Triggers e Stored Procedures no Banco de Dados

Professor: Jorge Baldez

Introdução

Triggers e **Stored Procedures** são mecanismos importantes em bancos de dados relacionais.

Eles automatizam processos e reforçam a integridade e segurança dos dados.

Triggers: Explicação e Exemplo em SQL

- **Triggers** são **ações automáticas** que ocorrem em resposta a eventos específicos no banco de dados, como **INSERT**, **UPDATE**, ou **DELETE**. Quando algum desses eventos acontece em uma tabela ou view associada, o trigger é acionado automaticamente, sem a necessidade de intervenção manual.
- Triggers são usados para automatizar tarefas, garantir a integridade dos dados, e executar processos que precisam acontecer sempre que há alterações em uma tabela.

Triggers: Explicação e Exemplo em SQL

- **Triggers** são **ações automáticas** que ocorrem em resposta a eventos específicos no banco de dados, como **INSERT**, **UPDATE**, ou **DELETE**. Quando algum desses eventos acontece em uma tabela ou view associada, o trigger é acionado automaticamente, sem a necessidade de intervenção manual.
- Triggers são usados para automatizar tarefas, garantir a integridade dos dados, e executar processos que precisam acontecer sempre que há alterações em uma tabela.

Triggers: Explicação e Exemplo em SQL

- **Principais eventos que disparam uma Trigger:**
  - **INSERT**: Quando novos dados são inseridos em uma tabela.
  - **UPDATE**: Quando um dado existente é modificado.
  - **DELETE**: Quando um registro é removido de uma tabela.

Triggers: Explicação e Exemplo em SQL

- **Melhorias e Contextos de Uso:**

- Validação de Dados: As triggers podem verificar se os dados inseridos ou atualizados atendem a certas condições antes de efetivar as alterações.
- Auditoria e Log: Podem ser usadas para registrar automaticamente quem fez mudanças e quais foram as mudanças em uma tabela.
- Manutenção de Consistência: Atualizam automaticamente outras tabelas que dependem dos dados alterados.

Triggers: Explicação e Exemplo em SQL

- Suponha que você tenha um sistema bancário, onde sempre que uma nova transação é inserida em uma tabela de transações, o saldo da conta precisa ser atualizado na tabela de contas. Nesse caso, uma trigger pode ser criada para automatizar esse processo.

```sql
CREATE TRIGGER atualiza_saldo
AFTER INSERT ON transacoes
FOR EACH ROW
BEGIN
  -- Atualiza o saldo da conta após uma nova transação
  UPDATE conta_corrente
  SET saldo = saldo + NEW.valor
  WHERE conta_id = NEW.conta_id;
END;
```

Explicação do Código:

- **AFTER INSERT**: Indica que a trigger será acionada **após** a inserção de uma nova transação na tabela transacoes.
- **FOR EACH ROW**: Define que a trigger será executada para cada linha inserida.
- **NEW.valor**: Refere-se ao valor da nova transação que acabou de ser inserida.
- **NEW.conta_id**: Refere-se ao ID da conta associada à nova transação.

Triggers: Explicação e Exemplo em SQL

- **Mais Um Exemplo: Trigger para Auditoria**
- Imagine que você deseja registrar automaticamente qualquer alteração feita na tabela usuarios em uma tabela de logs de auditoria.

```sql
CREATE TRIGGER log_alteracao_usuario
AFTER UPDATE ON usuarios
FOR EACH ROW
BEGIN
  INSERT INTO log_auditoria
  (usuario_id, data_modificacao, campo_modificado, valor_antigo, valor_novo)
  VALUES (OLD.id, NOW(), 'email', OLD.email, NEW.email);
END;
```

Explicação:

- **AFTER UPDATE**: A trigger é acionada após uma atualização na tabela usuarios.
- **OLD.email**: Refere-se ao valor antigo do campo email antes da atualização.
- **NEW.email**: Refere-se ao novo valor do campo email após a atualização.
- **NOW()**: Insere a data e hora atuais da modificação.

Neste caso, a trigger registra a alteração de e-mails dos usuários na tabela log_auditoria, mantendo um histórico de alterações de dados.

Triggers: Explicação e Exemplo em SQL

- **Vantagens:**
  - Automação: Nenhum código adicional é necessário no sistema para atualizar saldos, fazer auditoria ou outras ações que as triggers executam.
  - Integridade: Triggers garantem que certas regras de negócios (como atualização de saldos ou logs de auditoria) são sempre seguidas de forma consistente.
- Essa automação e flexibilidade tornam as triggers uma ferramenta poderosa para manter dados consistentes e automatizar processos em bancos de dados.

O que são Stored Procedures?

- **Stored Procedures** são blocos de código SQL pré-compilados e armazenados diretamente no banco de dados, prontos para serem executados sob demanda. Eles permitem encapsular um conjunto de instruções SQL dentro de um procedimento que pode ser chamado repetidamente, sem a necessidade de reescrever ou enviar essas instruções a cada vez. Isso melhora a organização do código e o desempenho do banco de dados.

O que são Stored Procedures?

- **Características Principais:**

- **Armazenamento e Reutilização**: Uma vez criada, uma stored procedure pode ser reutilizada por várias partes da aplicação ou sistema, sem a necessidade de reescrever as mesmas instruções.

- **Execução Controlada**: A stored procedure é executada explicitamente pelo desenvolvedor, aplicação, ou outro processo automatizado, como triggers.

O que são Stored Procedures?

