Banco de Dados Avançados

Consultas avançadas em Banco de Dados

Aula 12

Profa. Ma. Fabiana A. Rodrigues

EDUCAÇÃO METODISTA

O que São Gatilhos?

- Na linguagem SQL, os gatilhos (triggers) são procedimentos armazenados especiais.
- São desencadeados automaticamente pelo Sistema de Gerenciamento de Banco de Dados (SGBD) em resposta a eventos específicos.

A Mecânica dos Gatilhos

- Os gatilhos são armazenados como objetos independentes na base de dados.
- São semelhantes a procedimentos (*stored procedures*), mas sua execução é acionada pelo SGBD em resposta a eventos do banco de dados.
- Não aceitam argumentos e são executados automaticamente.

Casos de Uso Comuns

- Gatilhos são usados para manter a consistência dos dados em resposta a eventos em tabelas específicas.
- Exemplo: Registro de alterações de clientes para manter um histórico.

Diferença entre Stored Procedures e Gatilhos

- Os gatilhos são um tipo especial de procedimento armazenado.
- A principal diferença: os gatilhos são acionados automaticamente em resposta a eventos, enquanto os stored procedures são chamados manualmente.

Sintaxe da Trigger

```
CREATE TRIGGER nome_do_gatilho
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nome_da_tabela
FOR EACH ROW
BEGIN
  -- Lógica da trigger
END;
```

https://www.devmedia.com.br/mysql-basico-triggers/37462

Sintaxe da Trigger

1. **CREATE TRIGGER nome_do_gatilho:** Isso define o início da criação da trigger. O nome_do_gatilho é o nome que você escolhe para a sua trigger. É uma convenção usar nomes descritivos que indiquem o propósito da trigger.

1. **{BEFORE | AFTER}:** Indica se a trigger deve ser acionada antes ou depois da ação específica. Você pode escolher entre "BEFORE" (antes) ou "AFTER" (depois) da ação que desencadeia a trigger.

1. **{INSERT | UPDATE | DELETE}:** Especifica o tipo de ação do banco de dados que acionará a trigger. Pode ser "INSERT" (inserção de dados), "UPDATE" (atualização de dados) ou "DELETE" (exclusão de dados).

1. **ON nome_da_tabela:** Indica em qual tabela a trigger será aplicada. A trigger responderá a eventos ocorridos nesta tabela.

1. **FOR EACH ROW:** Indica que a trigger será acionada para cada linha afetada pela ação específica (por exemplo, para cada linha inserida, atualizada ou excluída).

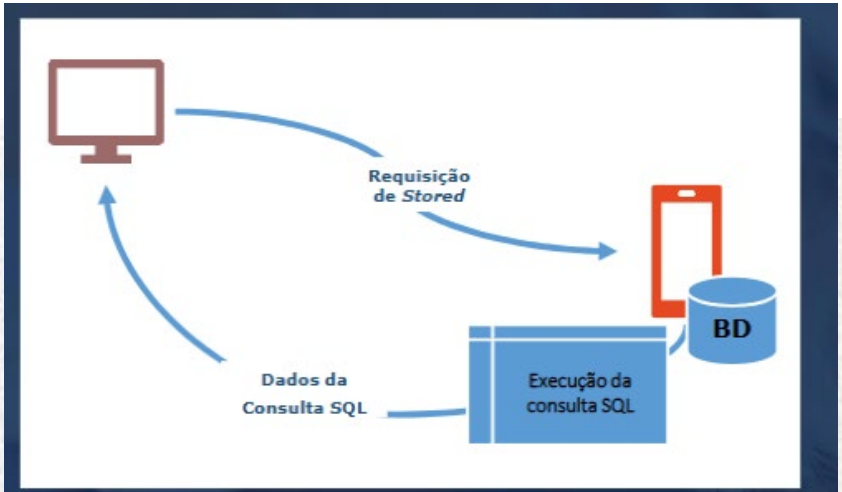
1. **BEGIN ... END:** Dentro deste bloco, você define a lógica da trigger. Este é o código que será executado quando a trigger for acionada. Você pode usar SQL para realizar operações desejadas, como atualizar outras tabelas, inserir registros em uma tabela de histórico, etc.

PRÁTICA

TRIGGERS

EDUCAÇÃO METODISTA

Uma Stored Procedure é um conjunto de comandos SQL que executa uma tarefa específica. Ao contrário de triggers, que são acionadas automaticamente, uma procedure requer ativação a partir de um programa ou manualmente pelo usuário.



Fonte: NOGUEIRA, 2017 p. 13

Os procedimentos armazenados (stored procedures) os processos são executados no servidor, ao invés de no navegador.

Benefícios das Stored Procedures

Reusabilidade de Código: uma stored procedure pode ser utilizada por vários usuários e em diferentes contextos, seja em scripts SQL, triggers ou aplicações.

Portabilidade: Stored procedures são altamente portáteis, permitindo uma fácil transferência entre sistemas.

Aprimoramento de Desempenho: armazenar a lógica da aplicação no banco de dados reduz o tráfego de rede em ambientes cliente/servidor, resultando em maior eficiência.

Manutenção Centralizada: ao manter o código no servidor, qualquer alteração feita é imediatamente refletida para todos os usuários, eliminando a necessidade de gerenciar versões distribuídas da aplicação.

Na transação SQL todo processamento acontece na estação de trabalho do usuário, ou seja, no próprio navegador.

