



Cursos


➔ Listagem de disciplinas

Selecione uma disciplina

Aulas

- 01 Introdu  o a Banco de Dados
- 02 Modelo de Entidade e Relacionamento
- 03 Modelo Relacional
- 04 Transforma  es ER para MR
- 05 Transforma  es ER para MR e dicion rio de dados
- 06 Normaliza  o b sica
- 07 Normaliza  o avan ada
- 08 Introdu  o   Linguagem SQL e Sistemas Gerenciadores de Banco de Dados
- 09 Linguagem SQL - cria  o, inser  o e modifica  o de tabelas
- 10 Linguagem SQL - Consulta simples de tabelas
- 11 Linguagem SQL - Consulta avan ada de tabelas
- 12 Linguagem SQL - Altera  o da estrutura de tabelas e ambientes de m ltiplas tabelas
- 13 Linguagem SQL - Subconsultas
- 14 Linguagem SQL - VIS  ES
- 15 Linguagem SQL - STORED PROCEDURES
- 16 Linguagem SQL - Fun  es
- 17 Linguagem SQL - Seguran a
- 18 Engenharia Reversa
- 19 Utilizando SQL em Java
- 20 Utilizando conceitos avan ados de SQL em Java

⬅ Voltar 🖨 Imprimir ⬆ Topo



Sistemas de Banco de Dados

Aula 16 – Linguagem SQL – Fun  es

Professores autores
Jos  Josemar de Oliveira J nior (josemar@ect.ufrn.br)
Luciana Ribeiro Veloso (luciana.veloso@globo.com)



Apresenta  o

Na aula anterior, discutimos como criar, executar e apagar procedimentos armazenados (*stored procedures*), bem como utiliz -los em conjunto com estruturas de controle de fluxo de dados.

Nesta aula, estudaremos o conceito de **fun  es**, usadas para encapsular opera  es  teis para manipula  o o acesso de dados em um sistema de banco de dados. Aprenderemos a criar, executar e apagar essas estruturas, bem como utiliz -las no processamento de consultas. Tamb m ser  discutido como se pode declarar vari veis no SQL.



Objetivo

Ao final desta aula, voc  ser  capaz de:

- criar **fun  es** com e sem par metros no sistema *MySQL*;
- declarar vari veis locais que podem ser utilizadas dentro das fun  es;
- utilizar as fun  es para aux lio no processamento de consultas.

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 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 discutir apenas como criar e utilizar as funções escalares.

Funções escalares

Uma função escalar retorna um único valor de dados de um tipo pré-definido 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 no quadro abaixo.

```
mysql> CREATE FUNCTION nome_da_funcao (parâmetros de entrada)
      RETURNS tipo_de_retorno
      BEGIN
        comandos em SQL
        RETURN valor_de_retorno
      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 9. Em seguida, caso haja, 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 no quadro a seguir.

```
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 9. O mesmo acontece para a definição do tipo de dado que será atribuído à variável.

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, consulte a Aula 13. Para começar, vamos criar uma função que retorne a quantidade total de produtos vendidos. O comando para criar essa **função** é descrito no destaque abaixo.

```
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
```

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.

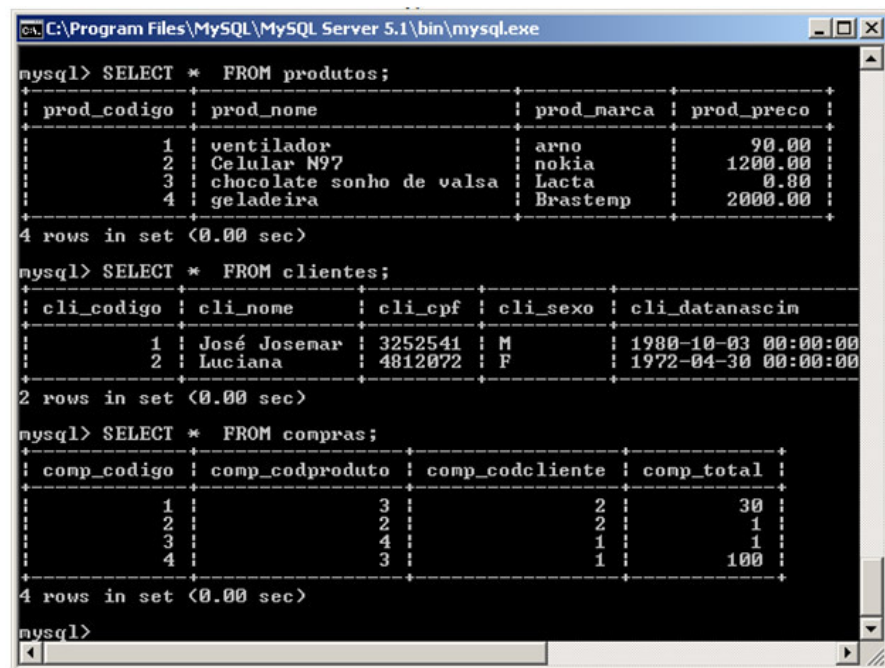
Analisando cuidadosamente os comandos, vemos o uso do comando DECLARE para definir uma variável intitulada qtde_vendas do tipo inteiro. Em seguida, é feita uma consulta (SELECT) da soma de todos os produtos vendidos da tabela **compras**. Esse valor é armazenado na variável qtde_vendas usando a cláusula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

