**BD2 – Prova - Resumo**

**Triggers**

Triggers são ações automáticas no banco de dados que são disparadas quando certos eventos ocorrem. Elas ajudam a automatizar processos e garantir a integridade dos dados.

Requisitos de um Trigger:

1. Evento: Define o tipo de evento que dispara o trigger. Exemplos:

* ON DELETE: Disparado quando uma linha é deletada.
* ON UPDATE: Disparado quando uma linha é atualizada.
* ON INSERT: Disparado quando uma linha é inserida

1. Condição (opcional): Define uma condição que precisa ser atendida para o trigger ser executado. Exemplo: "Só aciona o trigger se o valor da coluna 'status' for 'ativo'."
2. Ação: Define o que será feito quando o trigger for disparado. Exemplos de ações:

* Deletar registros.
* Atualizar dados.
* Inserir novos dados.
* Chamar uma função.

**Modelo ECA (Evento-Condicao-Ação)**: Esse modelo descreve o que deve ocorrer quando um evento é disparado, podendo incluir uma condição e a ação a ser executada.

| **Elemento** | **Significado** | **Exemplo** |
| --- | --- | --- |
| **Evento** | O que aconteceu na tabela? | Um cliente foi deletado (ON DELETE) |
| **Condição** (opcional) | Deve acontecer **somente se** algo for verdadeiro? | Só se cliente.status = 'ativo' |
| **Ação** | O que deve ser feito? | Deletar todos os pedidos desse cliente |

Exemplo: Tabela clientes e Tabela pedidos.

Suponha que, quando um cliente é excluído, todos os pedidos desse cliente também devem ser excluídos automaticamente. Para isso, podemos configurar um trigger.

CREATE TRIGGER exclui\_pedidos\_cliente

AFTER DELETE ON clientes

FOR EACH ROW

BEGIN

DELETE FROM pedidos WHERE cliente\_id = OLD.cliente\_id;

END;

Explicação:

* Evento: AFTER DELETE ON clientes – Quando um cliente é excluído.
* Ação: Deletar todos os pedidos do cliente excluído.

--------------------------------------------------------------------------------- chatgpt

Triggers (ou "gatilhos") são objetos de banco de dados que executam automaticamente uma ação quando certos eventos ocorrem em uma tabela ou view. Esses eventos podem ser:

Você define:

Quando a trigger deve agir (BEFORE ou AFTER a ação),

Qual evento deve acioná-la (INSERT, UPDATE, DELETE),

* INSERT (quando um novo registro é inserido),
* UPDATE (quando um registro existente é alterado),
* DELETE (quando um registro é removido).

O que fazer quando isso acontece (função executada).

**Para que servem as triggers?**

* As triggers são usadas para:
* Manter integridade dos dados, além das constraints normais.
* Auditar mudanças (ex: registrar quem alterou uma tabela e quando).
* Prevenir ações não permitidas (ex: impedir exclusão de registros importantes).
* Atualizar automaticamente dados em outras tabelas relacionadas.
* Aplicar regras de negócio no nível do banco.

**Como funciona?**

Imagine uma trigger como um “vigia” de uma tabela. Por exemplo:

“Toda vez que alguém inserir um funcionário na tabela employees, automaticamente registre a data e hora da inserção em uma tabela de log.”

-----------------------------------------------------------------------------------------------------------------------------------------------------------

**BEFORE x AFTER**

Esses termos definem o **momento em que a trigger será executada**, em relação à ação que ocorreu no banco de dados (como um INSERT, UPDATE ou DELETE).

| **Tipo** | **Quando executa?** | **Uso típico** |
| --- | --- | --- |
| BEFORE (antes) | Executa antes da ação (INSERT/UPDATE/DELETE) | Permite **validar**, **corrigir** ou até **bloquear** dados **antes** que eles sejam inseridos, atualizados ou deletados. |
| AFTER (depois) | Executa depois da ação (INSERT/UPDATE/DELETE) | Criar logs, atualizar outras tabelas, notificações.  Muito usado para registrar, notificar, ou executar ações relacionadas. |

**NEW e OLD**

Esses são **pseudo-registros** disponíveis dentro da trigger, usados **dentro da função da trigger** para acessar os valores da linha

| **Nome** | **Descrição** | **Disponível em** |
| --- | --- | --- |
| NEW | Representa os **dados novos** (após a ação) | INSERT, UPDATE |
| OLD | Representa os **dados antigos** (antes da ação) | UPDATE, DELETE |

