

# Conceitos de Banco de Dados

## Aula 08 - Linguagem SQL - Stored Procedures e Fun es

# Apresentação

---

Na aula anterior, conhecemos como utilizar subconsultas SQL e discutimos como analisar dados armazenados em várias tabelas de um modo mais simplificado usando o conceito de visões. Nesta aula, estudaremos o conceito de Stored Procedures, ou procedimentos armazenados, usados para definir operações extremamente úteis para manipulação e acesso de dados em um sistema de banco de dados. Aprenderemos a criar, executar e apagar essas estruturas, bem como utilizá-las usando o recurso de passagem de parâmetros. E, por fim, discutiremos como utilizar estruturas de controle de fluxo de dados, comuns em linguagem de programação, dentro desses procedimentos, aumentando assim a versatilidade de uso dessas estruturas.

Vamos ainda estudar o conceito de funções, que são usadas para encapsular operações úteis para manipulação do acesso de dados em um sistema de banco de dados. Aprenderemos a criar, executar e apagar essas estruturas, bem como utilizá-las no processamento de consultas. Também será discutido como se pode declarar variáveis no SQL.

## Objetivos

- Criar procedimentos armazenados com ou sem parâmetros no sistema MySQL.
- Utilizar as estruturas de controle de fluxo de dados, como estruturas IF e WHILE, na criação de procedimentos armazenados.
- Criar funções com e sem parâmetros no sistema MySQL.
- Declarar variáveis locais que podem ser utilizadas dentro das funções.
- Utilizar as funções para auxílio no processamento de consultas.

## Stored procedures

---

No dia a dia do trabalho com sistemas de banco de dados, surgem problemas relacionados à necessidade de se realizar operações complexas que envolvem a realização de um ou mais comandos de consulta a tabelas em conjunto com comandos de inclusão e/ou alteração de dados. Em outros casos, deseja-se realizar operações com base em parâmetros definidos pelo usuário.

Nessas situações, o ideal é que essas operações não sejam definidas na aplicação que vai fazer uso do banco de dados, mas no próprio servidor do banco de dados. Isso garante a segurança do banco de dados, uma vez que o mesmo pode controlar o acesso a essas operações especiais, evitando o conhecimento excessivo de sua estrutura pelos desenvolvedores de aplicações. Desse modo, as *Stored procedures*, traduzidas do inglês como procedimentos armazenados, são definidas como um conjunto de comandos da linguagem SQL definidos no servidor do banco de dados e acionados por eventuais chamadas de qualquer usuário que tenha autorização para sua execução.

Além da questão de segurança do banco de dados, já que o usuário só poderá executar os procedimentos a que tiver acesso, os procedimentos armazenados ajudam a diminuir o tráfego de informações entre o usuário e o servidor, tornando mais simples o processamento das informações. Por exemplo, suponha que diversos usuários desejem fazer a consulta especificada no destaque a seguir. Essa consulta retorna uma lista de produtos vendidos e se refere ao banco de dados **sistvendas** apresentado na **Aula 7**.

```
1 mysql> SELECT prod_nome
2   FROM produtos, compras
3   WHERE prod_codigo = comp_codproduto
4   GROUP BY prod_nome;
```

A execução desse conjunto de comandos por diversos usuários simultaneamente vai gerar um grande tráfego de informações pela rede, o que não ocorre se esse conjunto de comandos for definido usando um procedimento armazenado no próprio servidor do banco de dados e o usuário solicite apenas sua

execução e o retorno do resultado, usando a sintaxe descrita no quadro a seguir. Desse modo, o usuário solicita ao servidor que execute a operação e lhe devolva o resultado, não se preocupando com a estrutura do banco de dados.

```
1 | mysql> CALL nome_do_procedimento();
```

É possível também na criação dos procedimentos utilizar comandos comuns às linguagens de programação (que você já viu em disciplinas anteriores), como IF, WHILE, RETURN etc. Outra observação que deve ser feita é que visões não podem ser criadas em procedimentos armazenados.

## Criando, executando e apagando procedimentos armazenados

---

A instrução para criar um procedimento armazenado é simples, basta utilizar o comando CREATE PROCEDURE em conjunto com a lista de parâmetros (caso necessário) a serem usados. A sintaxe de criação de um procedimento armazenado é descrita no destaque a seguir.

```
1 | mysql> CREATE PROCEDURE nome_do_procedimento
2 |   ([parâmetros de entrada e/ou saída])
3 |   BEGIN
4 |     comandos em SQL
5 |   END;
```

Nessa expressão, no campo nome\_do\_procedimento, deve-se inserir o nome que se deseja atribuir para o procedimento, nome esse que deve seguir as mesmas regras usadas para os nomes das tabelas, conforme foi visto na **Aula 3**. Em seguida, caso haja, deve-se inserir a lista de parâmetros de entrada e/ou saída. Mesmo não havendo parâmetros, devem ser inseridos os parênteses. Finalmente, entre as palavras BEGIN e END, devem ser descritos os comandos SQL que definem a operação a ser executada.

A lista de parâmetros segue a notação [IN | OUT] nome\_do\_parâmetro tipo\_do\_dado, ou seja, deve-se usar a palavra-chave que identifica se o parâmetro é de entrada (IN) ou saída (OUT). Se nada for informado, será entendido que o

parâmetro é de entrada, o nome do parâmetro e o tipo que ele representa, conforme visto na **Aula 3**.

Vamos exercitar a criação de *Stored procedures* no banco de dados sistvendas para entendermos melhor o seu conceito? Na aula anterior sobre visões, um dos exemplos que foram trabalhados foi a consulta aos nomes de todos os produtos que foram vendidos. Visto que esse tipo de pesquisa tende a ser realizada diariamente num banco de dados de um sistema de vendas para gerenciamento de estoque, seria útil definir um procedimento armazenado para executar essa operação. O comando para criar essa *Stored procedure* é descrito no quadro a seguir.

```
1 mysql> CREATE PROCEDURE produtos_vendidos()
2   BEGIN
3     SELECT prod_nome
4     FROM produtos, compras
5     WHERE prod_codigo = comp_codproduto
6     GROUP BY prod_nome;
7   END
```

Observe que foi criado um procedimento armazenado denominado de `produtos_vendidos`, não sendo definido qualquer parâmetro, e os comandos que definem sua operação encontram-se entre as palavras-chave `BEGIN` e `END`. A execução do procedimento armazenado é feita usando o comando apresentado no quadro a seguir.

```
1 mysql> CALL produtos_vendidos();
```

Para analisarmos a aplicação dos procedimentos armazenados no banco de dados sistvendas, apresentam-se, na **Figura 1**, os dados presentes nas tabelas do banco de dados.