Agora, a partir dessa função, vamos criar uma outra que retorne a quantidade total de unidades vendidas de um produto dado o seu código. O comando para criar essa **função** é descrito no quadro abaixo.

```
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
```

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 vendidos 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 1, os dados presentes nas tabelas do banco de dados. A Figura 2 ilustra a resposta do SGBD após a criação da função.



```
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_datanascim
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;
```

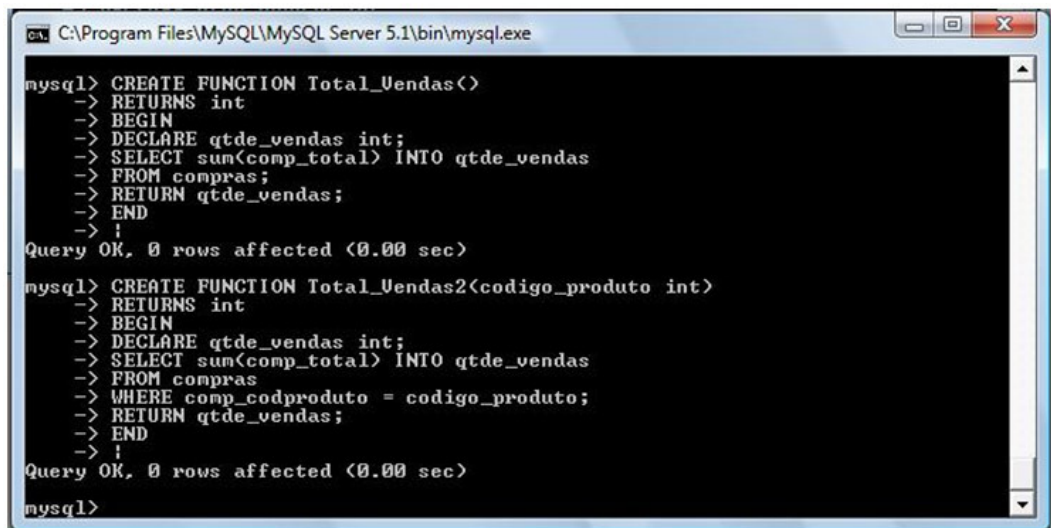
comp_codigo	comp_codproduto	comp_codcliente	comp_total
1	3	2	30
2	2	2	1
3	4	1	1
4	3	1	100

```
4 rows in set (0.00 sec)
```

```
mysql>
```

Figura 1 – Tela do MySQL após os comandos SELECT * FROM **produtos**, SELECT * FROM **clientes** e SELECT * FROM **compras**

