



## Cursos


► Listagem de disciplinas

Selecione uma disciplina

## Aulas

- 01 Introdução a Banco de Dados
- 02 Modelo de Entidade e Relacionamento
- 03 Modelo Relacional
- 04 Transformações ER para MR
- 05 Transformações ER para MR e dicionário de dados
- 06 Normalização básica
- 07 Normalização avançada
- 08 Introdução à Linguagem SQL e Sistemas Gerenciadores de Banco de Dados
- 09 Linguagem SQL - criação, inserção e modificação de tabelas
- 10 Linguagem SQL - Consulta simples de tabelas
- 11 Linguagem SQL - Consulta avançada de tabelas
- 12 Linguagem SQL - Alteração da estrutura de tabelas e ambientes de múltiplas tabelas
- 13 Linguagem SQL - Subconsultas
- 14 Linguagem SQL - VISÕES
- 15 Linguagem SQL - STORED PROCEDURES
- 16 Linguagem SQL - Funções
- 17 Linguagem SQL - Segurança
- 18 Engenharia Reversa
- 19 Utilizando SQL em Java
- 20 Utilizando conceitos avançados de SQL em Java


◀ Voltar    🖨 Imprimir    ⬆ Topo



### Sistemas de Banco de Dados

**Aula 09 – Linguagem SQL - criação, inserção e modificação de tabelas**


**Professores autores**  
José Josemar de Oliveira Júnior ([josemar@ect.ufrr.br](mailto:josemar@ect.ufrr.br))  
Luciana Ribeiro Veloso ([luciana.veloso@globomail.com](mailto:luciana.veloso@globomail.com))



## Apresentacao

Na aula anterior, nós vimos que existe uma linguagem padrão de acesso aos bancos de dados denominada de SQL e conhecemos um pouco de sua história. Em seguida, aprendemos que para usar um banco de dados é necessário instalar um Sistema Gerenciador de Banco de Dados (SGBD) e que existem diversas opções com características variadas. Realizamos os procedimentos de instalação e configuração do MySQL.

Nesta aula, daremos continuidade ao estudo da linguagem SQL. Iremos aprender como criar um banco de dados e suas tabelas. Aprenderemos como inserir, atualizar e apagar dados nas tabelas.



## Objetivo

**Ao final desta aula, você será capaz de:**

- criar banco de dados e tabelas;
- inserir dados em tabelas;
- atualizar e apagar dados em tabelas.

## Criação de tabelas

Já vimos em aulas anteriores, que em um banco de dados relacional, uma tabela é um conjunto de dados organizado em uma estrutura de linhas e colunas. Em uma tabela, cada linha (registro) contém todas as informações sobre um único objeto. As colunas (atributos) caracterizam os tipos de dados que deverão constar na tabela (numéricos, alfanuméricos, datas etc.).

Agora, vamos aprender a criar tabelas com a linguagem SQL? Primeiro você precisará criar um banco de dados para armazenar todas as suas tabelas. Vamos relembrar o comando que é usado para criar bancos de dados, visto na aula anterior? Para criar um banco de dados, digite o comando apresentado no quadro a seguir, não se esqueça de teclar *ENTER* após o sinal de ponto-e-vírgula.

```
mysql> CREATE DATABASE nome_do_banco_de_dados;
```

Ao executar esse comando corretamente, você obterá como resposta do *software* a mensagem "Query OK", que informa que o banco de dados foi criado corretamente.

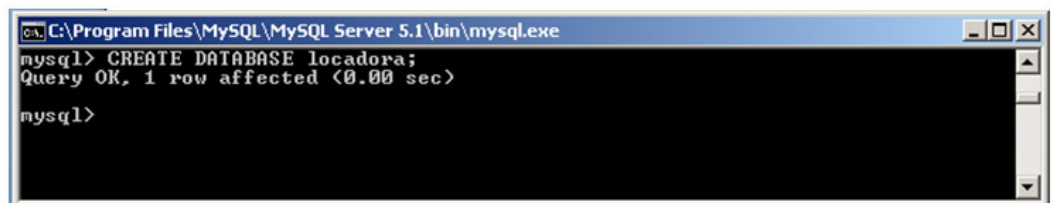
Será que estamos prontos para criar nossa primeira tabela? Não, apesar de termos criado o nosso banco de dados, que será o container das nossas tabelas, é preciso dizer ao *software* do SGBD qual o banco de dados que você quer trabalhar. A forma de "entrar" no banco de dados, para ajustar todas as suas tabelas, depende do seu SGBD. No *MySQL*, o comando é apresentado no quadro a seguir.

```
mysql> USE nome_do_banco_de_dados;
```

Agora, vamos praticar os conceitos aprendidos criando um banco de dados chamado **locadora**, no qual posteriormente vamos criar nossas tabelas. Para criarmos esse banco de dados, vamos digitar o comando a seguir:

```
mysql> CREATE DATABASE locadora;
```

A resposta do SGBD, no caso do *MySQL*, ao comando **CREATE DATABASE locadora** é ilustrada na Figura 1.