Exemplo: “se o salário de um funcionário foi alterado, você pode usar:”

OLD.salary → valor antigo

NEW.salary → valor novo

-----------------------------------------------------------------------------------------------------------------------------------------------------------

**FOR EACH ROW x FOR EACH STATEMENT**

A diferença não é se modifica ou não. É: quantas vezes a trigger roda e com que dados ela trabalha.

| **Tipo** | **Descrição** |
| --- | --- |
| FOR EACH ROW | A trigger é executada **uma vez para cada linha afetada**  A trigger roda para cada linha afetada.  Permite usar NEW e OLD.  Mais comum. |
| FOR EACH STATEMENT | Executa **uma vez por operação**, mesmo que várias linhas sejam afetadas  Não tem acesso direto ao conteúdo linha a linha.  Serve para ações gerais e para logs gerais |

Geralmente você vai usar **FOR EACH ROW quando quiser acessar NEW e OLD**.

Exemplo:

UPDATE employees SET dept\_no = 'd001' WHERE gender = 'F'

* Com FOR EACH ROW: a trigger roda uma vez para cada funcionária.
* Com FOR EACH STATEMENT: roda só uma vez, após o comando inteiro.

**Tipos de eventos -> criar triggers para os seguintes eventos**

| **Evento** | **Significado** | **Quando usar** |
| --- | --- | --- |
| INSERT | Quando um novo registro/linha é adicionado | Para validar dados ou criar log |
| UPDATE | Quando um registro/linha existente é alterado | Para registrar alterações ou aplicar regras |
| DELETE | Quando um registro/linha é excluído | Para impedir ou registrar exclusões |

Você também pode combinar:

AFTER INSERT OR UPDATE OR DELETE

-----------------------------------------------------------------------------------------------------------------------------------------------------------

**Funções (FUNCTION)**

A trigger sempre chama uma função que contém o que será executado.

* A função retorna NEW (para INSERT/UPDATE) ou OLD (para DELETE).

**RETURN NEW x RETURN OLD**

| **Contexto** | **O que retornar** |
| --- | --- |
| BEFORE INSERT / UPDATE | RETURN NEW; para confirmar a nova linha |
| BEFORE DELETE | RETURN OLD; para confirmar a remoção |
| AFTER | O retorno é ignorado (mas você ainda precisa colocar RETURN NEW; ou RETURN OLD;) |

**Verificações dentro de Triggers**

Usamos estruturas como IF, CASE, ou até cláusulas WHEN na trigger.

Usando IF dentro da função da trigger

É a forma mais direta. Você testa uma condição lógica e executa algo se for verdadeira.

Exemplo: Bloquear se salário < 1000

IF NEW.salario < 1000 THEN

RAISE EXCEPTION 'Salário inválido!';

END IF;

| **Expressão** | **Explicação** |
| --- | --- |
| NEW.salario < 1000 | Se o novo salário for menor que 1000 |
| RAISE EXCEPTION | Dispara erro e impede o comando |

Usando IF...ELSE para ações diferentes

IF NEW.status = 'ativo' THEN

-- faz algo

ELSE

-- faz outra coisa

END IF;

Usando cláusula WHEN na trigger

Você pode limitar o acionamento da trigger com uma condição diretamente na criação:

CREATE TRIGGER trg\_log\_salario

AFTER UPDATE ON funcionarios

FOR EACH ROW

WHEN (OLD.salario <> NEW.salario)

EXECUTE FUNCTION log\_salario();

Essa trigger só roda se o salário mudou. A condição OLD.salario <> NEW.salario deve ser verdadeira.

**Operadores lógicos que retornam verdadeiro/falso**

| **Operador** | **Uso** | **Exemplo** |
| --- | --- | --- |
| = | Igual a | NEW.cargo = 'gerente' |
| <> | Diferente de | OLD.nome <> NEW.nome |
| <, >, <=, >= | Comparações | NEW.salario > 5000 |
| AND | E | idade > 18 AND status = 'ativo' |
| OR | Ou | cargo = 'analista' OR cargo = 'gerente' |
| NOT | Negação | NOT (ativo) |
| IS NULL, IS NOT NULL | Nulo/não nulo | email IS NOT NULL |

**INSTEAD OF em Triggers**

Tipo especial de trigger que substitui a ação original (INSERT, UPDATE, DELETE) por outra definida na trigger.

Ou seja, em vez de o banco executar a operação normalmente, ele executa o que você mandar dentro da trigger.