Fonte: MySQL Server 5.1



```

C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql.exe

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)

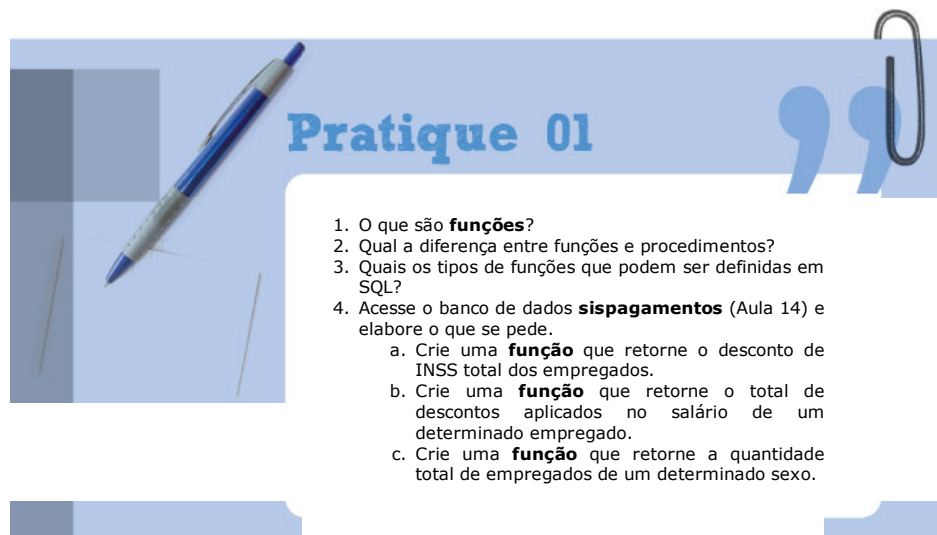
mysql> CREATE FUNCTION Total_Vendas2(codigo_produto int)
-> RETURNS int
-> BEGIN
-> DECLARE qtde_vendas int;
-> SELECT sum(comp_total) INTO qtde_vendas
-> FROM compras
-> WHERE comp_codproduto = codigo_produto;
-> RETURN qtde_vendas;
-> END
-> ;
Query OK, 0 rows affected (0.00 sec)

mysql>

```

Figura 2 – Tela do MySQL após os comandos CREATE FUNCTION. Fonte: MySQL Server 5.1

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.



Pratique 01

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** (Aula 14) 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.

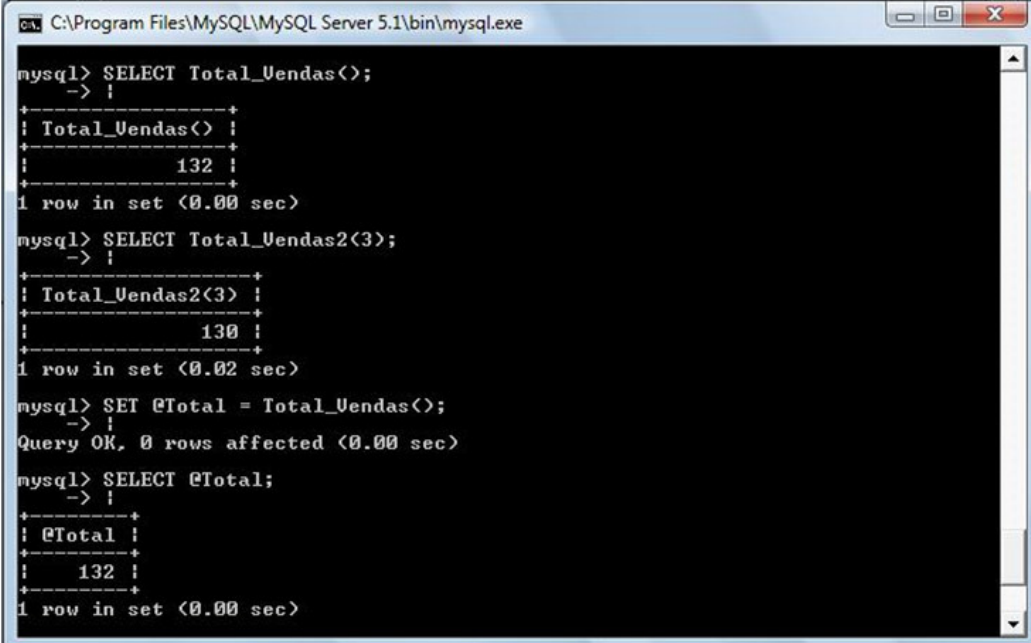
Utilizando funções escalares

Agora que já sabemos criar funções, vamos exemplificar duas formas de executá-las, imprimindo seu resultado na tela. Observe os comandos ilustrados no destaque abaixo e, na Figura 3, a resposta do SGBD à sua execução.

```

mysql> SELECT Total_Vendas();
mysql> SELECT Total_Vendas(3);
mysql> SET @Total = Total_Vendas();
mysql> SELECT @Total;

```



```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe

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

mysql> SELECT Total_Vendas2(3);
-> |
+-----+
| Total_Vendas2(3) |
+-----+
|          130    |
+-----+
1 row in set (0.02 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)

```

Figura 3 – Tela do MySQL após diversos comandos SELECT e SET. Fonte: MySQL Server 5.1