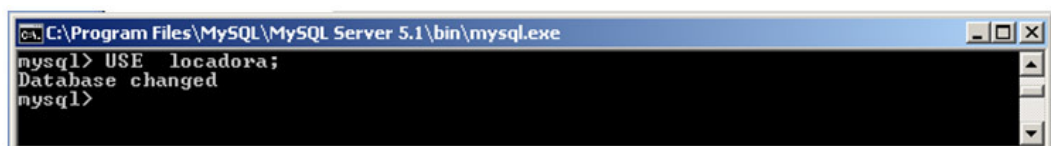


**Figura 1** – Tela do *MySQL* após a criação do banco de dados **locadora**  
Fonte: *MySQL* 5.1

O passo seguinte é dizer ao sistema que você quer utilizar o banco de dados **locadora**, através do comando:

```
mysql>USE locadora;
```

A mensagem fornecida pelo sistema ao comando **USE locadora** nos informa que o banco de dados corrente foi alterado (*Database changed*), conforme é ilustrado na Figura 2.



**Figura 2** – Tela do *MySQL* após o comando **USE locadora**  
Fonte: *MySQL* 5.1

Em geral, a maioria dos SGBDs possui editores gráficos de banco de dados que permitem a criação rápida e simples de qualquer tipo de tabela com qualquer tipo de formato. Entretanto, iremos estudar os comandos diretamente na linguagem SQL, ou seja, do modo como devem ser digitados na linha de comando.

Para criar uma tabela, devemos especificar diversos dados: o nome que queremos atribuir a essa tabela, seus atributos e seus tipos. Ademais, pode ser necessário especificar quais desses campos vão ser índices (chave primária, chave estrangeira, ...) e as restrições de integridade, a fim de evitar a inconsistência dos dados nas tabelas.

A sintaxe de criação pode variar ligeiramente entre os diferentes SGBDs, já que os tipos de campos aceitos não são completamente padronizados. O comando para criar uma tabela é semelhante ao comando de criação de um banco de dados (CREATE DATABASE nome\_do\_banco;), conforme é apresentado no quadro a seguir.

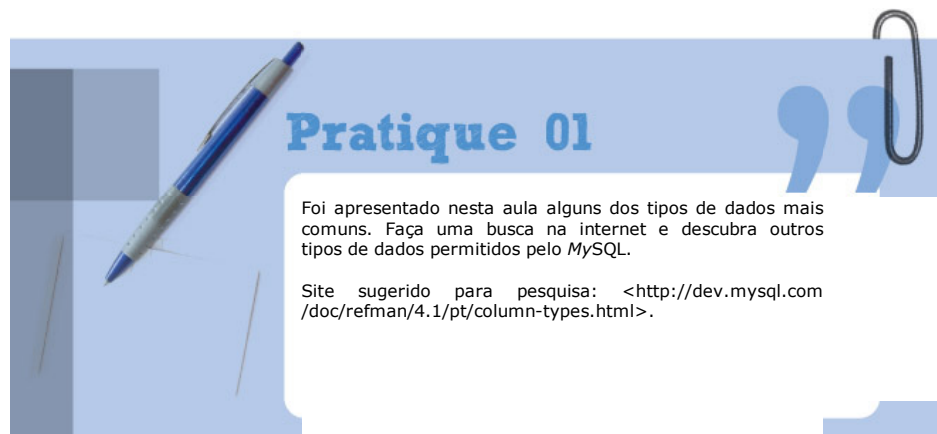
```
mysql>CREATE TABLE nome_da_tabela
(
    atributo 1 tipo1,
    atributo 2 tipo 2,
    ...
    atributo N tipo N
);
```

É importante que você lembre de usar um parêntese aberto antes do início da lista de atributos e um parêntese de fechamento após o final da definição dos atributos. Certifique-se de separar cada definição de coluna com uma vírgula. Lembre-se: todas as declarações SQL devem terminar com um ";".

