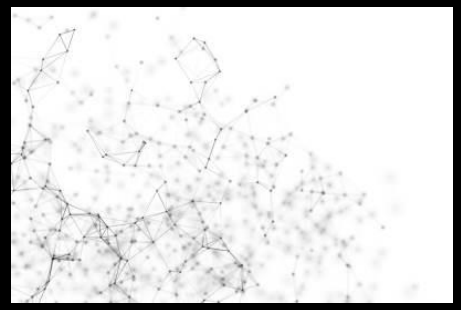
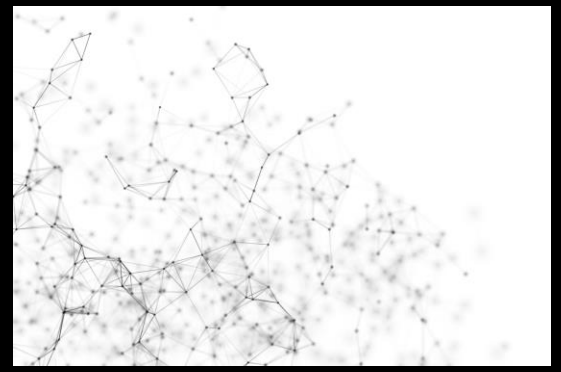




UC6- AULA 6

CRIAR E MANTER BANCO DE DADOS

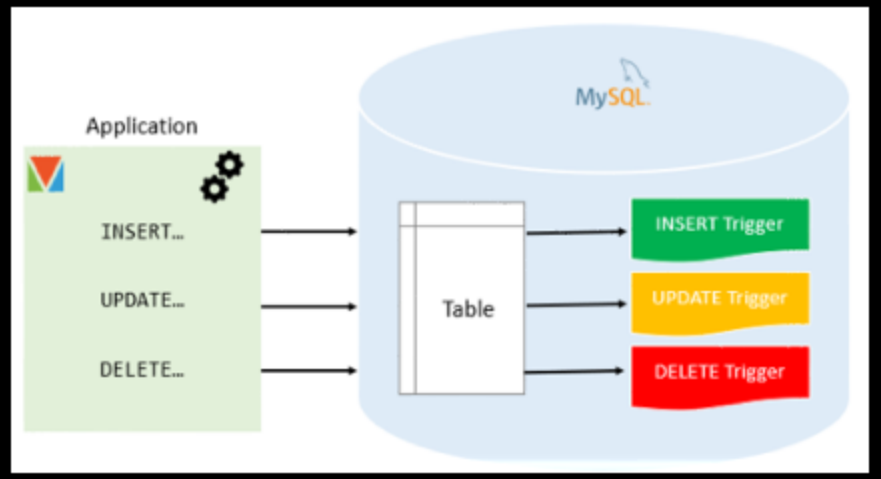
PROF. CALEBE PEREIRA LEMOS

EMAIL: CALEBE.PEREIRA@UFMS.BR

TRIGGER (GATILHO)

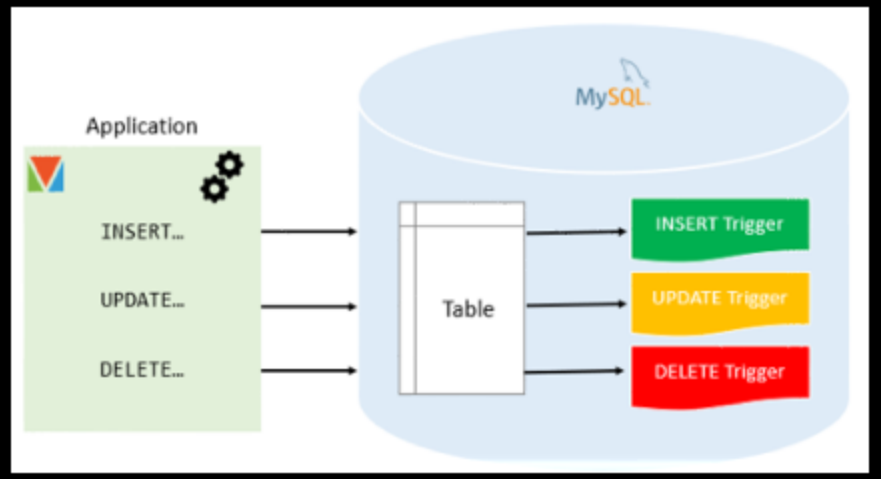
Triggers

É muito comum, em aplicações que utilizam bancos de dados, que ações sejam disparadas em resposta ou como consequência de outras, realizando operações de cálculo, validações e, em geral, surtindo alterações na base de dados.

