

# Aula 15 - Stored Procedures e Functions

🕒 Created	@January 28, 2022 10:32 AM
▼ Class	
▼ Type	
🔗 Materials	
☑ Reviewed	<input type="checkbox"/>
☰ Property	

## Stored Procedures

São MÓDULOS (blocos de comandos SQL) armazenados de modo persistente e executados pelo SGBD no SERVIDOR de banco de dados.

### Quando devemos utilizá-los?

1. Quando tem-se **várias aplicações** escritas em **diferentes linguagens**, ou rodam em **plataformas diferentes**, porém **EXECUTAM** a mesma **função** (procedimento) em um mesmo banco de dados.  
**CENTRALIZAÇÃO** - Facilidade de Manutenção / Melhora a modularidade do software.
2. Executar um programa no servidor pode **REDUZIR** o **tráfego de informações** na rede e, conseqüentemente, em certas situações, os **custos de comunicação** entre os clientes e o servidor.  
**VELOCIDADE** – Melhor desempenho / performance de rede; Evita a redundância de códigos.
3. Quando damos prioridade à **CONSISTÊNCIA** e **SEGURANÇA**.  
**SEGURANÇA** - Os bancos (Itaú, Bradesco, etc), por exemplo, em geral, utilizam Stored Procedures para todas as **operações em comum**. Os de

forma **correta** e procedimentos podem assegurar que as operações sejam registradas de maneira **segura**.

4. Quando quer garantir que toda a operação seja executada, ou caso algum erro ocorra, que nenhuma ação seja salva no banco de dados de forma inconsistente.

### **SUPORTE A TRANSAÇÕES.**

**Vantagens:** Pode reduzir o tráfego na rede, melhorar a performance de um banco de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processamento, etc.

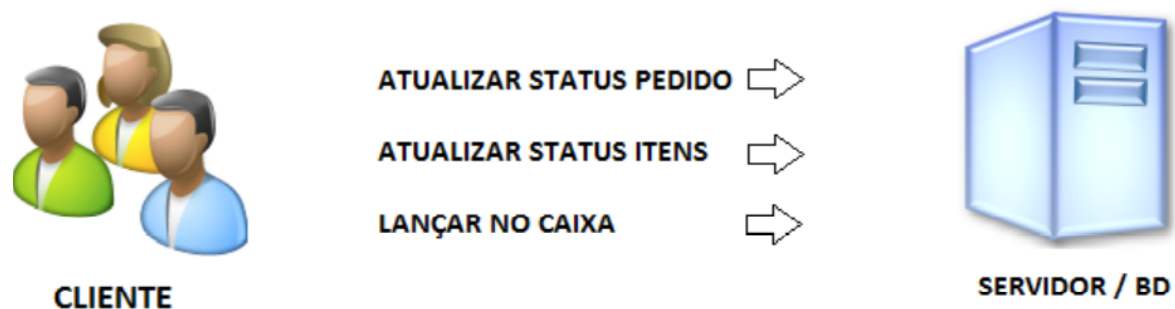
**Desvantagens:** O lado negativo dos procedimentos armazenados é que exigirá muito mais PROCESSAMENTO do lado do servidor.

## **Exemplo**

**Imagine o seguinte escopo:**

- O cliente faz um pedido, no qual são inseridos itens;
- O pedido (bem como os itens) permanece com status “PENDENTE” até ser confirmado;
- O operador confirma o pedido, registrando o movimento no caixa.

**Figura 1: Execução da rotina sem stored procedure**



Fonte: <https://www.devmedia.com.br/>

Por outro lado:

Poderíamos agrupar essas três instruções no corpo de um procedimento e chamá-lo a partir da aplicação uma única vez. As ações de update/insert/delete, a partir daí, ficariam por conta do servidor.

**Figura 2:** Execução da rotina com stored procedure



Fonte: <https://www.devmedia.com.br/>

## Stored Procedure: Criar, Chamar e Deletar

```
# Criando
DELIMITER $$
CREATE PROCEDURE cadProd(IN nome_produto VARCHAR(45))
BEGIN
    INSERT INTO produto(nome) VALUES (nome_produto);
END $$
DELIMITER ;

# Chamando
CALL cadProd('Mouse');

# Deletando
DROP PROCEDURE cadProd;
```

## Stored Procedure: Parâmetros e Corpo do Procediemento

## Parâmetros

**IN** - parâmetro de entrada, ou seja, um parâmetro cujo seu valor será utilizado no interior do procedimento para produzir algum resultado;

**OUT** - parâmetro que retorna algo para dentro do procedimento para o lado externo, colocando os valores manipulados disponíveis na memória ou no conjunto de resultados;

**INOUT** - faz os dois trabalhos simultaneamente.

**OBS: Os parâmetros são opcionais, e só são específicos se for preciso.**

## Corpo do procedimento

Onde são definidos os comandos SQL que farão alguma manipulação e/ou defenderão alguma lógica, podendo retornar ou não algum resultado.