No momento da criação de uma tabela em um banco de dados, devemos definir para cada atributo o seu respectivo tipo. Os tipos mais comuns de dados são:

- CHAR (tamanho) – sequência de caracteres (string) de comprimento fixo. O tamanho é especificado entre parênteses. O tamanho máximo permitido é de 255 caracteres.
- VARCHAR (tamanho) - sequência de caracteres (string) com tamanho variável. O tamanho máximo, ou seja, a quantidade máxima de caracteres que poderá ser armazenada no campo é especificada entre parênteses.
- INT – tipo numérico que aceita valores inteiros. Podemos representar com esse tipo qualquer valor inteiro na faixa entre -2.147.483.648 e 2.147.483.647, o que é mais que suficiente para armazenar a idade de alguém!
- NUMERIC(n,d) – tipo numérico que aceita valores reais (n indica a quantidade total de números e d indica a quantidade do total que corresponde a casas decimais). Exemplo: NUMERIC(5,2) corresponde a números com 5 dígitos com até duas casas decimais, como 256,12.
- TIME – tipo tempo no formato hora:minuto:segundo.
- DATE – tipo data no formato ano-mês-dia.

Quando as tabelas são criadas, é comum que uma ou mais colunas tenham **restrições** que lhes estão associadas. Restrição é basicamente uma regra associada a uma coluna que diz quais as limitações dos dados inseridos nessa coluna. Por exemplo, a restrição UNIQUE especifica que dois registros não podem ter o mesmo valor em uma determinada coluna. Eles devem ser todos originais. As restrições mais populares são NOT NULL e PRIMARY KEY. A restrição NOT NULL especifica que uma coluna não pode ser deixada em branco. E a restrição PRIMARY KEY (chave primária) define uma identificação única de cada registro (ou linha) em uma tabela. Iremos aprender mais sobre restrições no decorrer da disciplina.



## Pratique 01

Foi apresentado nesta aula alguns dos tipos de dados mais comuns. Faça uma busca na internet e descubra outros tipos de dados permitidos pelo MySQL.

Site sugerido para pesquisa: <<http://dev.mysql.com/doc/refman/4.1/pt/column-types.html>>.

Neste ponto, estamos aptos a criar algumas tabelas no nosso banco de dados chamado **locadora**. Duas tabelas importantes no banco de dados da nossa locadora são as que contêm as informações sobre os clientes, denominada **clientes**, e a outra com as informações sobre os filmes, denominada **filmes**. Os comandos para criação dessas duas tabelas são apresentados a seguir. Examine com cuidado e não deixe de praticar em seu banco de dados. Lembre-se que a prática leva à perfeição!!!

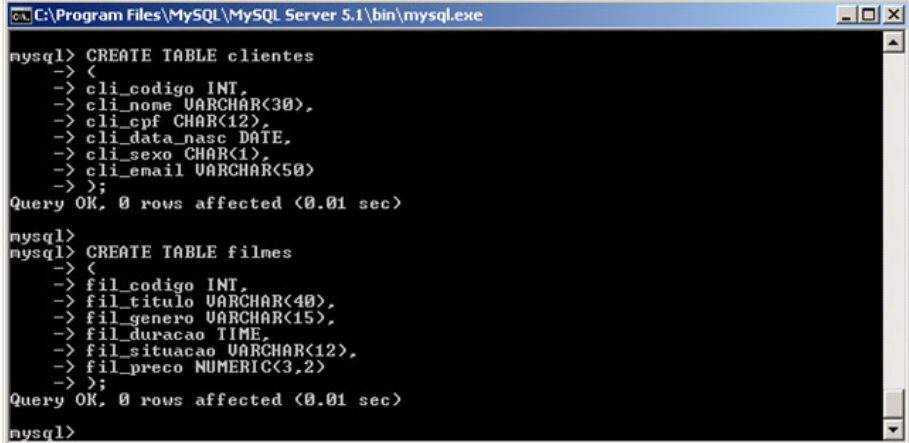
```
mysql>CREATE TABLE clientes
(
    cli_codigo INT,
    cli_nome VARCHAR(30),
    cli_cpf CHAR(12),
    cli_data_nasc DATE,
```

```

        cli_sexo CHAR(1),
        cli_email VARCHAR(50)
    );
mysql> CREATE TABLE filmes
(
    fil_codigo INT,
    fil_titulo VARCHAR(40),
    fil_genero VARCHAR(15),
    fil_duracao TIME,
    fil_situacao VARCHAR(12),
    fil_preco NUMERIC(3,2)
);