Fonte: NOGUEIRA, 2017 p. 13

Sintaxes

Sintaxe de Criação:

```
CREATE PROCEDURE nome_procedimento (parâmetros) AS
BEGIN
  -- declarações
END;
```

```
DELIMITER $$
CREATE PROCEDURE prog()
BEGIN
   SELECT * FROM `funcionarios` ;
END $$
DELIMITER ;
```

Sintaxes

Stored Procedure

```sql
-- Exemplo de uma Stored Procedure para inserir um novo produto e atualizar o estoque
CREATE PROCEDURE InserirProduto(IN pNome VARCHAR(255), IN pPreco DECIMAL(10, 2), IN pQuantidade INT)
BEGIN
    -- Inserir um novo produto na tabela Produtos
    INSERT INTO Produtos (Nome, Preco, Quantidade) VALUES (pNome, pPreco, pQuantidade);

    -- Atualizar o estoque total
    UPDATE EstoqueTotal SET Quantidade = Quantidade + pQuantidade WHERE ProdutoID = LAST_INSERT_ID();
END;
```

CREATE PROCEDURE InserirProduto(IN pNome VARCHAR(255), IN pPreco DECIMAL(10, 2), IN pQuantidade INT): Esta linha cria uma nova stored procedure chamada "InserirProduto". Ela aceita três parâmetros de entrada: "pNome" (nome do produto), "pPreco" (preço do produto) e "pQuantidade" (quantidade em estoque).

BEGIN: A cláusula **BEGIN** marca o início do corpo da stored procedure, onde todas as ações a serem executadas são definidas.

INSERT INTO Produtos (Nome, Preco, Quantidade) VALUES (pNome, pPreco, pQuantidade);: Esta instrução SQL insere um novo registro na tabela "Produtos" com os valores fornecidos nos parâmetros "pNome", "pPreco" e "pQuantidade".

UPDATE EstoqueTotal SET Quantidade = Quantidade + pQuantidade WHERE ProdutoID = LAST_INSERT_ID();: Esta instrução SQL atualiza o estoque total na tabela "EstoqueTotal". Ela aumenta a quantidade existente em estoque com base na quantidade fornecida no parâmetro "pQuantidade". O **LAST_INSERT_ID()** é uma função que retorna o ID do último registro inserido na tabela "Produtos", garantindo que o estoque total seja atualizado para o produto correto.

END;: A cláusula END marca o final do corpo da stored procedure.

PRÁTICA

Stored Procedure

EDUCAÇÃO METODISTA

O que é uma Função em SQL?

- Uma **Função em SQL** é um bloco de código reutilizável que executa um conjunto de instruções e retorna um valor específico.
- As funções são frequentemente usadas para **simplificar consultas complexas**, **automatizar cálculos** e **promover reutilização de lógica**.

- **Funções Internas (Built-In Functions):** Fornecidas pelo sistema, como SUM(), AVG(), MAX(), MIN(), etc.
- **Funções Definidas pelo Usuário (User Defined Functions - UDFs):** Criadas pelos desenvolvedores para executar cálculos específicos.

Sintaxe de como criar uma função


```sql
DELIMITER //
CREATE FUNCTION NomeDaFuncao (parametros) RETURNS tipo_de_dado
BEGIN
    DECLARE variaveis;
    -- lógica ou instrução SQL para calcular e retornar um valor
    RETURN valor_calculado;
END //
DELIMITER ;
```

Delimiter: Define um novo delimitador para separar os comandos SQL no script. Por padrão, o delimitador SQL é ;, mas ao criar funções e procedimentos, precisamos definir blocos que contêm vários ;. Então, para evitar que o MySQL entenda ; como final de comando, mudamos temporariamente o delimitador para //.

CREATE FUNCTION: Indica que estamos criando uma nova função.

NomeDaFuncao: O nome da função que está sendo criada. Esse nome será usado para chamá-la posteriormente.

(parametros): Os parâmetros de entrada que a função pode aceitar. Eles são usados como **entradas para cálculos** ou operações dentro da função. Cada parâmetro precisa ter um nome e um tipo de dado.

RETURNS tipo_de_dado: Define **qual tipo de valor a função retornará**. Pode ser um tipo como `INT`, `DECIMAL`, `VARCHAR`, etc. Esse tipo de dado precisa ser compatível com o valor que será retornado ao final da função.

Sintaxe de como criar uma função


```sql
DELIMITER //
CREATE FUNCTION NomeDaFuncao (parametros) RETURNS tipo_de_dado
BEGIN
    DECLARE variaveis;
    -- lógica ou instrução SQL para calcular e retornar um valor
    RETURN valor_calculado;
END //
DELIMITER ;
```

BEGIN ... END: Marca o **bloco principal de código** da função, onde será definida toda a lógica.

`BEGIN`: Início do bloco de execução da função.

`END`: Final do bloco. Tudo entre `BEGIN` e `END` faz parte da lógica da função.

DECLARE: Declara **variáveis locais** que podem ser usadas dentro da função.

variaveis: Aqui, você define variáveis com nomes e tipos específicos, que servirão para armazenar valores intermediários, realizar cálculos, ou armazenar resultados temporários.

Sintaxe de como criar uma função


```sql
DELIMITER //
CREATE FUNCTION NomeDaFuncao (parametros) RETURNS tipo_de_dado
BEGIN
    DECLARE variaveis;
    -- lógica ou instrução SQL para calcular e retornar um valor
    RETURN valor_calculado;
END //
DELIMITER ;
```

Comentário (`--`): Os comentários ajudam a descrever o que está acontecendo.

Lógica ou Instrução SQL: Neste ponto, você coloca a lógica que deseja executar. Pode ser uma consulta `SELECT`, uma atribuição de valor à variável (`SET`), ou outras instruções que permitem processar os dados de entrada.

RETURN: A função **retorna um valor**. Esse valor precisa ser do tipo definido na linha `RETURNS tipo_de_dado`.

valor_calculado: O valor final que será retornado pela função. Pode ser uma variável que foi calculada ao longo do código dentro de `BEGIN` e `END`.