Introdução

O que é uma Procedure?

Uma Procedure (ou Procedimento Armazenado) é um conjunto de instruções SQL que podem ser armazenadas no banco de dados e reutilizadas várias vezes. Ela permite encapsular lógica complexa em uma unidade que pode ser executada com um único comando.

Vantagens de Usar Procedures

Reutilização de Código: Procedures podem ser chamadas várias vezes em diferentes partes da aplicação, promovendo a reutilização de código.

Performance: Procedures são compiladas e armazenadas no banco de dados, o que pode resultar em uma execução mais rápida em comparação com scripts SQL ad-hoc.

Manutenção: Alterar uma Procedure centralizada é mais fácil do que alterar múltiplos scripts SQL espalhados pela aplicação.

Segurança: Permite encapsular lógica de negócios, restringindo o acesso direto às tabelas e dados.

Estrutura Básica de uma Procedure

A seguir, veremos como criar uma Procedure básica no MySQL.

Sintaxe de Criação

```
DELIMITER //

CREATE PROCEDURE nome_da_procedure(parametros)
BEGIN
  -- Corpo da Procedure
  -- Instruções SQL aqui
END //

DELIMITER ;
```

Exemplo Prático

Vamos criar uma Procedure simples que retorna todos os registros de uma tabela de exemplo chamada alunos.

```
DELIMITER //

CREATE PROCEDURE listar_alunos()
BEGIN
  SELECT * FROM alunos;
END //

DELIMITER ;
```

Executando uma Procedure

Para executar a Procedure criada, utilizamos o comando CALL:

```
CALL listar_alunos();
```

Procedure com Parâmetros

Procedures podem aceitar parâmetros de entrada e de saída. Vamos criar uma Procedure que busca um aluno pelo seu ID.

Exemplo com Parâmetros de Entrada

```
DELIMITER //

CREATE PROCEDURE buscar_aluno_por_id(IN aluno_id INT)
BEGIN
  SELECT * FROM alunos WHERE id = aluno_id;
END //

DELIMITER ;
```

Para executar:

```
CALL buscar_aluno_por_id(1);
```

Exemplo com Parâmetro de Saída

```
DELIMITER //

CREATE PROCEDURE contar_alunos(OUT total_alunos INT)
BEGIN
  SELECT COUNT(*) INTO total_alunos FROM alunos;
END //

DELIMITER ;
```

Para executar e obter o valor do parâmetro de saída:

```
CALL contar_alunos(@total);
SELECT @total;
```

Controle de Fluxo em Procedures

IF...ELSE

```
DELIMITER //

CREATE PROCEDURE verificar_idade(IN idade INT, OUT resultado VARCHAR(50))
BEGIN
  IF idade < 18 THEN
    SET resultado = 'Menor de idade';
  ELSE
    SET resultado = 'Maior de idade';
  END IF;
END //

DELIMITER ;

LOOP

DELIMITER //

CREATE PROCEDURE contar_ate_dez()
BEGIN
  DECLARE i INT DEFAULT 1;

  loop_inicio: LOOP
    IF i > 10 THEN
      LEAVE loop_inicio;
    END IF;

    SELECT i;
    SET i = i + 1;
  END LOOP loop_inicio;
END //

DELIMITER ;
```

Conclusão

Procedures são uma poderosa ferramenta no MySQL para encapsular lógica de negócios, melhorar a performance e a manutenção do código. Nesta aula inicial, cobrimos os conceitos básicos e exemplos práticos de criação e uso de Procedures. Em aulas futuras, exploraremos tópicos avançados como tratamento de erros, transações e uso de cursores.