## Stored Procedure: Exemplo Simples

```
# criando procedimento que soma dois números
DELIMITER $$
DROP PROCEDURE IF EXISTS soma $$
CREATE PROCEDURE soma(IN num1 INT, IN num2 INT)
BEGIN
    #calculando a soma e exibindo o resultado
    select (num1 + num2) as Soma;
END $$
DELIMITER ;

#chamando o procedimento soma, passando como parâmetros os valores 2 e 3
call soma(2,3);
```

## Stored Procedure: If e Elseif

### ➤ Syntax:

```
IF <condição> THEN <lista de declarações>
    ELSEIF <condição> THEN <lista de declarações>
END IF
```

```

#criando o procedimento testeif para verificar se um número é positivo ou negativo
DELIMITER $$
DROP PROCEDURE IF EXISTS testeif $$
CREATE PROCEDURE testeif(IN num1 INT, IN num2 INT)
BEGIN
    #criando uma variável do tipo int para armazenar uma soma de inteiros
    DECLARE soma INT;
    #criando uma variável que armazenará o valor da resposta do resultado
    DECLARE resposta varchar(15);
    # atribuindo um valor para a variável criada
    SET soma = num1 + num2;
    #verificando se o valor é positivo
    IF soma > 0
        # atribuindo a string 'positivo' para a variável de "resposta"
        THEN SET resposta = 'positivo';
    ELSE
        # atribuindo a string 'negativo' para a variável de "resposta"
        SET resposta = 'negativo';
    END IF;

    #exibindo o resultado
    select concat('O resultado da soma resulta em um numero: ', resposta) as Resposta;
END $$
DELIMITER ;

```

## Stored Procedures: While

### Syntax:

```

WHILE <condição> DO
    <lista de declarações>
END WHILE;

```

```

#criando um procedimento com comando while para calcular fatorial
DELIMITER $$
DROP PROCEDURE IF EXISTS testeWhile $$
CREATE PROCEDURE testeWhile (IN num INT)
BEGIN
    DECLARE res int;
    SET res = 1;

    WHILE num > 0 DO
        SET res = res * num;
        SET num = num - 1;
    END WHILE;

```

```
SELECT res as FATORIAL;  
END $$  
DELIMITER ;  
  
call testWhile(3);
```

# Functions

## Introdução



Também são MÓDULOS (blocos de comandos SQL) armazenados de modo persistente e executados pelo SGBD no SERVIDOR do banco de dados.

- **Objetivo:** Realizar pequenas operações, normalmente auxiliares, que possam ser solicitadas em um processo de transação;  
E não realizar transações de negócio completas (Stored Procedures).
- **Responsáveis** pelo tratamento de textos e variáveis, formatação, operações repetitivas e rotineiras e que possam ser compartilhadas para os Stored Procedures e/ou Aplicações;  
Enquanto os Stored Procedures se encarregam da lógica de negócio e do gerenciamento de transações.

→ Retornam **VALORES** e **PARÂMETROS**.

Enquanto **Stored Procedures** não possuem valor de retorno, mas podem gerar conjuntos de resultados e retorná-los via parâmetros.

→ Podem utilizar-se de estruturas **CONDICIONAIS** (desvios) e estruturas de **LAÇOS** (loops).

```
IF <condição> THEN <lista de declarações>
    ELSEIF <condição> THEN <lista de declarações>
END IF
```

```
WHILE <condição> DO
    <lista de declarações>
END WHILE;
```

## Como declarar e chamar?

### 1) Forma de **declarar** a **Function**:

```
DELIMITER $$
DROP FUNCTION IF EXISTS nome_função $$
CREATE FUNCTION nome_função (<parâmetros>) RETURNS INT
BEGIN
<corpo da função>
END $$
DELIMITER ;
```

### 2) Forma de **chamar** a **Function**:

```
SELECT nome_função (<parâmetros>);
```

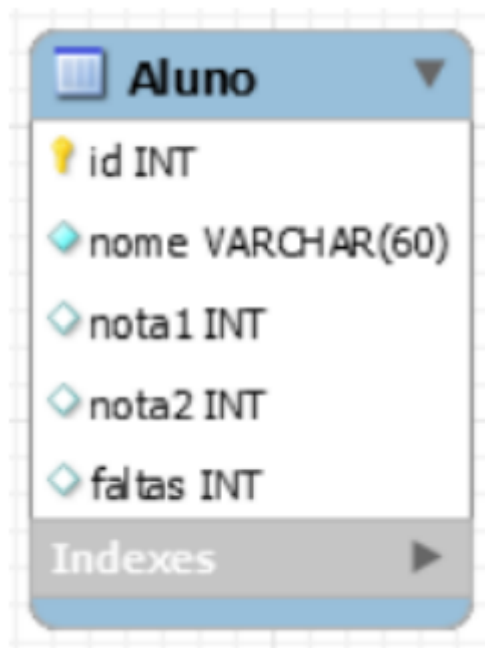
## Exemplo 1 - Function

```
# função que multiplica dois números
DELIMITER $$
DROP FUNCTION IF EXISTS mult $$
CREATE FUNCTION mult(a FLOAT, b FLOAT) RETURNS FLOAT
deterministic
BEGIN
    return a*b;
END $$
DELIMITER ;

select mult (2.5,4);
```