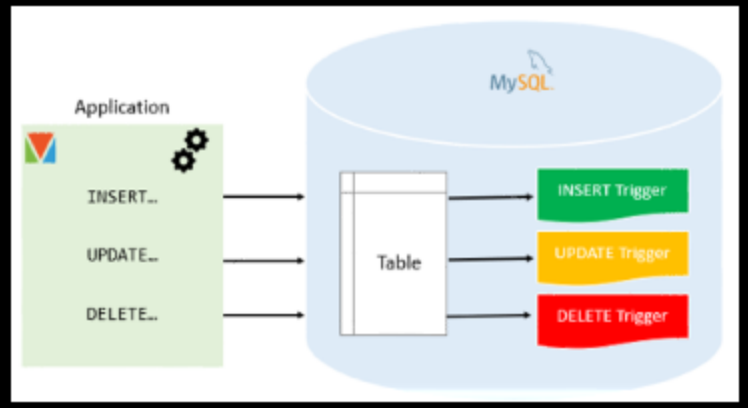


Triggers

Em muitos casos, os programadores optam por executarem tais ações a partir da própria aplicação, executando várias instruções SQL em sequência para obter o resultado esperado.

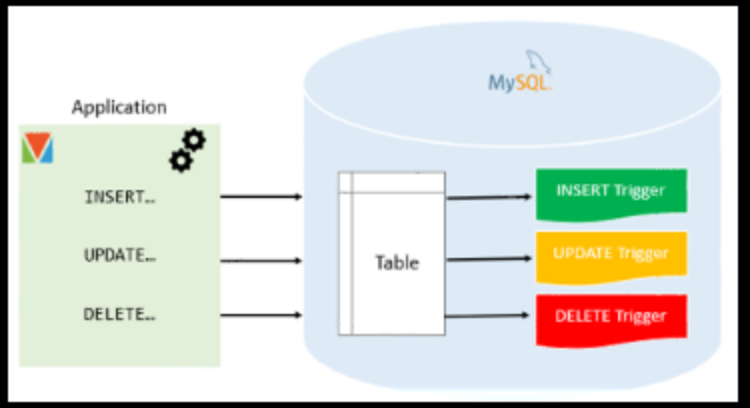


Triggers



Em bancos de dados, um trigger é um tipo de procedimento armazenado que é automaticamente executado (ou "disparado") quando um evento específico ocorre em um banco de dados. Esses eventos podem ser operações de INSERT, UPDATE, DELETE em uma tabela ou até mesmo um evento de login ou logout de um usuário.

Triggers



Triggers podem ser executadas ANTES ou DEPOIS das operações de INSERT, UPDATE e DELETE de registros.

- Trigger "Before" (Antes): Executa a ação definida antes da operação que disparou o trigger (antes da inserção, atualização ou exclusão dos dados na tabela).
- Trigger "After" (Depois): Executa a ação após a operação ter sido realizada com sucesso (após a inserção, atualização ou exclusão dos dados na tabela).

Triggers – Funcionalidades Principais

- **Automatização de ações:** Triggers automatizam a execução de comandos ou ações no banco de dados quando determinados eventos ocorrem.
- **Aplicação de regras de negócio:** Permitem impor regras específicas de negócio no nível do banco de dados, garantindo consistência e integridade dos dados.
- **Registro de alterações:** São úteis para o registro de modificações em tabelas, podendo ser utilizadas para auditoria ou histórico de alterações.

Exemplo

Suponha que temos duas tabelas: uma para armazenar informações sobre os funcionários (funcionarios) e outra para registrar as informações de salário (registro_salario). Queremos criar um trigger que, após a inserção de um novo salário na tabela registro_salario, atualize automaticamente o total de salários pagos para um determinado funcionário na tabela funcionarios.

