

2. Criar a definição da trigger

```
CREATE OR REPLACE TRIGGER trg_converter_lead
    AFTER INSERT -- Dispare APÓS um novo cliente ser inserido
    ON loja.tb01_cliente
    FOR EACH ROW
    EXECUTE FUNCTION loja.fn_marcar_lead_convertido();
```

6. RISCOS AO USAR TRIGGERS

1. **Risco de Performance**
   - Um trigger `FOR EACH ROW` (para cada linha) é executado *uma vez para cada linha*.
   - Se você executar um `UPDATE` que afeta **1 milhão de linhas**, o seu trigger de auditoria será disparado **1 milhão de vezes**.

6. RISCOS AO USAR TRIGGERS

1. **Risco de Performance**
   - Um trigger `FOR EACH ROW` (para cada linha) é executado *uma vez para cada linha*.
   - Se você executar um `UPDATE` que afeta **1 milhão de linhas**, o seu trigger de auditoria será disparado **1 milhão de vezes**.

2. **Risco de Loops Infinitos**
   - Você tem um trigger na `Tabela A` que, ao ser atualizada, atualiza a `Tabela B`.
   - Você tem *outro* trigger na `Tabela B` que, ao ser atualizada, atualiza a `Tabela A`.

   Resultado: Você atualiza a `Tabela A` -> Dispara Trigger 1 -> Atualiza `Tabela B` -> Dispara Trigger 2 -> Atualiza `Tabela A` -> Dispara Trigger 1 -> Atualiza `Tabela B`...

6. RISCOS AO USAR TRIGGERS

1. **Risco de Performance**
   - Um trigger `FOR EACH ROW` (para cada linha) é executado *uma vez para cada linha*.
   - Se você executar um `UPDATE` que afeta **1 milhão de linhas**, o seu trigger de auditoria será disparado **1 milhão de vezes**.

2. **Risco de Loops Infinitos**
   - Você tem um trigger na `Tabela A` que, ao ser atualizada, atualiza a `Tabela B`.
   - Você tem *outro* trigger na `Tabela B` que, ao ser atualizada, atualiza a `Tabela A`.

   Resultado: Você atualiza a `Tabela A` -> Dispara Trigger 1 -> Atualiza `Tabela B` -> Dispara Trigger 2 -> Atualiza `Tabela A` -> Dispara Trigger 1 -> Atualiza `Tabela B`…

3. **Risco de Manutenção**

   Triggers escondem a lógica de negócio

6. RISCOS AO USAR TRIGGERS

1. **Risco de Performance**
   - Um trigger `FOR EACH ROW` (para cada linha) é executado *uma vez para cada linha*.
   - Se você executar um `UPDATE` que afeta **1 milhão de linhas**, o seu trigger de auditoria será disparado **1 milhão de vezes**.

2. **Risco de Loops Infinitos**
   - Você tem um trigger na `Tabela A` que, ao ser atualizada, atualiza a `Tabela B`.
   - Você tem *outro* trigger na `Tabela B` que, ao ser atualizada, atualiza a `Tabela A`.

   Resultado: Você atualiza a `Tabela A` -> Dispara Trigger 1 -> Atualiza `Tabela B` -> Dispara Trigger 2 -> Atualiza `Tabela A` -> Dispara Trigger 1 -> Atualiza `Tabela B`...

3. **Risco de Manutenção**

   Triggers escondem a lógica

DOCUMENTAÇÃO É A SALVAÇÃO!

OBRIGADO
A TODOS!

CEUB

EDUCAÇÃO SUPERIOR