**Figura 01** - Tela do MySQL após os comandos `SELECT * FROM produtos`, `SELECT * FROM clientes` e `SELECT * FROM compras`.



```
mysql> SELECT * FROM produtos;
+-----+-----+-----+-----+
| prod_codigo | prod_nome                | prod_marca | prod_preco |
+-----+-----+-----+-----+
| 1           | Ventilador                | ARNO       | 90.00      |
| 2           | Celular N97               | NOKIA      | 1200.00    |
| 3           | Chocolate sonho de valsa  | Lacta      | 0.80       |
| 4           | Geladeira                 | Brastemp   | 2000.00    |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM clientes;
+-----+-----+-----+-----+-----+
| cli_codigo | cli_nome                | cli_CPF    | cli_sexo | cli_dataNascimento |
+-----+-----+-----+-----+-----+
| 1          | José Josemar           | 3252541    | M        | 1980-10-03 00:00:00 |
| 2          | Luciana                 | 4812072    | F        | 1972-04-30 00:00:00 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM compras;
+-----+-----+-----+-----+-----+
| comp_codigo | comp_codproduto | comp_codcliente | comp_total |
+-----+-----+-----+-----+-----+
| 1           | 3               | 2               | 30         |
| 2           | 2               | 2               | 1          |
| 3           | 4               | 1               | 1          |
| 4           | 3               | 1               | 100       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

**Fonte:** MySQL 5.7 Command Line Client

A resposta do sistema SGBD para os comandos anteriores é ilustrada na **Figura 2**. É interessante notar o uso do operador `DELIMITER` antes da criação do procedimento. Ele é usado para redefinir o caractere delimitador de comandos como `|`, isso é feito para que possamos usar o caractere `;` no meio do procedimento. Caso não efetuemos essa troca, o procedimento será enviado pela metade e uma mensagem de erro será enviada ao terminal, por problemas na sintaxe.

**Figura 02** - Tela do MySQL após os comandos CREATE PROCEDURE e CALL.

```
MySQL 5.7 Command Line Client
mysql> use sistvendas;
Database changed
mysql> DELIMITER |
mysql> CREATE PROCEDURE produtos_vendidos()
-> BEGIN
->   SELECT prod_nome FROM produtos, compras
->   WHERE prod_codigo = comp_codproduto
->   GROUP BY prod_nome;
-> END
-> |
Query OK, 0 rows affected (0.01 sec)

mysql> CALL produtos_vendidos(); |
+-----+
| prod_nome |
+-----+
| Celular N97 |
| Chocolate sonho de valsa |
| Geladeira |
+-----+
3 rows in set (0.01 sec)

Query OK, 0 rows affected (0.03 sec)

mysql>
```

**Fonte:** MySQL 5.7 Command Line Client



**Vídeo 01** - Introdução a *Stored Procedures*

Agora que aprendemos a criar procedimentos armazenados, que tal aprendermos a criá-los usando parâmetros? Para exemplificar essa situação, suponha que desejamos saber a quantidade total de produtos comprados por um determinado cliente no nosso banco de dados sistvendas. Para isso, criamos o procedimento descrito no destaque a seguir.

```

1 mysql> CREATE PROCEDURE comprastotal_cliente
2   (nome_cliente varchar(50), OUT quantidade_total int)
3   BEGIN
4     SELECT SUM(comp_total)
5     INTO quantidade_total
6     FROM clientes, compras
7     WHERE cli_nome=nome_cliente
8     AND comp_codcliente=cli_codigo;
9   END

```

Analisando cuidadosamente os comandos, podemos ver que o procedimento é definido com dois parâmetros, um de entrada denominado de nome\_cliente e outro de saída denominado de quantidade\_total. Perceba que não é necessário colocar a cláusula IN para definir o parâmetro como sendo de entrada, mas se deve usar a cláusula OUT na definição do parâmetro de saída. Observe também que após o nome dos parâmetros é definido seu tipo, como visto na **Aula 3**. Outra novidade mostrada na sequência de comandos é a estrutura SELECT ... INTO, cuja função é direcionar o resultado da consulta para uma variável definida após a cláusula INTO. No nosso caso, o resultado do SELECT é atribuído à variável quantidade\_total, que é um parâmetro de saída. A **Figura 3** ilustra a resposta do SGBD após a criação do procedimento.

**Figura 03** - Tela do MySQL após os comandos CREATE PROCEDURE, CALL e SELECT.

```

MySQL 5.7 Command Line Client
mysql> DELIMITER |
mysql> CREATE PROCEDURE comprastotal_cliente
-> (nome_cliente VARCHAR(50), OUT quantidade_total INT)
-> BEGIN
->   SELECT SUM(comp_total) INTO quantidade_total FROM clientes, compras
->   WHERE cli_nome = nome_cliente
->   AND comp_codcliente = cli_codigo;
-> END
-> |
Query OK, 0 rows affected (0.00 sec)

mysql> CALL comprastotal_cliente('Luciana',@Total); |
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @Total; |
+-----+
| @Total |
+-----+
|      32 |
+-----+
1 row in set (0.02 sec)

mysql>

```

**Fonte:** MySQL 5.7 Command Line Client

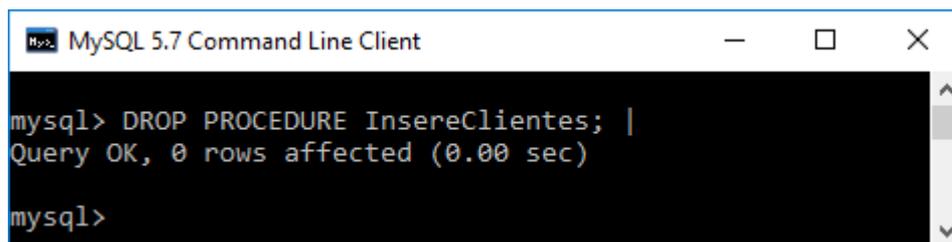
Também é ilustrada na **Figura 3** a execução do procedimento, consultando o total de produtos comprados pela cliente Luciana e armazenando o valor numa variável denominada Total. Ao se passar o nome de uma variável para um procedimento, é adicionado o caractere “@” antes do nome da variável. É interessante observar que durante o processamento interno do procedimento, como ocorre em qualquer linguagem de programação, a variável Total será associada ao parâmetro quantidade\_total.

Observe a utilização do comando SELECT para verificar o conteúdo da variável Total.

Para excluir um procedimento armazenado do banco de dados, você deve utilizar o comando DROP, o qual tem sua sintaxe descrita no quadro a seguir. E a resposta do SGBD a esse comando é ilustrada na **Figura 4**.