Quando usamos? Principalmente em **Views (visões)**

* Views geralmente não permitem UPDATE, INSERT ou DELETE diretamente, pois não são tabelas reais.
* Usamos INSTEAD OF para permitir essas ações em uma view, redirecionando a operação para as tabelas reais.

Exemplo Prático

CREATE VIEW funcionarios\_view AS

SELECT emp\_no, first\_name, last\_name

FROM funcionarios;

Você não pode fazer UPDATE diretamente nela. Mas com uma trigger INSTEAD OF, pode!

Criando a função:

CREATE OR REPLACE FUNCTION atualiza\_funcionario\_view()

RETURNS TRIGGER AS $$

BEGIN

UPDATE funcionarios

SET first\_name = NEW.first\_name,

last\_name = NEW.last\_name

WHERE emp\_no = NEW.emp\_no;

RETURN NEW;

END;

Criando a Trigger:

CREATE TRIGGER trg\_update\_func\_view

INSTEAD OF UPDATE ON funcionarios\_view

FOR EACH ROW

EXECUTE FUNCTION atualiza\_funcionario\_view();

O que acontece aqui?

| **Componente** | **O que faz** |
| --- | --- |
| INSTEAD OF UPDATE | Intercepta o UPDATE na view |
| funcionarios\_view | A trigger age na view, não na tabela original |
| UPDATE funcionarios | A ação real ocorre na tabela base funcionarios |

Resumo:

| **Tipo** | **Onde se aplica?** | **Para quê serve?** |
| --- | --- | --- |
| BEFORE, AFTER | Tabelas reais | Validar, registrar, modificar |
| INSTEAD OF | **Views** | Substituir a operação por outra, permitindo modificações |

***OBSERVAÇÕES***

**O que é um log?**

No contexto de banco de dados (e sistemas em geral), um log é um registro de eventos que ocorreram — como uma "caixa preta" que guarda o histórico de ações.

* Um log é uma tabela (ou arquivo) onde são gravadas informações sobre operações realizadas, como:
* Inserções, atualizações ou deleções de dados.
* Quem fez a ação.
* Quando a ação foi feita.
* O que foi alterado.

Exemplo prático:

Imagine que você tem uma tabela employees com dados de funcionários. Se alguém atualiza o salário de um funcionário, você pode registrar essa alteração em uma tabela de log, por exemplo:

| emp\_no | ação | campo | valor\_antigo | valor\_novo | data\_hora |
| --- | --- | --- | --- | --- | --- |
| 1001 | UPDATE | salary | 4000 | 4500 | 2025-05-14 10:45:12 |

Isso permite que você saiba o que aconteceu, quando e por quem, mesmo que os dados originais tenham mudado.

**Estrutura TRIGGER**

CREATE TRIGGER meu\_trigger -- cria o trigegr

BEFORE INSERT ON minha\_tabela -- antes de realizar o insert na tabela

FOR EACH ROW -- faça isso em cada linha que realiza o insert

BEGIN -- comece

-- lógica aqui (use NEW.coluna, OLD.coluna) -- faça tal coisa (qualquer coisa kkkk)

END; -- finalize

**Exemplo: Bloquear salários menores que 1000 na tabela funcionários**

Quando alguém tenta inserir ou atualizar um funcionário com salário menor que 1000, a operação deve ser bloqueada com erro.

Estrutura da Tabela:

CREATE TABLE funcionarios (

id INT PRIMARY KEY AUTO\_INCREMENT,

nome VARCHAR(100),

salario DECIMAL(10,2)

);

Trigger BEFORE INSERT:

DELIMITER //

CREATE TRIGGER trg\_valida\_salario\_insert

BEFORE INSERT ON funcionarios

FOR EACH ROW

BEGIN

IF NEW.salario < 1000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Salário não pode ser menor que 1000.';

END IF;

END //

DELIMITER ;

Trigger BEFORE UPDATE:

DELIMITER //

CREATE TRIGGER trg\_valida\_salario\_update

BEFORE UPDATE ON funcionarios

FOR EACH ROW

BEGIN

IF NEW.salario < 1000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Salário atualizado não pode ser menor que 1000.';

END IF;

END //

DELIMITER ;

Explicação:

| **Comando** | **Significado** |
| --- | --- |
| DELIMITER $$ | Muda o delimitador temporariamente para permitir múltiplas linhas na trigger |
| FOR EACH ROW | Roda a trigger para **cada linha afetada** |
| NEW.salario | Valor que está sendo inserido/atualizado |
| SIGNAL SQLSTATE | Gera um erro personalizado |