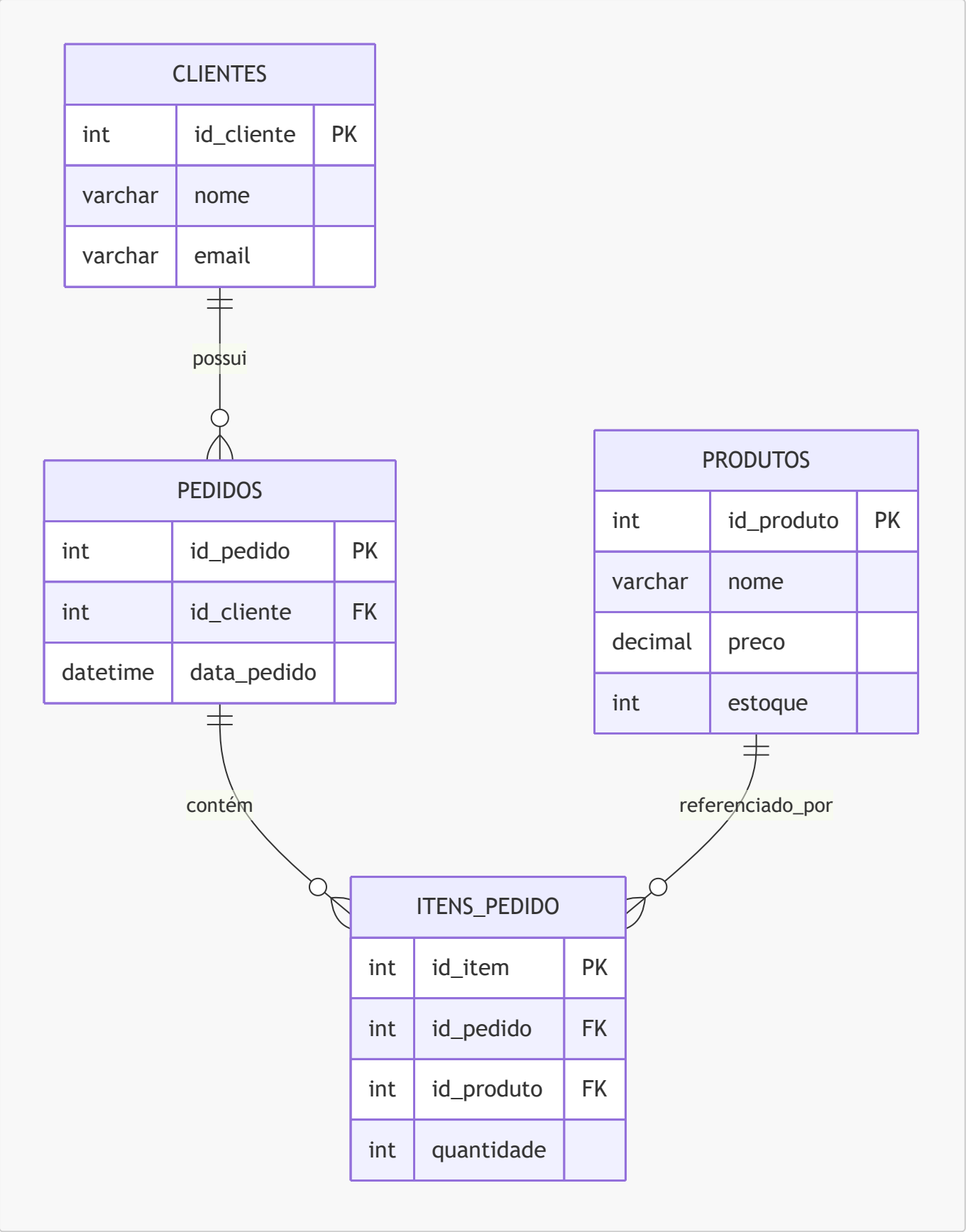
Aula Completa: Gatilhos (Triggers), Funções e Views em MySQL

Modelo ER do Banco Usado para os Exemplos



Conceito de Gatilhos

Um gatilho (trigger) é um bloco de código SQL que é executado automaticamente em resposta a um evento INSERT, UPDATE ou DELETE em uma tabela. São úteis para garantir integridade de dados, aplicar regras de negócio e automatizar operações internas no banco.