```
1 | mysql>DROP PROCEDURE nome_do_procedimento;
```

**Figura 04** - Tela do MySQL após o comando DROP PROCEDURE.



```
MySQL 5.7 Command Line Client
mysql> DROP PROCEDURE InsereClientes; |
Query OK, 0 rows affected (0.00 sec)
mysql>
```

Fonte: MySQL 5.7 Command Line Client



**Vídeo 02** - Parâmetros

## Usando estruturas de programação em procedimentos armazenados

---

Os procedimentos armazenados são estruturas extremamente poderosas que facilitam a interação dos usuários com o servidor de banco de dados, além de ajudar no controle de acesso aos dados, aumentando a segurança do sistema, como

mencionado anteriormente. Um dos requisitos que contribuem para essas características dos procedimentos está na possibilidade de uso de estruturas de controle de fluxo de dados, como estruturas de decisão e de repetição.

Você está familiarizado com algumas dessas estruturas, pois já foram apresentadas nas disciplinas de Lógica e Linguagem de Programação. As estruturas de controle de fluxo definidas no MySQL são: IF, CASE, LOOP, LEAVE, ITERATE, REPEAT e WHILE. Vamos discutir as duas mais usadas, o IF, que define estruturas de decisão, e o WHILE, que define estruturas de repetição. O destaque a seguir apresenta a estrutura do IF em MySQL.

```
1 IF condição 1 THEN comandos em SQL
2   [ELSEIF condição 2 THEN comandos em SQL] ...
3   [ELSE comandos em SQL]
4 END IF
```

A sintaxe pode ser explicada do seguinte modo: SE (IF) a condição 1 é satisfeita, um grupo de comandos em SQL é executado, SENÃO SE (ELSEIF) a condição 2 é satisfeita, outro grupo de comandos em SQL é executado e, finalmente, se nenhuma (SENÃO – ELSE) das condições anteriores for satisfeita, o último grupo de comando é executado. Lembrando que os termos entre chaves (“[” e “]”) são opcionais.

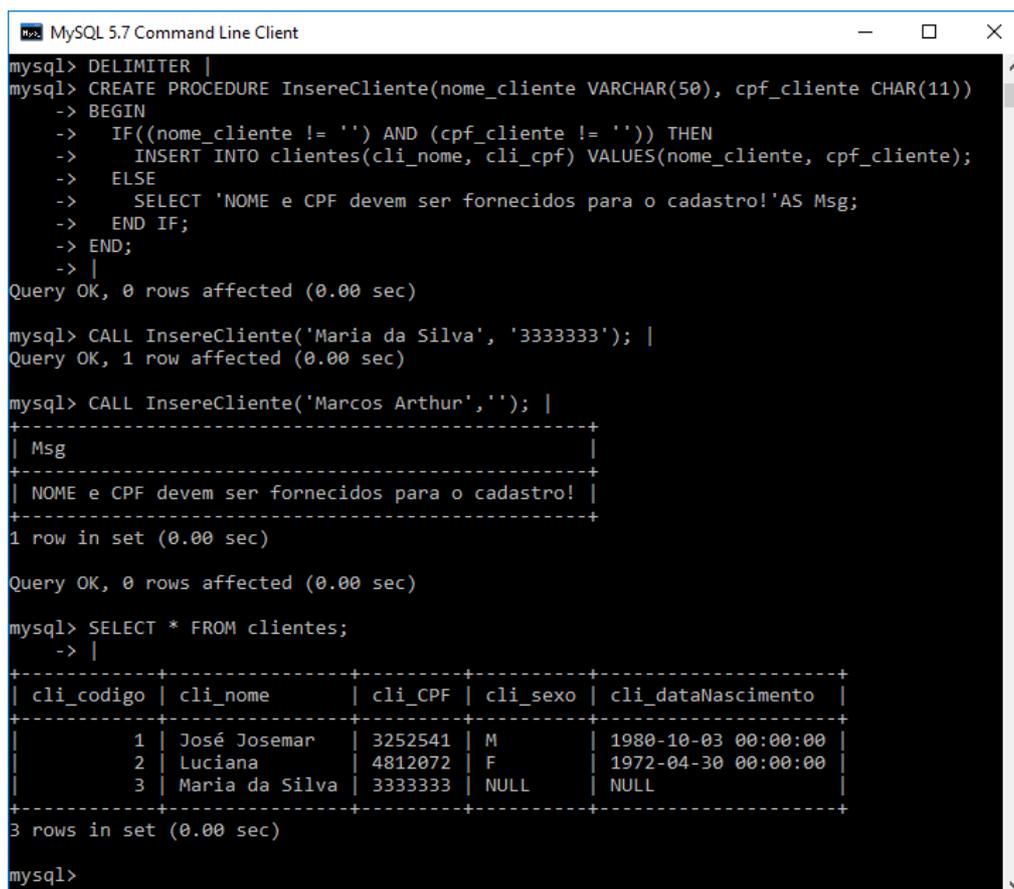
Para exemplificar a utilização da estrutura IF ... END IF num procedimento armazenado, imagine que precisamos definir um procedimento para inserção de clientes no nosso banco de dados sistvendas, mas para isso, devemos garantir que sejam informados o nome e o CPF do cliente. O destaque a seguir apresenta uma solução para esse problema. Analise com atenção a sequência de comandos apresentada.

```
1 mysql> CREATE PROCEDURE InsereCliente (nome_cliente varchar(50), cpf_cliente char(11))
2 BEGIN
3   IF ((nome_cliente != "") AND (cpf_cliente != "")) THEN
4     INSERT INTO clientes (cli_nome, cli_cpf) VALUES (nome_cliente, cpf_cliente);
5   ELSE
6     SELECT 'NOME e CPF devem ser fornecidos para o cadastro!' AS Msg;
7   END IF;
8 END;
```

Observe que no quadro anterior é definido um procedimento com dois parâmetros de entrada (nome\_cliente, cpf\_cliente) que serão inseridos na tabela **clientes** SE os respectivos parâmetros não forem nulos, SENÃO é apresentada na

tela uma mensagem de erro. A condição do IF é definida usando o operador "!=" que significa DIFERENTE DE e o operador lógico AND, ou seja, SE nome\_cliente DIFERENTE DE vazio E cpf\_cliente DIFERENTE DE vazio, a operação de inserção é executada. Observe também o uso do comando SELECT para impressão de mensagens na tela. A **Figura 5** apresenta a construção do procedimento descrito anteriormente no MySQL.

**Figura 05** - Tela do MySQL após o comando CREATE PROCEDURE com estrutura IF/ELSE, CALL e SELECT.