```

É uma boa prática de programação colocar na frente do atributo uma informação que permita identificar de forma simples, por exemplo, a qual tabela aquele atributo pertence. No exemplo, na tabela **clientes** foi adicionada uma abreviação da palavra clientes (**cli**) antes de cada atributo da tabela. Essa prática evita confusões de atributos iguais (por exemplo, código) nas tabelas **clientes** e **filmes**.



```

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

mysql> CREATE TABLE clientes
-> <
-> cli_codigo INT,
-> cli_nome VARCHAR(30),
-> cli_cpf CHAR(12),
-> cli_data_nasc DATE,
-> cli_sexo CHAR(1),
-> cli_email VARCHAR(50)
-> >;
Query OK, 0 rows affected (0.01 sec)

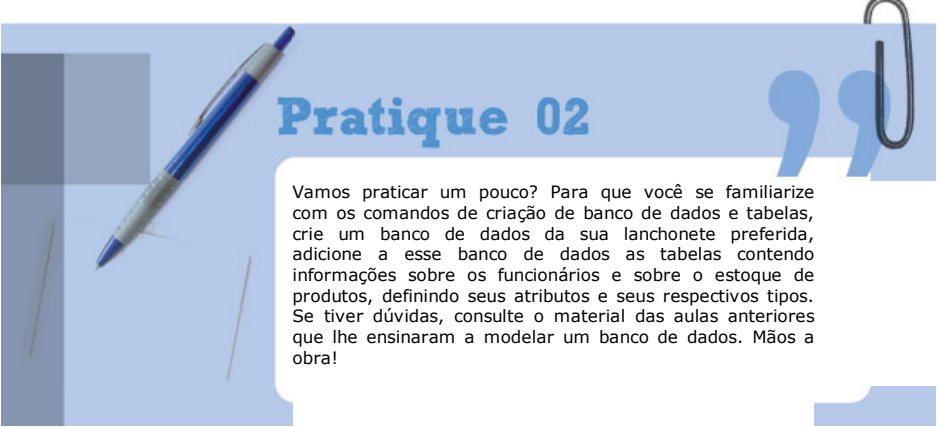
mysql>
mysql> CREATE TABLE filmes
-> <
-> fil_codigo INT,
-> fil_titulo VARCHAR(40),
-> fil_genero VARCHAR(15),
-> fil_duracao TIME,
-> fil_situacao VARCHAR(12),
-> fil_preco NUMERIC(3,2)
-> >;
Query OK, 0 rows affected (0.01 sec)

mysql>

```

**Figura 3** – Tela do MySQL após os comandos CREATE TABLE **clientes** e CREATE TABLE **filmes**.  
Fonte: MySQL 5.1

As respostas do SGBD, no caso o MySQL, aos comandos CREATE TABLE **clientes** e CREATE TABLE **filmes** são ilustradas na Figura 3. A mensagem "Query OK" informa que as tabelas **clientes** e **filmes** foram criadas corretamente.



## Pratique 02

Vamos praticar um pouco? Para que você se familiarize com os comandos de criação de banco de dados e tabelas, crie um banco de dados da sua lanchonete preferida, adicione a esse banco de dados as tabelas contendo informações sobre os funcionários e sobre o estoque de produtos, definindo seus atributos e seus respectivos tipos. Se tiver dúvidas, consulte o material das aulas anteriores que lhe ensinaram a modelar um banco de dados. Mãos à obra!

## Inserir dados nas tabelas

Inserir dados em uma tabela significa preencher as linhas de uma tabela com dados correspondentes aos tipos determinados durante a criação da tabela. A sintaxe para incluir dados em uma tabela é descrita no quadro abaixo.

```

mysql> INSERT INTO nome_da_tabela (atributo1, atributo2, ...)
VALUES (valor1, valor2, ...);

```

Os dados do tipo CHAR, VARCHAR, DATE, TIME (texto em geral, ainda que seja apenas um caractere como A ou B) devem ser representados entre aspas simples (''). Os dados do tipo INT ou NUMERIC não são representados por aspas simples. Observe ainda que as casas decimais dos números devem ser separadas por pontos ao invés de vírgulas, e os valores do tipo VARCHAR podem conter acentos e espaços em branco.

É importante ressaltar que os valores valor1, valor2, ..., seguem a mesma ordem dos atributos listados na primeira linha do comando INSERT. Essa lista de atributos é usada para indicar os atributos da tabela que devem ser preenchidos e com que valores. Essa lista é obrigatória quando alguns campos não forem preenchidos, ou quando a ordem dos valores informados for diferente da ordem definida na criação da tabela.