Tipos de Gatilhos

Tipo	Quando ocorre
BEFORE INSERT	Antes de inserir uma nova linha
AFTER INSERT	Após a inserção de uma linha
BEFORE UPDATE	Antes de atualizar uma linha
AFTER UPDATE	Após a atualização de uma linha
BEFORE DELETE	Antes de excluir uma linha
AFTER DELETE	Após a exclusão de uma linha

Estrutura de um Gatilho

```
DELIMITER $$
CREATE TRIGGER nome_gatilho
[BEFORE | AFTER] [INSERT | UPDATE | DELETE]
ON nome_tabela
FOR EACH ROW
BEGIN
    -- corpo do gatilho
END $$
DELIMITER ;
```

Estrutura de uma Função

```
DELIMITER $$
CREATE FUNCTION nome_funcao(parâmetros)
RETURNS tipo_retorno
[DETERMINISTIC | NOT DETERMINISTIC]
BEGIN
    -- corpo da função
    RETURN valor;
END $$
DELIMITER ;
```

Estrutura de uma View

```
CREATE VIEW AS [STATEMENT]
```

Criação do Banco e Tabelas

```
CREATE DATABASE IF NOT EXISTS Loja;
USE Loja;

CREATE TABLE CLIENTES (
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE PRODUTOS (
    id_produto INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10,2),
    estoque INT
);

CREATE TABLE PEDIDOS (
    id_pedido INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    data_pedido DATETIME,
    total_pedido DECIMAL(10,2) DEFAULT 0,
    FOREIGN KEY (id_cliente) REFERENCES CLIENTES(id_cliente)
);

CREATE TABLE ITENS_PEDIDO (
    id_item INT AUTO_INCREMENT PRIMARY KEY,
    id_pedido INT,
    id_produto INT,
    quantidade INT,
    FOREIGN KEY (id_pedido) REFERENCES PEDIDOS(id_pedido),
    FOREIGN KEY (id_produto) REFERENCES PRODUTOS(id_produto)
);
```

Povoamento do Banco de Dados

```
INSERT INTO CLIENTES (nome, email) VALUES
('João Silva', 'joao@exemplo.com'),
('Maria Souza', 'maria@exemplo.com'),
('Manoela Medeiros', 'manoela@exemplo.com');

INSERT INTO PRODUTOS (nome, preco, estoque) VALUES
('Notebook', 3500.00, 10),
('Teclado', 200.00, 50),
```

```
('Mouse', 100.00, 100),
('Fone de Ouvido', 50.00, 120);
```

Views com Exemplo de Uso

View 1: Clientes que realizaram pedidos

View para exibir todos os clientes que possuem pedidos.

```
CREATE VIEW cliente_pedido AS
SELECT
    c.id_cliente,
    c.nome,
    date_format(p.data_pedido, '%h/%m/%Y') AS 'Data Pedido'
FROM clientes c
JOIN pedidos p ON p.id_cliente = c.id_cliente;

-- Exemplo de uso
SELECT * FROM cliente_pedido;
```

View 2: Produtos com estoque em limite crítico

View para exibir produtos com estoque críticos, ou seja, quantidade igual ou inferior a 10 unidades.

```
CREATE VIEW lista_estoque_critico AS
SELECT * FROM produtos p WHERE p.estoque <= 10;

-- Exemplo de uso
SELECT * FROM lista_estoque_critico;
```

Funções com Exemplo de Uso

Função 1: calcular_total_item

Calcula o valor total de um item do pedido com base no preço e quantidade.

```
DELIMITER $$
CREATE FUNCTION calcular_total_item(preco DECIMAL(10,2), quantidade INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN preco * quantidade;
END $$
DELIMITER ;

-- Exemplo de uso:
SELECT calcular_total_item(150.00, 2); -- Resultado: 300.00
```

Função 2: calcular_total_pedido

Soma todos os itens de um pedido e retorna o total.

```
DELIMITER $$
CREATE FUNCTION calcular_total_pedido(pedido_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(pr.preco * ip.quantidade)
    INTO total
    FROM ITENS_PEDIDO ip
    JOIN PRODUTOS pr ON pr.id_produto = ip.id_produto
    WHERE ip.id_pedido = pedido_id;
    RETURN IFNULL(total, 0);
END $$
DELIMITER ;
```