```
mysql> DELIMITER |
mysql> CREATE PROCEDURE InsereCliente(nome_cliente VARCHAR(50), cpf_cliente CHAR(11))
-> BEGIN
-> IF((nome_cliente != '') AND (cpf_cliente != '')) THEN
->   INSERT INTO clientes(cli_nome, cli_cpf) VALUES(nome_cliente, cpf_cliente);
-> ELSE
->   SELECT 'NOME e CPF devem ser fornecidos para o cadastro!' AS Msg;
-> END IF;
-> END;
-> |
Query OK, 0 rows affected (0.00 sec)

mysql> CALL InsereCliente('Maria da Silva', '3333333'); |
Query OK, 1 row affected (0.00 sec)

mysql> CALL InsereCliente('Marcos Arthur',''); |
+-----+
| Msg |
+-----+
| NOME e CPF devem ser fornecidos para o cadastro! |
+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM clientes;
-> |
+-----+-----+-----+-----+-----+
| cli_codigo | cli_nome | cli_CPF | cli_sexo | cli_dataNascimento |
+-----+-----+-----+-----+-----+
| 1 | José Josemar | 3252541 | M | 1980-10-03 00:00:00 |
| 2 | Luciana | 4812072 | F | 1972-04-30 00:00:00 |
| 3 | Maria da Silva | 3333333 | NULL | NULL |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Fonte: MySQL 5.7 Command Line Client

Além da criação do procedimento discutido no quadro anterior, a **Figura 5** traz um exemplo de uso do procedimento informando corretamente os dados de inserção, outro em que os dados não são informados de forma correta e os dados contidos na tabela clientes após a execução do procedimento. Lembre que, pela definição da tabela clientes, a coluna cli\_codigo é chave primária, de modo que, para não haver problemas na utilização do procedimento, é necessário que esta coluna seja do tipo **AUTO\_INCREMENT**, além disso a tabela clientes deverá permitir a

inserção de novos registros com os atributos cli\_sexo e cli\_dataNascimento com os valores NULL, pois a nossa procedure que criamos utiliza apenas o nome e CPF para inserir um novo cliente.



### Vídeo 03 - Estruturas de Controle

Agora que vimos a utilização das estruturas de decisão, vamos discutir um dos exemplos de estrutura de repetição. O quadro a seguir apresenta a estrutura do WHILE em MySQL. A sintaxe pode ser explicada do seguinte modo: ENQUANTO (WHILE) a condição é satisfeita EXECUTE (DO) um grupo de comandos em SQL.

```
1 WHILE condição DO
2   comandos em SQL
3 END WHILE
```

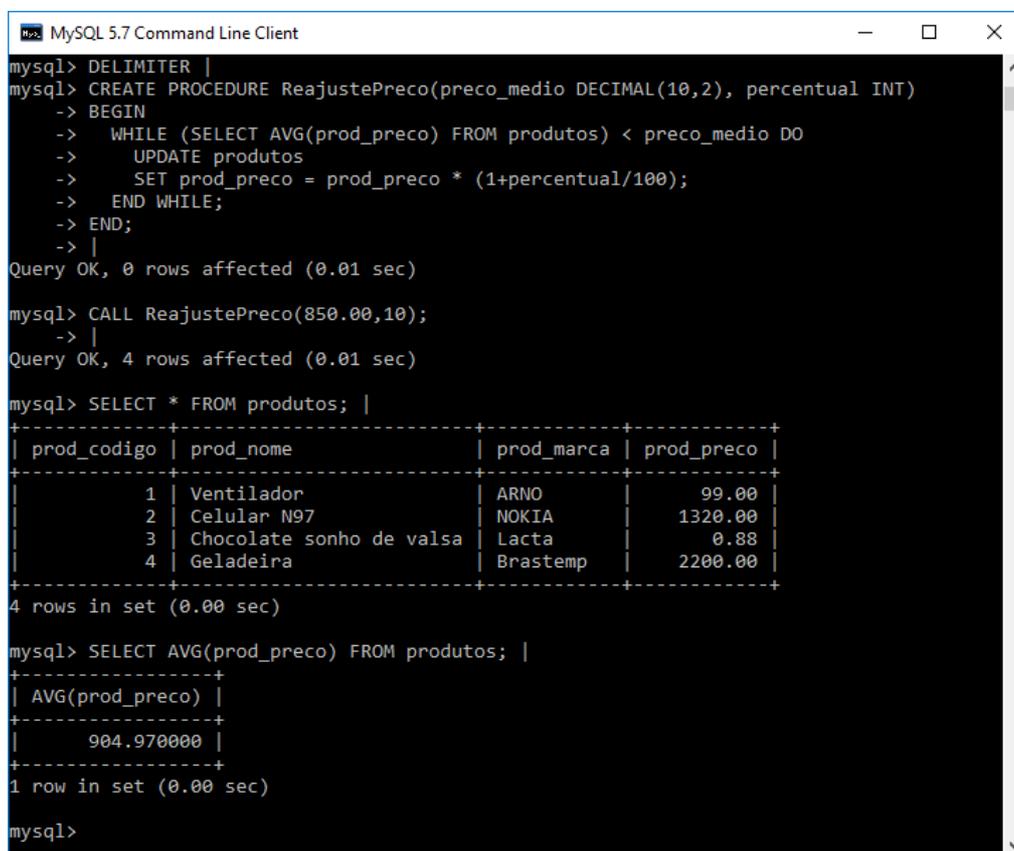
Para exemplificar a utilização da estrutura WHILE ... END WHILE num procedimento armazenado, imagine que precisamos definir um procedimento para reajustar os preços da tabela produtos do nosso banco de dados sistvendas, com a condição de que os preços de todos os produtos sejam reajustados de um percentual fixo, enquanto um determinado preço médio não seja atingido. O quadro a seguir apresenta uma solução para esse problema. Analise com atenção a sequência de comandos apresentada.

```
1 mysql> CREATE PROCEDURE ReajustePreco (preco_medio decimal(10,2), percentual int)
2   BEGIN
3     WHILE (SELECT AVG(prod_preco) FROM produtos) < preco_medio DO
4       UPDATE produtos
5         SET prod_preco = prod_preco*(1+percentual/100);
6     END WHILE;
7   END;
```

Observe que no quadro anterior é definido um procedimento com dois parâmetros de entrada (preco\_medio, percentual), que são usados dentro de uma estrutura de repetição (WHILE) de modo que, ENQUANTO a média dos preços dos

produtos for menor que preco\_medio, todos os produtos têm seu valor atualizado de um determinado percentual. A **Figura 6** apresenta a construção do procedimento descrito anteriormente no MySQL.