Os dois primeiros exemplos utilizam o comando SELECT para imprimir na tela o resultado das funções Total_Vendas() e Total_Vendas2(), sendo passado nessa 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 no quadro abaixo, dois casos de uso da função Total_Vendas2() criada anteriormente.

```

mysql> SELECT prod_codigo ASCodigo, prod_nome AS Nome,
Total_Vendas2(prod_codigo) AS Vendas
FROM produtos
ORDER BY prod_codigo;

```

```

mysql> SELECT prod_codigo ASCodigo, prod_nome AS Nome,
Total_Vendas2(prod_codigo) AS Vendas
FROM produtos
WHERE Total_Vendas2(prod_codigo) >= 15
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 a 15 unidades. A Figura 4 apresenta o resultado das consultas anteriores ao serem processadas no MySQL.

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe

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        | 1       |
| 3      | Chocolate sonho de | 130     |
| 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 |
+-----+-----+-----+
| 3      | Chocolate sonho de | 130     |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

Figura 4 – Tela do MySQL após diversos comandos SELECT utilizando a função Total_Vendas2()
Fonte: MySQL Server 5.1

Finalmente, para excluir uma **função** do banco de dados, você deve utilizar o comando DROP FUNCTION, o qual tem sua sintaxe descrita no destaque abaixo. E a resposta do SGBD a esse comando aplicado nas funções criadas anteriormente é ilustrada na Figura 5.

mysql> DROP FUNCTION nome_da_funcao;

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe


mysql> DROP FUNCTION Total_Vendas;
-> !
Query OK, 0 rows affected (0.01 sec)

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

mysql>

```

Figura 5 – Tela do MySQL após os comandos DROP FUNCTION. Fonte: MySQL Server 5.1



Pratique 02


1. Vamos praticar um pouco para que você se familiarize com a utilização de funções escalares. Entre no banco de dados **sispagamentos** (Aula 14). Considerando as funções que você criou na atividade anterior, elabore o que se pede.
 - a. Execute as funções e imprima seu resultado na tela.
 - b. Utilize as funções para obter uma relação dos dados dos empregados que têm um desconto de INSS maior que R\$ 50,00.
 - c. 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.
 - d. Exclua do banco de dados as funções anteriores.

Encerramos por aqui nossa aula sobre **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 de funções. Escreva no seu caderno todos os comandos SQL (e respectivas funções) aprendidos nesta aula para não esquecer. Bons estudos e boa sorte!

Resumo

Nesta aula, estudamos a criação e utilização das **funções**. Aprendemos a usar os comandos CREATE FUNCTION e DROP FUNCTION para criar e apagar uma função. 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 "@". Vimos também o uso das funções no processamento de consultas.



Evolucao

1. Considere o banco de dados CursoX criado na autoavaliação da Aula 9 cuja estrutura de tabelas é apresentada abaixo.

TABELA alunos

ATRIBUTO	TIPO	DESCRIÇÃO
aluno_cod	Número inteiro	Código do aluno
aluno_nome	Alfanumérico	Nome do aluno
aluno_endereco	Alfanumérico	Endereço do aluno
aluno_cidade	Alfanumérico	Cidade do aluno

TABELA disciplina

ATRIBUTO	TIPO	DESCRIÇÃO
dis_cod	Número inteiro	Código da disciplina
dis_nome	Alfanumérico	Nome da disciplina
dis_carga	Número inteiro	Carga horária da disciplina
dis_professor	Alfanumérico	Professor da disciplina

TABELA professores

ATRIBUTO	TIPO	DESCRIÇÃO
prof_cod	Número inteiro	Código do professor
prof_nome	Alfanumérico	Nome do professor
prof_endereco	Alfanumérico	Endereço do professor
prof_cidade	Alfanumérico	Cidade do professor

a. Crie uma **função** que calcule a quantidade de professores e alunos que moram em uma determinada cidade.

b. Crie uma **função** que calcule a carga horária média de um determinado professor. Elabore uma consulta usando a **função** criada.

c. 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.
- d.rie 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.

Referencias

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

MySQL 5.1 Reference Manual. Disponível em: <<http://dev.mysql.com/doc/refman/5.1/en/>>. Acesso em: 24 set. 2010.

WIKIPÉDIA. **SQL**. Disponível em: <<http://pt.wikipedia.org/wiki/SQL>>. Acesso em: 24 set. 2010.

[Voltar](#)[Imprimir](#)[Topo](#)