Vamos praticar o comando INSERT adicionando dados na tabela **clientes** criada anteriormente?

```
mysql>INSERT INTO clientes (cli_codigo, cli_nome, cli_cpf, cli_data_nasc, cli_sexo, cli_email)
VALUES (1, 'José da Silva', '123456789-10', '1980-12-10', 'M', 'joseSilva@cursoSQL.com');
```

```
mysql>INSERT INTO clientes (cli_codigo, cli_email, cli_nome, cli_cpf, cli_data_nasc, cli_sexo)
VALUES (2, 'mariaSilva@cursoSQL.com', 'Maria da Silva', '012345678-99', '1982-02-28', 'F');
```

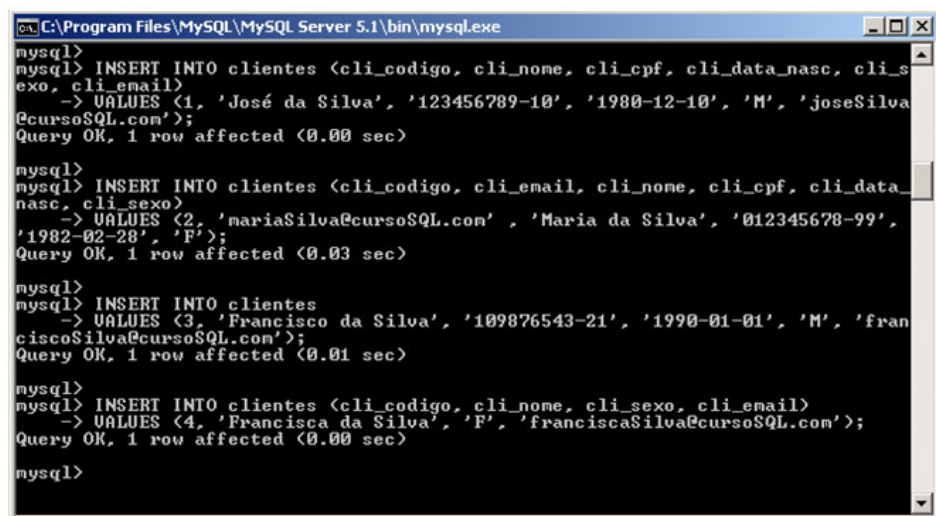
```
mysql>INSERT INTO clientes
VALUES (3, 'Francisco da Silva', '109876543-21', '1990-01-01', 'M', 'franciscoSilva@cursoSQL.com');
```

```
mysql>INSERT INTO clientes (cli_codigo, cli_nome, cli_sexo, cli_email)
VALUES (4, 'Francisca da Silva', 'F', 'franciscaSilva@cursoSQL.com');
```

Observe com cuidado as duas primeiras inserções de dados na tabela. Percebeu a ordem dos atributos? Agora olhe para os valores, eles estão na mesma ordem dos atributos listados. Desde que os valores informados sejam referentes aos respectivos atributos listados, a ordem utilizada no INSERT não importa. Agora, analise a terceira inserção de dados na tabela **clientes**. A lista de atributos foi omitida no comando INSERT. Neste caso, os valores devem estar todos ali, e na mesma ordem que os atributos foram definidos durante a criação da tabela. No último exemplo de inserção, alguns valores não foram inseridos. Como o seu sistema SQL não sabe a que atributos esses valores pertencem, você deve especificá-los na lista de atributos.

Você pode estar se perguntando o que acontece com os atributos que não tiveram seus campos preenchidos com o comando INSERT. Nesse caso, os campos não informados serão preenchidos com NULL. Um valor NULL é um valor indefinido.

A mensagem "QUERY OK, 1 ROW AFFECTED" fornecida pelo sistema ao comando INSERT INTO **clientes** nos informa que uma linha de dados foi corretamente inserida na tabela **clientes**, conforme é ilustrado na Figura 4.



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql>
mysql> INSERT INTO clientes (cli_codigo, cli_nome, cli_cpf, cli_data_nasc, cli_sexo, cli_email)
-> VALUES (1, 'José da Silva', '123456789-10', '1980-12-10', 'M', 'joseSilva@cursoSQL.com');
Query OK, 1 row affected (0.00 sec)