- **Lógica Complexa**: Além de simples instruções SQL (como SELECT, INSERT, UPDATE, DELETE), stored procedures podem conter estruturas mais avançadas, como:
  - **Condicionais**: IF, CASE
  - **Laços de repetição**: LOOP, WHILE
  - **Tratamento de exceções**: HANDLER
  - **Manipulação de cursores**: Para percorrer resultados linha a linha
  - **Parâmetros de entrada e saída**: Facilitando a passagem de dados para dentro e fora da procedure

Vantagens de Stored Procedures

Melhor Desempenho: Como são pré-compiladas, stored procedures executam mais rápido do que consultas SQL ad-hoc, pois o plano de execução já está armazenado no servidor.

Segurança: O acesso direto às tabelas pode ser restringido, permitindo que usuários ou aplicações interajam com os dados apenas por meio das stored procedures. Isso aumenta o controle e minimiza riscos.

Modularidade: Permitem que a lógica de negócios do banco de dados seja separada do código da aplicação, melhorando a manutenção e o reuso.

Redução de Tráfego na Rede: Em vez de enviar múltiplas consultas separadas do lado da aplicação, é possível consolidar várias operações em uma única chamada de stored procedure, reduzindo o tráfego entre o cliente e o servidor.

Tipos de Stored Procedures

- **Procedures com Parâmetros**: Recebem dados para serem processados e podem retornar resultados.
- **Procedures Sem Retorno**: Executam ações no banco de dados sem devolver dados diretamente.
- **Procedures com Valores de Retorno**: Retornam um valor ou conjunto de valores, semelhante a funções em linguagens de programação.

Tipos de Stored Procedures

- **Procedures com Parâmetros**: Recebem dados para serem processados e podem retornar resultados.
- **Procedures Sem Retorno**: Executam ações no banco de dados sem devolver dados diretamente.
- **Procedures com Valores de Retorno**: Retornam um valor ou conjunto de valores, semelhante a funções em linguagens de programação.

Exemplo de Stored Procedure Simples

- Um exemplo básico de uma stored procedure para atualizar a senha de um usuário:

```
CREATE PROCEDURE altera_senha_usuario(IN usuario_id INT, IN nova_senha VARCHAR(255))
BEGIN
  UPDATE usuarios
  SET senha = nova_senha
  WHERE id = usuario_id;
END;
```

```
CALL altera_senha_usuario(1, 'novaSenha123');
```

Explicação:

- IN: Define parâmetros de entrada (nesse caso, usuario_id e nova_senha).
- UPDATE: A procedure executa uma atualização na tabela usuarios para mudar a senha do usuário com o id fornecido.

Esse procedimento pode ser chamado de maneira simples, passando o ID do usuário e a nova senha, eliminando a necessidade de reescrever a lógica de atualização em várias partes do sistema.

Exemplo de Stored Procedure com Lógica Complexa

- Agora, um exemplo mais avançado, onde a procedure insere um novo pedido, verifica o estoque, e se for insuficiente, retorna um erro:

```
CREATE PROCEDURE processa_pedido(IN produto_id INT, IN quantidade INT)
BEGIN
  DECLARE estoque_atual INT;

  -- Verifica o estoque atual do produto
  SELECT estoque INTO estoque_atual FROM produtos WHERE id = produto_id;

  -- Verifica se o estoque é suficiente
  IF estoque_atual >= quantidade THEN
    -- Insere o pedido
    INSERT INTO pedidos (produto_id, quantidade) VALUES (produto_id, quantidade);

    -- Atualiza o estoque
    UPDATE produtos SET estoque = estoque - quantidade WHERE id = produto_id;
  ELSE
    -- Lança um erro se o estoque for insuficiente
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Estoque insuficiente!';
  END IF;
END;
```

Explicação:

1. DECLARE: Declara uma variável para armazenar o valor do estoque atual.
2. SELECT INTO: Armazena o valor do estoque atual da tabela produtos na variável estoque_atual.
3. IF: Verifica se o estoque é suficiente para processar o pedido.
4. SIGNAL: Gera um erro personalizado se o estoque for insuficiente, avisando o usuário ou sistema.

Essa procedure centraliza toda a lógica de verificação de estoque e inserção de pedidos, assegurando que a operação seja realizada de forma consistente e segura.

Parâmetros de Entrada e Saída

- Parâmetros de Entrada (IN): Fornecem valores para a procedure.
- Parâmetros de Saída (OUT): Retornam resultados da procedure.
- Parâmetros INOUT: São usados tanto para fornecer como para retornar valores.

Exemplo com Parâmetros de Saída

```sql
CREATE PROCEDURE calcular_total_pedidos(IN cliente_id INT, OUT total DECIMAL(10, 2))
BEGIN
  -- Calcula o total de pedidos do cliente
  SELECT SUM(valor) INTO total FROM pedidos WHERE id_cliente = cliente_id;
END;
```

Explicação:

- **IN: Recebe o ID do cliente.**
- **OUT: Retorna o total calculado dos pedidos do cliente através do parâmetro total.**

A chamada da procedure seria

```
CALL calcular_total_pedidos(123, @total);
SELECT @total;  -- Exibe o valor calculado
```

Conclusão:

Stored procedures são ferramentas extremamente poderosas e versáteis para automatizar e encapsular a lógica de negócios diretamente no banco de dados. Elas oferecem melhor desempenho, segurança, e simplificam o desenvolvimento de aplicações robustas e escaláveis.