Sintaxe:

```
CREATE TRIGGER nome_da_trigger  -- NOME ATRIBUIDO
BEFORE/AFTER AÇÃO_DESEJADA (INSERT/UPDATE/DELETE)  --MOMENTO E AÇÃO
ON tabela
FOR EACH ROW – INDICA QUE SERÁ EXECUTADA PARA CADA LINHA AFETADA
BEGIN – DELIMITA O BLOCO DE COMANDOS QUE SERÃO EXECUTADOS
        -- Corpo da trigger com as ações a serem executadas
END;
```

Observação

Não é possível criar mais de uma trigger para o mesmo evento e momento de execução na mesma tabela.

Por exemplo: Não podemos criar dois gatilhos ***AFTER INSERT*** na mesma tabela.

NEW and OLD

Como triggers são executadas em conjunto com operações de inclusão ou exclusão, é necessário ter acesso aos registros que estão sendo incluídos ou removidos, para isso, utiliza-se as palavras reservadas **New** e **Old**.

NEW and OLD

New: Possibilita o acesso aos valores consequentes do comando que disparou o gatilho (novo registro);

Old: Possibilita o acesso aos valores como estavam antes da execução do comando que disparou o gatilho (registro antigo);

COMANDO	OLD	NEW
INSERT	Não se aplica	Valores do registro inserido
UPDATE	Valores do registro antes da alteração	Valores do registro após da alteração
DELETE	Valores do registro excluído	Não se aplica

Exemplo

Situação: Automatizar o cálculo de desconto de um produto. Vamos criar um trigger que, no momento da inserção, capture o valor do produto a ser inserido e aplique um desconto de 10% a esse valor.

Tabelas e inserções:

```
CREATE TABLE produto (
        idProduto INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        nome VARCHAR(50) NOT NULL,
        preco_normal DECIMAL(10,2) NOT NULL,
         preco_desconto DECIMAL(10,2) NOT NULL
);

INSERT INTO produto(nome, preco_normal)
VALUES ('Monitor’, 600.00);
```

Trigger:

```
CREATE TRIGGER tr_desconto BEFORE INSERT
ON  produto
FOR EACH ROW
SET NEW.preco_desconto = (NEW.preco_normal * 0.90);
```

DELIMITER - Delimitador

No MySQL, o delimitador padrão para comandos SQL é o ponto e vírgula (;). O delimitador é usado para separar diferentes instruções SQL em uma única consulta ou script. No entanto, ao criar rotinas, como triggers, funções ou procedimentos armazenados, esses blocos de código podem conter várias instruções SQL e, portanto, vários pontos e vírgulas. Isso pode confundir o MySQL ao tentar determinar onde termina o bloco de código da rotina.

DELIMITER - Delimitador

Para resolver isso, o comando **DELIMITER** é usado para temporariamente alterar o delimitador padrão de comandos do MySQL de ; para outro caractere. Geralmente, // ou $ são usados como delimitadores alternativos.

DELIMITER – Sintaxe:

Para resolver isso, o comando **DELIMITER** é usado para temporariamente alterar o delimitador padrão de comandos do MySQL de ; para outro caractere. Geralmente, // ou $ são usados como delimitadores alternativos.

DELIMITER novo_delimitador (exemplo: $)
-- Seu código do procedimento, trigger ou função aqui
DELIMITER ;

Exemplo Trigger e Delimitador:

```
DELIMITER $
CREATE TRIGGER tr_desconto BEFORE INSERT
ON  produto
FOR EACH ROW
BEGIN
        SET NEW.preco_desconto = (NEW.preco_normal * 0.90);
END;
$
DELIMITER ;
```

Exemplo

Situação: Ao concretizar suas vendas, é essencial que o mercado automatize a atualização do estoque de produtos, reduzindo-o de forma imediata. A devolução ao estoque também deve ser automática, caso ocorra remoção do produto da venda.