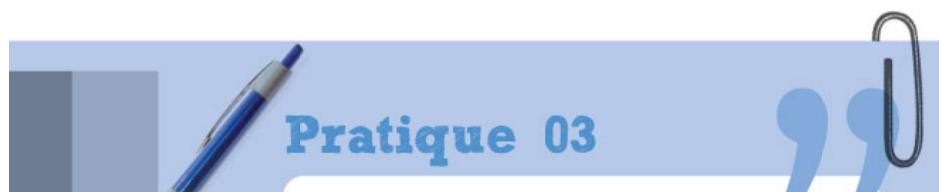
mysql>
mysql> INSERT INTO clientes (cli_codigo, cli_email, cli_nome, cli_cpf, cli_data_nasc, cli_sexo)
-> VALUES (2, 'mariaSilva@cursoSQL.com', 'Maria da Silva', '012345678-99', '1982-02-28', 'F');
Query OK, 1 row affected (0.03 sec)

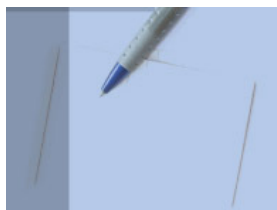
mysql>
mysql> INSERT INTO clientes
-> VALUES (3, 'Francisco da Silva', '109876543-21', '1990-01-01', 'M', 'franciscoSilva@cursoSQL.com');
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> INSERT INTO clientes (cli_codigo, cli_nome, cli_sexo, cli_email)
-> VALUES (4, 'Francisca da Silva', 'F', 'franciscaSilva@cursoSQL.com');
Query OK, 1 row affected (0.00 sec)

mysql>
```

Figura 4 - Tela do MySQL após os comandos INSERT INTO **clientes**.  
Fonte: MySQL 5.1





1) Seu SGBD avisa a você quando algo está errado no seu código, mas às vezes a resposta é meio vaga. Examine abaixo os comandos INSERT e tente descobrir o que há de errado com o comando, a seguir digite no seu sistema e observe a mensagem exibida.

a) INSERT INTO filmes(fil\_codigo, fil\_titulo, fil\_genero, fil\_duracao, fil\_situacao, fil\_preco) VALUES (1, 'E o vento Levou', 'romance', 'alugado', 5.00);

b) INSERT INTO filmes(fil\_codigo, fil\_titulo, fil\_genero, fil\_duracao, fil\_situacao) VALUES (2, 'O silêncio dos inocentes', 'policial', '0:02:00', 'disponível', 02.50);

c) INSERT INTO filmes VALUES (3, 'Procurando Nemo', 'animação', '0:01:40' 'alugado', 02.50);

d) INSERT INTO filmes(fil\_codigo, fil\_titulo, fil\_genero, fil\_situacao, fil\_duracao) VALUES (4, 'Cidade de Deus', 'ação', 'disponível', 0:02:10);

2) Corrija os comandos do exercício anterior e insira de forma correta os dados na tabela **filmes** do banco de dados **locadora**.

## Modificação de tabelas

Vamos supor que um cliente da nossa locadora, o Sr. José da Silva, mudou seu email. Sendo assim, para que nosso banco de dados não fique desatualizado, devemos atualizar a nossa tabela **clientes**. Para isso, podemos utilizar o comando UPDATE (atualizar), que altera apenas os valores das colunas especificadas.

A sintaxe do comando UPDATE é descrita no quadro a seguir.

```
mysql>UPDATE nome_da_tabela
SET atributo = valor
```

```
WHERE condição
```

A palavra SET diz ao SGBD para alterar o atributo antes do sinal de igual para conter o dado cujo valor é especificado depois do sinal de igualdade. A cláusula WHERE é opcional, mas quando está presente diz ao sistema para mudar somente as linhas que tenham a condição atendida. Se a condição não for informada, a atualização será realizada em toda a tabela, então, você deve tomar muito cuidado ao atualizar um campo sem a cláusula WHERE.

Você pode utilizar o UPDATE com um único atributo ou com um conjunto de atributos, mas para isso é preciso adicionar mais pares de **atributo = valor** na cláusula SET. Não se esqueça de colocar vírgula após cada um dos pares de **atributo = valor**.

Vamos exercitar a utilização do comando UPDATE fazendo pequenas atualizações nas tabelas **clientes** e **filmes** do nosso banco de dados **locadora**.

Inicialmente, vamos mudar o email do cliente com nome José da Silva para [SilvaJose@cursosql.com](mailto:SilvaJose@cursosql.com) do seguinte modo:

```
mysql>UPDATE clientes
SET cli_email = 'SilvaJose@cursosql.com'
WHERE cli_nome = 'José da Silva';
```

Veja que especificamos na cláusula WHERE que o email a ser atualizado era do cliente José da Silva, caso não tivesse especificado, todos os emails cadastrados na tabela clientes teriam sido alterados.

Se quisermos alterar o valor cobrado para locação de um filme, por exemplo, oferecendo um desconto de R\$ 1 em todos os filmes, podemos usar a seguinte estrutura:

```
mysql>UPDATE filmes
SET fil_preco = fil_preco - 1;
```

A estrutura (fil\_preco = fil\_preco - 1) é válida, pois fil\_preco é um atributo numérico. Qualquer operação aritmética básica pode ser utilizada na construção desse tipo de estrutura (+ para adição, - para subtração, \* para multiplicação e / para divisão).

Caso queira especificar o preço de todos os filmes para, por exemplo, R\$ 4.50, faça a atualização do seguinte modo:

```
mysql>UPDATE filmes  
SET fil_preco = 4.50;
```



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe  
mysql>  
mysql> UPDATE clientes  
-> SET cli_email = 'SilvaJose@cursosql.com'  
-> WHERE cli_nome='José da Silva';  
Query OK, 1 row affected (0.00 sec)  
Rows matched: 1 Changed: 1 Warnings: 0  
  
mysql>  
mysql> UPDATE filmes  
-> SET fil_situacao = 'disponível'  
-> WHERE fil_codigo=1;  
Query OK, 1 row affected (0.00 sec)  
Rows matched: 1 Changed: 1 Warnings: 0  
  
mysql>  
mysql> UPDATE filmes  
-> SET fil_preco = 4.50;  
Query OK, 4 rows affected (0.00 sec)  
Rows matched: 4 Changed: 4 Warnings: 0  
  
mysql>
```

**Figura 5** – Tela do MySQL após diversos comandos UPDATE.  
Fonte: MySQL 5.1

A mensagem fornecida pelo sistema ao comando UPDATE informa se o comando foi realizado com sucesso e quantas linhas da tabela foram alteradas, conforme é ilustrado na Figura 5.

O próximo comando a ser estudado é o DELETE que exclui linhas simples ou linhas múltiplas dependendo da cláusula WHERE. A sintaxe do comando DELETE é descrita no quadro a seguir.

```
mysql>DELETE FROM nome_da_tabela  
WHERE condição;
```

Você não pode utilizar o comando DELETE para apagar o valor de um atributo ou de uma porção de atributos. Mas sim para apagar linhas simples ou múltiplas dependendo da cláusula WHERE. Atenção, a cláusula WHERE é opcional no comando DELETE, se não for informada você pode excluir todas as linhas de uma tabela, mas não exclui a tabela do Banco de dados. Para excluir a tabela inteira (dados e estrutura) do banco de dados você deve utilizar o comando DROP, o qual tem sua sintaxe descrita no quadro abaixo.

```
mysql>DROP TABLE nome_da_tabela;
```

Exemplos:

Apagar cadastros de todos os clientes do sexo masculino:

```
mysql>DELETE FROM clientes  
WHERE cli_sexo = 'M';
```

Apagar cadastros de todos os filmes de terror:

```
mysql>DELETE FROM filmes  
WHERE fil_genero= 'terror';
```

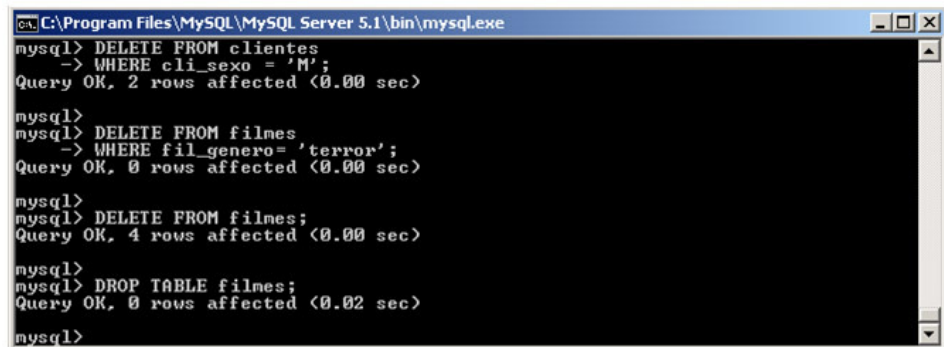
Apagar o cadastro de todos os filmes:

```
mysql>DELETE FROM filmes;
```

Excluir a tabela **filmes** do banco de dados:



```
mysql> DROP TABLE filmes;
```



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql> DELETE FROM clientes
  -> WHERE cli_sexo = 'M';
Query OK, 2 rows affected (0.00 sec)

mysql>
mysql> DELETE FROM filmes
  -> WHERE fil_genero = 'terror';
Query OK, 0 rows affected (0.00 sec)