**Figura 06** - Tela do MySQL após o comando CREATE PROCEDURE com estrutura WHILE, CALL e SELECT.



```
mysql> DELIMITER |
mysql> CREATE PROCEDURE ReajustePreco(preco_medio DECIMAL(10,2), percentual INT)
-> BEGIN
-> WHILE (SELECT AVG(prod_preco) FROM produtos) < preco_medio DO
-> UPDATE produtos
-> SET prod_preco = prod_preco * (1+percentual/100);
-> END WHILE;
-> END;
-> |
Query OK, 0 rows affected (0.01 sec)

mysql> CALL ReajustePreco(850.00,10);
-> |
Query OK, 4 rows affected (0.01 sec)

mysql> SELECT * FROM produtos; |
+-----+-----+-----+-----+
| prod_codigo | prod_nome           | prod_marca | prod_preco |
+-----+-----+-----+-----+
|          1 | Ventilador          | ARNO       |      99.00 |
|          2 | Celular N97         | NOKIA      |     1320.00|
|          3 | Chocolate sonho de valsa | Lacta     |         0.88|
|          4 | Geladeira           | Brastemp   |     2200.00|
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT AVG(prod_preco) FROM produtos; |
+-----+
| AVG(prod_preco) |
+-----+
|      904.970000 |
+-----+
1 row in set (0.00 sec)

mysql>
```

**Fonte:** MySQL 5.7 Command Line Client

Além da criação do procedimento discutido no destaque anterior, a **Figura 6** traz um exemplo de uso do procedimento informando como parâmetros de entrada, preco\_medio igual a 850.00 e percentual igual a 10; a tabela de produtos em que se pode constatar o aumento dos preços quando comparada com tabela similar mostrada na **Figura 1** e o uso do SELECT para calcular o novo preço médio depois da aplicação do aumento.

Vamos treinar um pouco o uso das estruturas de decisão e repetição?

# Atividade 01

---

1. O que são procedimentos armazenados (*Stored Procedures*)?
2. Apresente duas vantagens em se utilizar esse tipo de recurso em um sistema de banco de dados.

Vamos praticar um pouco para que você se familiarize com os comandos apresentados.

Acesse o banco de dados **sispagamentos (Aula 07)** e elabore o que se pede.

1. Crie um procedimento armazenado para obter a lista dos empregados que não pagam imposto de renda (IR).
2. Crie um procedimento armazenado com passagem de parâmetros para cadastro de funcionários, inserindo seus dados na tabela empregados.
3. Crie um procedimento armazenado que retorne por uma variável de saída o número total de empregados que não tem desconto de imposto de renda.
4. Exclua do banco de dados os procedimentos anteriores.
5. Crie um procedimento armazenado com passagem de parâmetros para cadastro de funcionários, inserindo seus dados na tabela empregados.
6. Crie um procedimento armazenado com passagem de parâmetros para cadastro de funcionários, atualizando seus dados na tabela empregados.
7. Crie um procedimento armazenado com passagem de parâmetros para cadastro de funcionários, apagando seus dados na tabela empregados. Exclua do banco de dados os procedimentos anteriores.
8. Nos três casos, certifique-se de que os valores estão sendo informados de forma correta, caso contrário, imprima uma mensagem de erro.

# Funções

---

Assim como ocorre com os procedimentos, é possível ter uma sequência de comandos SQL encapsulados em estruturas denominadas **funções**. Como você viu em disciplinas anteriores, que trataram de lógica e programação, a principal diferença entre uma função e um procedimento está no fato de que a função obrigatoriamente deve retornar um valor. Nesta disciplina, já trabalhamos, em aulas anteriores, com funções internas, pré-definidas pelo próprio SGBD, como AVG(), SUM(), COUNT() etc. Mas o usuário pode definir suas próprias funções com parâmetros de entrada e variáveis locais. É possível no SQL construir dois tipos de funções:

- **Funções escalares:** estruturas semelhantes a funções internas, que retornam um único valor.
- **Funções com valor de tabela:** estruturas semelhantes a visões com a diferença de aceitarem parâmetros de entrada, que retornam uma tabela como resultado do seu processamento.

No caso do *MySQL*, não é permitido que uma função retorne uma tabela. Desse modo, vamos estudar apenas como criar e utilizar as **funções escalares**.

## Funções escalares

---

Uma função escalar retorna um único valor de dados de um tipo predefinido e pode ser utilizada do mesmo modo que uma função interna, sendo mais usada como:

- Uma expressão na lista de um comando SELECT.
- Uma expressão em uma cláusula WHERE ou HAVING.
- Uma expressão em uma cláusula ORDER BY ou GROUP BY.
- Uma expressão na cláusula SET de um comando UPDATE.
- Uma expressão na cláusula VALUES de um comando INSERT.

A instrução para criar uma função é simples, basta utilizar o comando CREATE FUNCTION em conjunto com a lista de parâmetros (caso necessário) a serem usados. A sintaxe de criação de uma função é descrita a seguir.

```
1 mysql> CREATE FUNCTION nome_da_funcao (parâmetros de entrada)
2 RETURNS tipo_de_retorno
3 BEGIN
4     comandos em SQL
5     RETURN valor_de_retorno;
6 END;
```

Nessa expressão, no campo *nome\_da\_funcao* deve-se inserir o nome que se deseja atribuir para a função. Esse nome deve seguir as mesmas regras usadas para os nomes das tabelas, que foi visto na **Aula 03**. Em seguida, caso haja parâmetros, deve-se fornecer a lista de parâmetros de entrada, que segue o mesmo formato definido na aula anterior sobre **procedimentos armazenados**. Mesmo não havendo parâmetros, devem ser inseridos os parênteses () na criação da função. A cláusula RETURNS define o tipo de dado que será retornado pela função, por exemplo, um número inteiro, um conjunto de caracteres etc. Finalmente, entre as palavras BEGIN e END devem ser descritos os comandos SQL que definem a operação a ser executada. Lembrando que antes do END deve ser usada a cláusula RETURN que define a variável ou expressão que irá retornar o resultado da função.

Para termos o retorno da função por meio de uma variável, é necessário sabermos como se faz a declaração de variáveis locais no MySQL. Isso é feito usando o comando DECLARE, cuja sintaxe é apresentada a seguir.

```
1 mysql> DECLARE nome_da_variavel tipo_de_dado;
```

Nessa expressão, no campo *nome\_da\_variavel* deve-se inserir o nome da variável que está sendo criada e esse nome deve seguir as mesmas regras usadas para os nomes das tabelas, que foi visto na **Aula 03**. O mesmo acontece para a definição do tipo de dado que será atribuído à variável.