Exemplos banco erp_senai:

Procedure para Listar Todas as Agências Bancárias

```
DELIMITER //

CREATE PROCEDURE listar_agencias()
BEGIN
  SELECT * FROM agencia_banco;
END //

DELIMITER ;
```

Procedure para Buscar Agências Bancárias por Banco

```
DELIMITER //

CREATE PROCEDURE buscar_agencias_por_banco(IN banco_id INT)
BEGIN
  SELECT * FROM agencia_banco WHERE BANCO_ID = banco_id;
END //

DELIMITER ;
```

Procedure para Inserir Nova Agência Bancária

```
DELIMITER //

CREATE PROCEDURE inserir_agencia(
  IN p_CEP_ID INT,
  IN p_BANCO_ID INT,
  IN p_CODIGO INT,
  IN p_NOME VARCHAR(100),
  IN p_ENDERECO VARCHAR(100),
```

```
    IN p_TELEFONE VARCHAR(10),
    IN p_GERENTE VARCHAR(30),
    IN p_CONTATO VARCHAR(30),
    IN p_OBS TEXT
)
BEGIN
    INSERT INTO agencia_banco (CEP_ID, BANCO_ID, CODIGO, NOME, ENDERECO, TELEFONE, GERENTE,
CONTATO, OBS)
    VALUES (p_CEP_ID, p_BANCO_ID, p_CODIGO, p_NOME, p_ENDERECO, p_TELEFONE, p_GERENTE,
p_CONTATO, p_OBS);
END //

DELIMITER ;
```

Procedure para Atualizar Informações de uma Agência Bancária

```
DELIMITER //

CREATE PROCEDURE atualizar_agencia(
    IN p_ID INT,
    IN p_CEP_ID INT,
    IN p_BANCO_ID INT,
    IN p_CODIGO INT,
    IN p_NOME VARCHAR(100),
    IN p_ENDERECO VARCHAR(100),
    IN p_TELEFONE VARCHAR(10),
    IN p_GERENTE VARCHAR(30),
    IN p_CONTATO VARCHAR(30),
    IN p_OBS TEXT
)
BEGIN
    UPDATE agencia_banco
    SET CEP_ID = p_CEP_ID,
        BANCO_ID = p_BANCO_ID,
        CODIGO = p_CODIGO,
        NOME = p_NOME,
        ENDERECO = p_ENDERECO,
```

```
    TELEFONE = p_TELEFONE,
    GERENTE = p_GERENTE,
    CONTATO = p_CONTATO,
    OBS = p_OBS
  WHERE ID = p_ID;
END //

DELIMITER ;

Procedure para Excluir uma Agência Bancária
DELIMITER //

CREATE PROCEDURE excluir_agencia(IN p_ID INT)
BEGIN
  DELETE FROM agencia_banco WHERE ID = p_ID;
END //

DELIMITER ;

Procedure para Registrar Alterações na Tabela auditoria
DELIMITER //

CREATE PROCEDURE registrar_auditoria(
  IN p_data_hora DATETIME,
  IN p_tabela VARCHAR(100),
  IN p_registro_id INT,
  IN p_antigo VARCHAR(100),
  IN p_novo VARCHAR(100)
)
BEGIN
  INSERT INTO auditoria (data_hora, tabela, registro_id, antigo, novo)
  VALUES (p_data_hora, p_tabela, p_registro_id, p_antigo, p_novo);
END //

DELIMITER ;
```

Procedure para Listar Todos os Clientes

```
DELIMITER //

CREATE PROCEDURE listar_clientes()
BEGIN
  SELECT * FROM cliente;
END //

DELIMITER ;
```

Procedure para Buscar Cliente por Nome

```
DELIMITER //

CREATE PROCEDURE buscar_cliente_por_nome(IN nome_cliente VARCHAR(150))
BEGIN
  SELECT * FROM cliente WHERE NOME LIKE CONCAT('%', nome_cliente, '%');
END //

DELIMITER ;
```

Procedure para Atualizar Salário de um Cargo e Registrar Log

```
DELIMITER //

CREATE PROCEDURE atualizar_salario(
  IN p_ID INT,
  IN p_salario_novo DOUBLE
)
BEGIN
  DECLARE p_salario_antigo DOUBLE;

  -- Captura o salário antigo
  SELECT SALARIO INTO p_salario_antigo FROM cargo WHERE ID = p_ID;

  -- Atualiza o salário
```

```
    UPDATE cargo SET SALARIO = p_salario_novo WHERE ID = p_ID;

    -- Registra no log de alteração de salário
    INSERT INTO log_cargo (cargo_id, salario_antigo, salario_novo)
    VALUES (p_ID, p_salario_antigo, p_salario_novo);
END //

DELIMITER ;
```