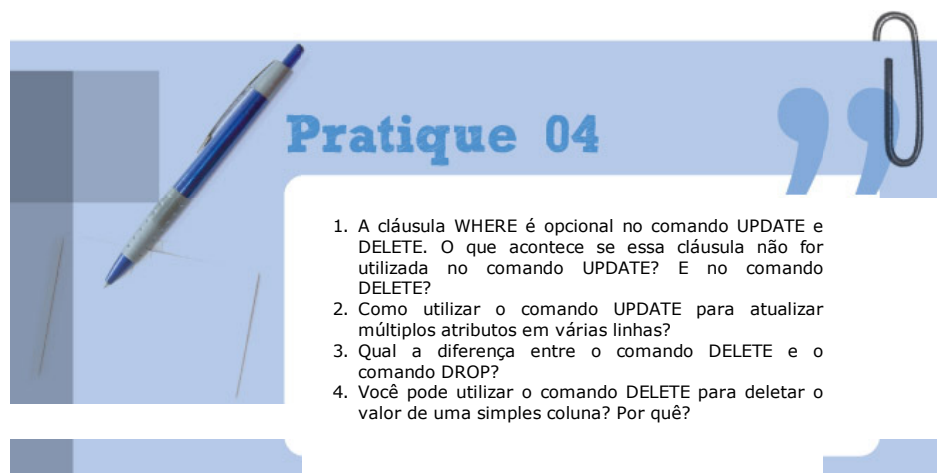
mysql>
mysql> DELETE FROM filmes;
Query OK, 4 rows affected (0.00 sec)

mysql>
mysql> DROP TABLE filmes;
Query OK, 0 rows affected (0.02 sec)

mysql>
```

**Figura 6** – Tela do MySQL após os comandos DELETE FROM e DROP TABLE.  
Fonte: MySQL 5.1

As respostas do SGBD, no caso o MySQL, aos comandos DELETE FROM e DROP TABLE são ilustradas na Figura 6.



## Pratique 04

1. A cláusula WHERE é opcional no comando UPDATE e DELETE. O que acontece se essa cláusula não for utilizada no comando UPDATE? E no comando DELETE?
2. Como utilizar o comando UPDATE para atualizar múltiplos atributos em várias linhas?
3. Qual a diferença entre o comando DELETE e o comando DROP?
4. Você pode utilizar o comando DELETE para deletar o valor de uma simples coluna? Por quê?

Bom, concluímos por aqui nossa segunda aula sobre a linguagem SQL, mas temos um longo caminho pela frente. Na próxima aula, você vai conhecer o poderoso comando SELECT que lhe permitirá que você acesse as informações inseridas nas tabelas, possibilitando a construção das nossas tão desejadas consultas. Faça a autoavaliação com atenção e veja se precisa parar e refletir mais um pouco sobre o que estudamos. É uma boa escrever no seu caderno todos os comandos SQL (e respectivas funções) que está aprendendo para não esquecer. Bons estudos e boa sorte!

## Resumo

Nesta aula, vimos que para criar um banco de dados utilizamos o comando CREATE DATABASE e para entrar efetivamente no banco de dados é necessário utilizarmos o comando USE. A seguir, foram estudados os comandos CREATE TABLE, que cria a estrutura de uma tabela, e o INSERT INTO que é utilizado para preencher a tabela com os dados. Estudamos, também, como fazemos atualizações e apagamos linhas nas tabelas de um banco de dados através dos comandos UPDATE e DELETE FROM, respectivamente. E finalizamos nossa aula com o comando DROP TABLE que exclui tanto os dados como a estrutura de uma tabela. Todos os comandos aprendidos foram praticados na ferramenta MySQL através da criação e inserção de dados nas tabelas clientes e filmes do banco de dados locadora.



## Evolucao





- 1) Crie um banco de dados chamado CursoX.
- 2) Nesse banco, crie as tabelas a seguir, de acordo com as informações do Dicionário de Dados (visto na Aula 3).

#### 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

- 3) Escreva os comandos SQL necessários para inserir nessas tabelas as seguintes informações:

- dados de 10 alunos;
- dados de 5 disciplinas distintas;
- dados de 5 professores distintos.

- 4) Atualize a tabela **disciplina**, aumentando a carga horária de cada disciplina em 10 horas/ aulas.

- 5) Apague da tabela **disciplina** todos as disciplinas ministradas por um determinado professor.

## 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: 23 ago. 2010.

IKIPÉDIA. **SQL**. Disponível em: <<http://pt.wikipedia.org/wiki/SQL>>. Acesso em: 23 ago. 2010.

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