Agora, vamos exercitar a criação de funções no banco de dados **sistvendas** para entendermos melhor o seu conceito? Se você quiser relembrar a definição desse banco de dados. Para começar, vamos criar uma função que retorne a quantidade total de produtos vendidos. O comando para criar essa função é descrito no destaque a seguir.

```
1 mysql> CREATE FUNCTION Total_Vendas()
2 RETURNS int
3 BEGIN
4 DECLARE qtde_vendas int;
5 SELECT sum(comp_total) INTO qtde_vendas
6 FROM compras;
7 RETURN qtde_vendas;
8 END
```

Observe que foi criada uma função denominada *Total\_Vendas*, sem qualquer parâmetro, que retorna um valor do tipo inteiro. Os comandos que definem sua operação encontram-se entre as palavras chave *BEGIN* e *END*.

Analisando cuidadosamente os comandos, vemos o uso do comando *DECLARE* para definir uma variável intitulada *qtde\_vendas* do tipo inteiro. Em seguida, é feita uma consulta (*SELECT*) da soma de todos os produtos vendidos da tabela *compras*. Esse valor é armazenado na variável *qtde\_vendas* usando a cláusula *INTO*. Finalmente, a cláusula *RETURN* define que a função retorna o valor contido na variável *qtde\_vendas*.

Agora, a partir dessa função, vamos criar outra que retorne a quantidade total de unidades vendidas de um produto dado o seu código. O comando para criar essa função é descrito a seguir.

```

1 mysql> CREATE FUNCTION Total_Vendas2 (codigo_produto int)
2 RETURNS int
3 BEGIN
4 DECLARE qtde_vendas int;
5 SELECT sum(comp_total) INTO qtde_vendas
6 FROM compras
7 WHERE comp_codproduto = codigo_produto;
8 RETURN qtde_vendas;
9 END

```

Compare a diferença entre as duas funções apresentadas. Observe que a função denominada Total\_Vendas2 contém um parâmetro de entrada do tipo inteiro. E que é feita uma consulta (SELECT) da soma de todos os produtos da tabela **compras** cujo código equivale ao parâmetro de entrada codigo\_produto.

Para analisarmos a aplicação das funções no banco de dados **sistvendas**, apresentam-se, na **Figura 7**, os dados presentes nas tabelas do banco de dados. A **Figura 8** ilustra a resposta do SGBD após a criação da função.

**Figura 07** - Tela do MySQL após os comandos SELECT \* FROM **produtos**, SELECT \* FROM **clientes** e SELECT \* FROM **compras**.

```

mysql> SELECT * FROM produtos;
+-----+-----+-----+-----+
| prod_codigo | prod_nome           | prod_marca | prod_preco |
+-----+-----+-----+-----+
| 1           | Ventilador          | ARNO       | 90.00      |
| 2           | Celular N97         | NOKIA      | 1200.00    |
| 3           | Chocolate sonho de valsa | Lacta      | 0.80       |
| 4           | Geladeira           | Brastemp   | 2000.00    |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM clientes;
+-----+-----+-----+-----+-----+
| cli_codigo | cli_nome           | cli_CPF | cli_sexo | cli_dataNascimento |
+-----+-----+-----+-----+-----+
| 1           | José Josemar       | 3252541 | M        | 1988-10-03 00:00:00 |
| 2           | Luciana            | 4812072 | F        | 1972-04-30 00:00:00 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM compras;
+-----+-----+-----+-----+
| comp_codigo | comp_codproduto | comp_codcliente | comp_total |
+-----+-----+-----+-----+
| 1           | 3               | 2               | 30         |
| 2           | 2               | 2               | 1          |
| 3           | 4               | 1               | 1          |
| 4           | 3               | 1               | 100        |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>

```

Fonte: MySQL 5.7 Command Line Client

**Figura 08** - Tela do MySQL após os comandos CREATE FUNCTION.

```
MySQL 5.7 Command Line Client
mysql> DELIMITER |
mysql> CREATE FUNCTION Total_Vendas()
-> RETURNS INT
-> BEGIN
->   DECLARE qtde_vendas INT;
->   SELECT SUM(comp_total) INTO qtde_vendas FROM compras;
->   RETURN qtde_vendas;
-> END
-> |
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE FUNCTION Total_Vendas2(codigo_produto INT)
-> RETURNS INT
-> BEGIN
->   DECLARE qtde_vendas INT;
->   SELECT SUM(comp_total) INTO qtde_vendas FROM compras
->   WHERE comp_codproduto = codigo_produto;
->   RETURN qtde_vendas;
-> END
-> |
Query OK, 0 rows affected (0.00 sec)

mysql>
```

**Fonte:** MySQL 5.7 Command Line Client

Note que está sendo usado o caractere “|” como delimitador de comandos, do mesmo modo como no procedimento, isso é feito para que possamos usar o caractere “;” no meio da função.