Tabelas e inserções:

```
CREATE TABLE produtos (
        referencia VARCHAR(3) PRIMARY KEY,
        descricao VARCHAR(50) UNIQUE,
        estoque INT NOT NULL DEFAULT 0
);

INSERT INTO produtos VALUES ('001', 'Feijão', 10);
INSERT INTO produtos VALUES ('002', 'Arroz', 5);
INSERT INTO produtos VALUES ('003', 'Farinha', 15);
```

Tabelas e inserções:

```
CREATE TABLE itensvenda (
        venda INT PRIMARY KEY,
        produto VARCHAR(3),
        quantidade INT NOT NULL
);

INSERT INTO itensvenda VALUES (1, '001',3);
INSERT INTO itensvenda VALUES (2, '002',1);
INSERT INTO itensvenda VALUES (3, '003',5);
```

Trigger:

```
DELIMITER $
CREATE TRIGGER Tgr_ItensVenda_Insert AFTER INSERT
ON  itensvenda
FOR EACH ROW
BEGIN
        UPDATE produtos SET estoque = (estoque – NEW.quantidade)
        WHERE referencia = NEW.produto;
END;
```

Trigger:

```
CREATE TRIGGER Tgr_ItensVenda_Delete AFTER DELETE
ON  itensvenda
FOR EACH ROW
BEGIN
        UPDATE produtos SET estoque = (estoque + OLD.quantidade)
        WHERE referencia = OLD.produto;
END;
$
DELIMITER ;
```

Exercício 1

- Imagine uma tabela Comentarios e outra Estatisticas_Postagem. Crie um trigger que, ao excluir um comentário da tabela Comentarios, atualize automaticamente o contador de comentários na tabela Estatisticas_Postagem.

```
CREATE TABLE Comentarios (
        id_comentario INT PRIMARY KEY,
        fk_id_postagem INT,
        comentario TEXT,
        FOREIGN KEY (fk_id_postagem) REFERENCES
        Estatisticas_Postagem(id_postagem)
);
```

Exercício 1

```
CREATE TABLE Estatisticas_Postagem (
        id_postagem INT PRIMARY KEY,
        titulo VARCHAR(100),
        contador_comentarios INT DEFAULT 0
);
```

Exercício 2

- Considere uma tabela Vendas e outra tabela Total_Vendas. Crie um trigger que, ao atualizar o valor de uma venda na tabela Vendas, atualize automaticamente o valor total de vendas na tabela Total_Vendas. OBS: INSERIR PRODUTOS, VENDAS E CONTADOR DO TOTAL DE VENDAS PARA TESTAR TRIGGER DE UPDATE;

```
CREATE TABLE Vendas (
        id_venda INT PRIMARY KEY,
        fk_id_produto INT,
        valor DECIMAL(10, 2),
        FOREIGN KEY (fk_id_produto) REFERENCES
        Produtos(id_produto)
);
```

Exercício 2

```
CREATE TABLE Total_Vendas (
        id_total_vendas INT PRIMARY KEY,
        total DECIMAL(10, 2)
);

CREATE TABLE Produtos (
        id_produto INT PRIMARY KEY,
        nome VARCHAR(100),
        quantidade INT,
        preco DECIMAL(10, 2)
);
```

Exercício 3

- Quando um novo funcionário é contratado e inserido na tabela Funcionarios, o trigger deve atualizar automaticamente o contador de funcionários no departamento correspondente na tabela Total_Funcionarios. OBS: INSERIR DEPARTAMENTOS E IDs do total de FUNCIONARIOS PARA TESTAR TRIGGER;

```
CREATE TABLE Funcionarios (
        id_funcionario INT PRIMARY KEY,
        nome VARCHAR(100),
        fk_id_departamento INT,
        FOREIGN KEY (fk_id_departamento) REFERENCES
        Departamentos(id_departamento)
);
```

Exercício 3

```
CREATE TABLE Departamentos (
        id_departamento INT PRIMARY KEY,
        nome_departamento VARCHAR(100)
);

CREATE TABLE Total_Funcionarios (
        id_total_funcionarios INT PRIMARY KEY,
        total INT
);
```

Exercício 4

- Quando um funcionário é removido da tabela Funcionarios, o trigger deve atualizar automaticamente o contador de funcionários no departamento correspondente na tabela Total_Funcionarios.