## Exemplo 2 - Function

Criar uma função que retorne qual a situação Final de um aluno (Aprovado, Reprovado ou NP3).



As condições são:

- Média = (nota1 + nota2)/2;
- Se Média >= 60 e Faltas <= 10: Aprovado
- Se Média < 30 ou Faltas > 10: Reprovado
- Se Média entre 30 e 59 e Faltas <= 10: NP3



## Resolução:

```
CREATE TABLE IF NOT EXISTS `dbEscola`.`Aluno` (  
  `id` INT NOT NULL,  
  `nome` VARCHAR(60) NOT NULL,  
  `nota1` INT NULL,  
  `nota2` INT NULL,  
  `faltas` INT NULL,  
  PRIMARY KEY (`id`))  
ENGINE = InnoDB;  
  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('1', 'Ana', '70', '70', '2');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('2', 'Pedro', '50', '40', '15');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('3', 'Carlos', '60', '75', '10');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('4', 'Joana', '25', '20', '0');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('5', 'Maria', '100', '98', '2');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('6', 'Luis', '30', '30', '4');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('7', 'Patricia', '50', '20', '0');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('8', 'João', '58', '59', '8');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('9', 'Márcio', '100', '20', '2');  
INSERT INTO `dbescola`.`aluno` (`id`, `nome`, `nota1`, `nota2`, `faltas`) VALUES ('10', 'Reinaldo', '40', '40', '10');  
  
DELIMITER $$  
DROP FUNCTION IF EXISTS SituacaoAluno $$  
CREATE FUNCTION SituacaoAluno(pNota1 int, pNota2 int, pFaltas int)  
RETURNS VARCHAR(20)  
BEGIN  
  DECLARE situacao VARCHAR(20);  
  DECLARE media int;  
  set media = (pNota1 + pNota2)/2;  
  IF media >= 60 AND pFaltas <= 10 THEN  
    SET situacao = 'APROVADO';  
  ELSEIF (media < 30 OR pFaltas > 10) THEN  
    SET situacao = 'REPROVADO';  
  ELSEIF ((media >= 30 AND media < 60) and pFaltas <= 10 ) THEN  
    SET situacao = 'NP3';  
  END IF;  
  RETURN (situacao);  
END$$  
DELIMITER ;  
  
SELECT nome AS "NOME ALUNO", situacaoAluno(nota1, nota2, faltas) AS "RESULTADO FINAL"  
FROM Aluno order by nome;
```

# Stored Procedure: Exercício

## 1) Elaborar as seguintes Stored Procedures para a tabela Aluno:

- Inserir alunos;
- Deletar registros de alunos (pelo ID);
- Editar todos os dados de alunos (pelo ID);

```
DROP DATABASE IF EXISTS aula_procedure;
CREATE DATABASE aula_procedure;
USE aula_procedure;

CREATE TABLE Aluno(
    id int not null auto_increment primary key,
    nome varchar(50),
    idade int,
    email varchar(100)
);
```

**OBS: Utilizem o DELIMITER como nos exemplos anteriores!**

## 2) Exercício para entregar:

```
DROP DATABASE IF EXISTS loja;
CREATE DATABASE loja;
USE loja;
SET GLOBAL log_bin_trust_function_creators = 1;
CREATE TABLE compra(
    id int not null auto_increment primary key,
    preco float,
    pagamento float
);

INSERT INTO compra VALUES (id,9.5,10.25);
INSERT INTO compra VALUES (id,18.99,25);
INSERT INTO compra VALUES (id,3.99,5);
INSERT INTO compra VALUES (id,8.85,8.89);
INSERT INTO compra VALUES (id,19.49,20);
```

Utilizando o **DATABASE** acima, realize as seguintes operações:

- Crie uma **FUNCTION** que calcule o troco (pagamento - preço) da compra e:
  - Se troco < 0.05, retorne “**Sem troco**”;

- Se troco  $\leq 1$ , retorne “**Balinhas de café**”; ou
- Retorne “**Em dinheiro**”;
- BUSQUE pelo troco de todas as compras, utilizando a **FUNCTION** criada(semelhante a imagem do Exemplo 2);