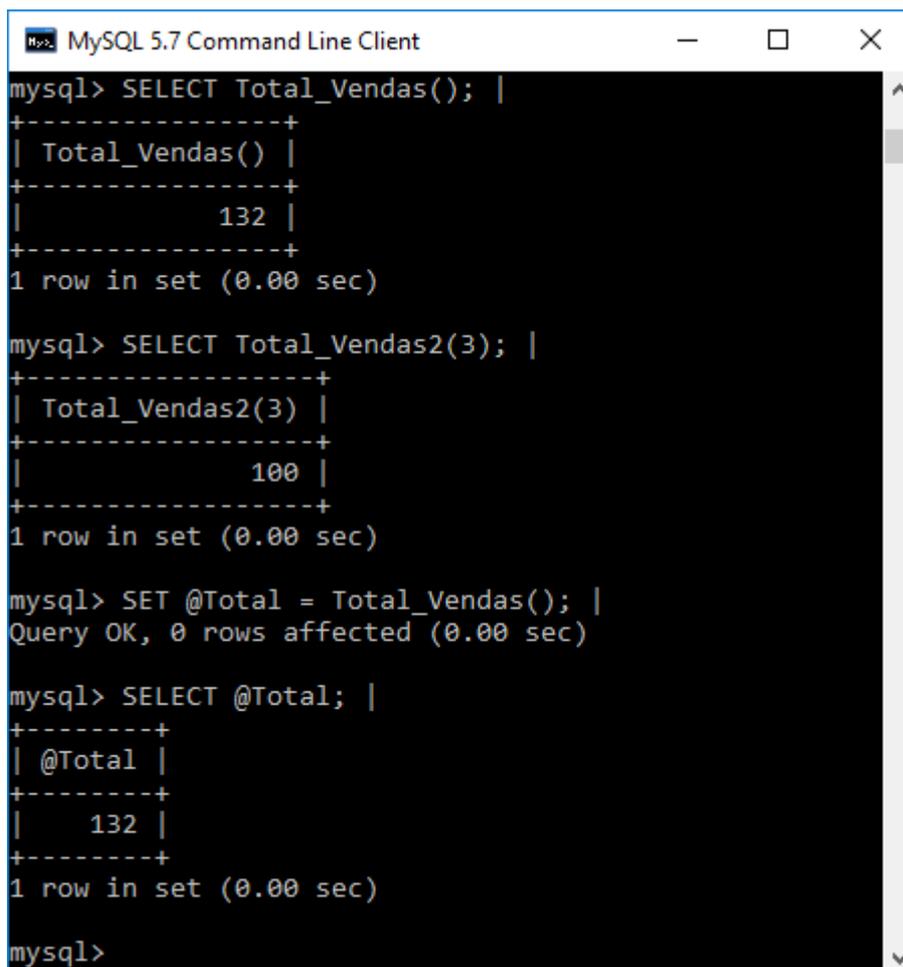
**Vídeo 04** - Introdução a Funções

## Utilizando funções escalares

Agora que já sabemos criar funções, vamos exemplificar duas formas de executá-las, imprimindo seu resultado na tela. Observe os comandos ilustrados no destaque a seguir e, na **Figura 9**, a resposta do SGBD à sua execução.

```
1 mysql> SELECT Total_Vendas();
2 mysql> SELECT Total_Vendas2(3);
3 mysql> SET @Total = Total_Vendas();
4 mysql> SELECT @Total;
```

**Figura 09** - Tela do MySQL após diversos comandos SELECT e SET.



```
mysql> SELECT Total_Vendas(); |
+-----+
| Total_Vendas() |
+-----+
|          132 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT Total_Vendas2(3); |
+-----+
| Total_Vendas2(3) |
+-----+
|          100 |
+-----+
1 row in set (0.00 sec)

mysql> SET @Total = Total_Vendas(); |
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @Total; |
+-----+
| @Total |
+-----+
|      132 |
+-----+
1 row in set (0.00 sec)

mysql>
```

**Fonte:** MySQL 5.7 Command Line Client

Os dois primeiros exemplos utilizam o comando SELECT para imprimir na tela o resultado das funções Total\_Vendas() e Total\_Vendas2(), sendo passado nesta última o parâmetro 3, ou seja, se deseja saber o número de itens vendidos do produto de código igual a 3. Resultado similar pode ser obtido utilizando o comando SET para atribuir o resultado de uma função a uma variável e em seguida imprimindo o mesmo na tela usando SELECT. Observe que nesse caso o SGBD reconhece que uma variável está sendo definida pela inserção do caractere “@” antes do seu nome e que não é necessário associar um tipo específico a ela, sendo uma forma alternativa de se definir uma variável.

Embora já saibamos como criar uma função e obter o resultado da sua execução, ainda não mostramos como ela pode ser utilizada na manipulação de dados num SGBD. Para isso, vamos exemplificar a seguir, dois casos de uso da função Total\_Vendas2() criada anteriormente.

```

1 mysql> SELECT prod_codigo AS Codigo, prod_nome AS Nome,
2   Total_Vendas2(prod_codigo) AS Vendas
3   FROM produtos
4   ORDER BY prod_codigo;
5
6 mysql> SELECT prod_codigo AS Codigo, prod_nome AS Nome,
7   Total_Vendas2(prod_codigo) AS Vendas
8   FROM produtos
9   WHERE Total_Vendas2(prod_codigo) >= 15
10  ORDER BY prod_codigo;

```

No primeiro exemplo, a função `Total_Vendas2()` é usada numa consulta para gerar as linhas de uma coluna denominada `Vendas`, mostrando assim o total de vendas de cada produto associado a tabela **produtos**. No segundo exemplo, temos um caso similar, porém, a função também é utilizada na cláusula `WHERE` para filtrar os produtos com venda superior ou igual a 15 unidades. A **Figura 10** apresenta o resultado das consultas anteriores ao serem processadas no *MySQL*.

**Figura 10** - Tela do MySQL após diversos comandos `SELECT` utilizando a função `Total_Vendas2()`.

```

MySQL 5.7 Command Line Client
mysql> SELECT prod_codigo AS Codigo, prod_nome AS Nome, Total_Vendas2(prod_codigo) AS Vendas
-> FROM produtos
-> ORDER BY prod_codigo;
+-----+-----+-----+
| Codigo | Nome                | Vendas |
+-----+-----+-----+
| 1      | Ventilador          | NULL   |
| 2      | Celular N97         | 31     |
| 3      | Chocolate sonho de valsa | 100    |
| 4      | Geladeira           | 1      |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT prod_codigo AS Codigo, prod_nome AS Nome, Total_Vendas2(prod_codigo) AS Vendas
-> FROM produtos
-> WHERE Total_Vendas2(prod_codigo) >= 15
-> ORDER BY prod_codigo;
+-----+-----+-----+
| Codigo | Nome                | Vendas |
+-----+-----+-----+
| 2      | Celular N97         | 31     |
| 3      | Chocolate sonho de valsa | 100    |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

**Fonte:** MySQL 5.7 Command Line Client

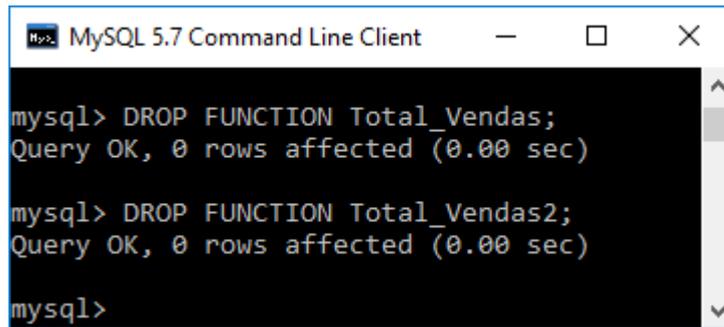
Finalmente, para excluir uma função do banco de dados, você deve utilizar o comando `DROP FUNCTION`, o qual tem sua sintaxe descrita no destaque a seguir. E a resposta do SGBD a esse comando aplicado nas funções criadas anteriormente é ilustrada na **Figura 11**.

```

1 mysql> DROP FUNCTION nome_da_funcao;

```

**Figura 11** - Tela do MySQL após os comandos DROP FUNCTION.