Função 3: realizar_pedido

Realiza um pedido automaticamente criando o registro em PEDIDOS e ITENS_PEDIDO. Valida o estoque.

```
DELIMITER $$
CREATE FUNCTION realizar_pedido(
    pid_cliente INT,
    pid_produto INT,
    qtd INT
)
RETURNS VARCHAR(100)
MODIFIES SQL DATA
DETERMINISTIC
BEGIN
    DECLARE estoque_atual INT;
    DECLARE novo_pedido INT;

    SELECT estoque INTO estoque_atual FROM PRODUTOS WHERE id_produto =
pid_produto;

    IF estoque_atual < qtd THEN
        RETURN 'Estoque insuficiente';
    END IF;

    INSERT INTO PEDIDOS (id_cliente, data_pedido)
    VALUES (pid_cliente, NOW());

    SET novo_pedido = LAST_INSERT_ID();

    INSERT INTO ITENS_PEDIDO (id_pedido, id_produto, quantidade)
```

```
    VALUES (novo_pedido, pid_produto, qtd);

    RETURN 'Pedido realizado com sucesso';
END $$
DELIMITER ;

-- Exemplo de uso:
SELECT realizar_pedido(1, 2, 3);
```

> **LAST_INSERT_ID()** é uma função do MySQL que retorna o valor do último **AUTO_INCREMENT** inserido na sessão atual. Isso é útil quando você insere uma nova linha em uma tabela com chave primária auto incremental (como id_pedido) e precisa usar esse valor imediatamente em outra operação.

Gatilhos com Exemplo de Uso

AFTER INSERT - Atualizar estoque após venda

Reduz automaticamente o estoque do produto após um item ser adicionado ao pedido.

```
DELIMITER $$
CREATE TRIGGER tg_atualizar_estoque
AFTER INSERT ON ITENS_PEDIDO
FOR EACH ROW
BEGIN
    UPDATE PRODUTOS
    SET estoque = estoque - NEW.quantidade
    WHERE id_produto = NEW.id_produto;
END $$
DELIMITER ;
```

AFTER INSERT - Atualizar total do pedido

Recalcula o valor total do pedido após a inserção de um item.

```
DELIMITER $$
CREATE TRIGGER tg_atualizar_total_pedido
AFTER INSERT ON ITENS_PEDIDO
FOR EACH ROW
BEGIN
    UPDATE PEDIDOS
    SET total_pedido = calcular_total_pedido(NEW.id_pedido)
    WHERE id_pedido = NEW.id_pedido;
END $$
DELIMITER ;
```

BEFORE INSERT - Validar estoque

Impede a inserção de itens caso a quantidade ultrapasse o estoque.

```
DELIMITER $$
CREATE TRIGGER tg_validar_estoque
BEFORE INSERT ON ITENS_PEDIDO
FOR EACH ROW
BEGIN
    DECLARE estoque_atual INT;
    SELECT estoque INTO estoque_atual FROM PRODUTOS WHERE id_produto =
NEW.id_produto;
    IF estoque_atual < NEW.quantidade THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Estoque insuficiente para o produto';
    END IF;
END $$
DELIMITER ;
```

AFTER DELETE - Repor estoque ao remover item

Aumenta o estoque ao excluir um item de um pedido e atualiza o total.

```
DELIMITER $$
CREATE TRIGGER tg_repor_estoque
AFTER DELETE ON ITENS_PEDIDO
FOR EACH ROW
BEGIN
    UPDATE PRODUTOS
    SET estoque = estoque + OLD.quantidade
    WHERE id_produto = OLD.id_produto;

    UPDATE PEDIDOS
    SET total_pedido = calcular_total_pedido(OLD.id_pedido)
    WHERE id_pedido = OLD.id_pedido;
END $$
DELIMITER ;
```

AFTER UPDATE - Recalcular total do pedido

Atualiza o total do pedido quando um item é modificado.

```
DELIMITER $$
CREATE TRIGGER tg_recalcular_total_update
AFTER UPDATE ON ITENS_PEDIDO
FOR EACH ROW
BEGIN
    UPDATE PEDIDOS
    SET total_pedido = calcular_total_pedido(NEW.id_pedido)
    WHERE id_pedido = NEW.id_pedido;
```

```
END $$
DELIMITER ;
```

Testando os Recursos

```
-- Criar um pedido diretamente pela função:
SELECT realizar_pedido(1, 1, 2);

-- Conferir estoque e total do pedido:
SELECT * FROM PRODUTOS;
SELECT * FROM PEDIDOS;
SELECT * FROM ITENS_PEDIDO;

-- Excluir um item:
DELETE FROM ITENS_PEDIDO WHERE id_item = 1;

-- Verificar estoque e total novamente:
SELECT * FROM PRODUTOS;
SELECT * FROM PEDIDOS;
```

Exercícios

1. Crie uma função que retorna o número de pedidos de um cliente.
2. Crie um gatilho que impeça o cadastro de e-mails repetidos na tabela CLIENTES.
3. Adicione um campo `ultima_atualizacao` em PEDIDOS e crie um gatilho AFTER UPDATE para atualizá-lo.