```
mysql> DROP FUNCTION Total_Vendas;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> DROP FUNCTION Total_Vendas2;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql>
```

**Fonte:** MySQL 5.7 Command Line Client



**Vídeo 05** - Funções: Estrutura de Controle Condicional

## Atividade 02

---

1. O que são funções?
2. Qual a diferença entre funções e procedimentos?
3. Quais os tipos de funções que podem ser definidas em SQL?
4. Acesse o banco de dados sispagamentos, visto na Aula 07, e elabore o que se pede.
  - a. Crie uma função que retorne o desconto de INSS total dos empregados.
  - b. Crie uma função que retorne o total de descontos aplicados no salário de um determinado empregado.
  - c. Crie uma função que retorne a quantidade total de empregados de um determinado sexo.
  - d. Execute as funções e imprima seu resultado na tela.
  - e. Utilize as funções para obter uma relação dos dados dos empregados que têm um desconto de INSS maior que R\$ 50,00.

f. Utilize as funções para obter uma relação dos dados dos empregados que têm um salário líquido (salário bruto menos descontos) menor que R\$ 1000,00.

g. Exclua do banco de dados as funções anteriores.

## Estrutura de Controle e Repetição com Funções

---



**Vídeo 06** - Funções: Estrutura de Controle de Repetição.

## Conclusão

---

Encerramos por aqui nossa aula sobre procedimentos armazenados e funções na linguagem SQL. Na próxima aula, aprenderemos os conceitos de segurança de sistemas e segurança de banco de dados e como podemos controlar o acesso aos nossos dados.

Faça a autoavaliação com atenção e veja se precisa parar e refletir mais sobre a criação e o uso dos procedimentos armazenados e as funções, principalmente como as estruturas de controle de fluxo dados, que podem ser úteis em diferentes aplicações

Escreva no seu caderno todos os comandos SQL (e respectivas funções e *stored procedures*) aprendidos nesta aula para não esquecer.

Faça a autoavaliação com atenção e veja se precisa parar e refletir mais sobre a criação e o uso das *stored procedures* e funções

Bons estudos e boa sorte!

# Resumo

---

Nesta aula, você estudou a criação e utilização dos procedimentos armazenados e funções. Aprendeu a usar os comandos CREATE PROCEDURE, CALL e DROP PROCEDURE para criar, executar e apagar um procedimento e também os comandos CREATE FUNCTION e DROP FUNCTION para criar e apagar uma função. Viu também como utilizar o recurso de lista de parâmetros para definir parâmetros de entrada e saída para os procedimentos e funções. Por fim, você observou o uso dos procedimentos ou funções em conjunto com as estruturas de controle de fluxo de dados, IF ... END IF e WHILE ... END WHILE.

Além disso conhecemos o comando DECLARE para declaração de variáveis e vimos que também é possível declarar e inicializar uma variável usando o comando SET e o caractere "@". Além disso, vimos o uso das funções no processamento de consultas.

## Autoavaliação

---

1. Execute as operações a seguir, considerando o banco de dados sispagamentos.
  - a. Crie uma *Stored Procedure* que mostre para cada empregado a cidade em que ele mora.
  - b. Crie uma *Stored Procedure* para exibir quantos empregados moram em uma determinada cidade. Passe como parâmetro de entrada o nome da cidade.
  - c. Crie uma *Stored Procedure* que, passando como parâmetro o código do empregado, mostre o nome dele e seu salário bruto e líquido (após os descontos).
  - d. Crie uma *Stored Procedure* que tenha como entrada o código do empregado e mostre como parâmetros de saída o seu nome e cidade onde reside.

2. Execute as operações a seguir, considerando o banco de dados **sisvendas**:

- a. Crie uma *Stored Procedure* que tenha como parâmetro de entrada o código da compra e mostre como parâmetro de saída a quantidade de itens total nessa compra.
- b. Crie uma *Stored Procedure* para informar a classificação de cada cliente. Caso a soma do preço dos produtos comprados pelo cliente seja maior que R\$ 1000, sua categoria é cliente VIP; caso seja entre R\$ 500 e 1000, sua categoria é cliente normal; caso seja menor que R\$ 500, cliente popular. O código do cliente deve ser passado como parâmetro.

Consulte o manual de referência do MySQL (disponível em <https://dev.mysql.com/doc/refman/5.7/en/flow-control-statements.html>) e analise as estruturas de controle de fluxo de dados CASE e REPEAT.

3. Considere o banco de dados **CursoX**, criado na autoavaliação da Aula 03, cuja estrutura de tabelas é apresentada a seguir.

| ATRIBUTO              | TIPO           | DESCRIÇÃO         |
|-----------------------|----------------|-------------------|
| <b>aluno_cod</b>      | Número inteiro | Código do aluno   |
| <b>aluno_nome</b>     | Alfanumérico   | Nome do aluno     |
| <b>aluno_endereco</b> | Alfanumérico   | Endereço do aluno |
| <b>aluno_cidade</b>   | Alfanumérico   | Cidade do aluno   |

**Tabela:** Alunos

| ATRIBUTO        | TIPO           | DESCRIÇÃO            |
|-----------------|----------------|----------------------|
| <b>dis_cod</b>  | Número inteiro | Código da disciplina |
| <b>dis_nome</b> | Alfanumérico   | Nome da disciplina   |

| ATRIBUTO             | TIPO           | DESCRIÇÃO                   |
|----------------------|----------------|-----------------------------|
| <b>dis_carga</b>     | Número inteiro | Carga horária da disciplina |
| <b>dis_professor</b> | Alfanumérico   | Professor da disciplina     |

**Tabela:** Disciplina

| ATRIBUTO             | TIPO           | DESCRIÇÃO             |
|----------------------|----------------|-----------------------|
| <b>prof_cod</b>      | Número inteiro | Código do professor   |
| <b>prof_nome</b>     | Alfanumérico   | Nome do professor     |
| <b>prof_endereco</b> | Alfanumérico   | Endereço do professor |
| <b>prof_cidade</b>   | Alfanumérico   | Cidade do professor   |

**Tabela:** Professores

- Crie uma função que calcule a quantidade de professores e alunos que moram em uma determinada cidade.
- Crie uma função que calcule a carga horária média de um determinado professor. Elabore uma consulta usando a função criada.
- Crie uma função que receba o código do professor como parâmetro de entrada e retorne a cidade em que ele mora. Depois, elabore uma consulta para listar a quantidade de professores por cidade em que residem.
- Crie uma função que retorne a disciplina de maior carga horária de um professor. Depois, use essa função para gerar uma tabela com o nome do professor e nome da disciplina de maior carga horária dele.

## Referências

---

BEIGHLEY, L. Use a cabeça SQL. Rio de Janeiro: Editora AltaBooks, 2008.

MYSQL 5.7 Reference Manual. Disponível em: <<http://dev.mysql.com/doc/refman/5.7/en/>>. Acesso em: 28 jan. 2017.

WIKIPÉDIA. SQL. Disponível em: <<http://pt.wikipedia.org/wiki/SQL>>. Acesso em: 26 set. 2012.