



Cursos

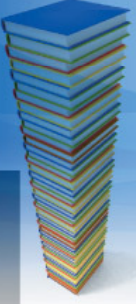
➤ Listagem de disciplinas

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 14 – Linguagem SQL – VISÕES

Professores autores
José Josemar de Oliveira Júnior (josemar@ect.ufrn.br)
Luciana Ribeiro Veloso (luciana.veloso@globo.com)



Apresentacao

Na aula anterior, aprendemos a usar o resultado de uma consulta como entrada para outra consulta, ou seja, trabalhamos com consultas aninhadas, denominadas subconsultas.

Nesta aula, aprenderemos uma forma alternativa de olhar os dados contidos em uma ou mais tabelas através das **visões** ou VIEWS. Com VIEWS, é possível tratar os resultados de uma consulta como uma tabela. É ótimo para transformar as consultas complexas em consultas simples.



Objetivo

Ao final desta aula, você será capaz de:

- criar **visões**;
- visualizar os dados contidos em uma **visão**;
- manipular os dados de uma tabela por meio de **visões** associadas a elas.

VIEWS

Visões em SQL são consultas armazenadas em uma estrutura de fácil acesso baseadas num comando SELECT. Essa consulta armazenada funciona como uma tabela virtual, com comportamento similar a uma tabela real, entretanto, sem armazenar dados, não existindo como uma entidade independente no banco de dados. Os dados que são exibidos nas **visões** são gerados dinamicamente toda vez que a **visão** é referenciada.

O SGBD armazena apenas a definição das **visões** (nome da **visão** e o comando SELECT). Quando o usuário chama uma **visão**, o sistema de banco de dados associa os dados apropriados a ela. Uma **visão** apresenta o resultado final desse processo, ocultando todos os detalhes técnicos.

A utilização de **visões** permite simplificar e personalizar tabelas no seu banco de dados. Também oferece um mecanismo de segurança (restringindo o acesso de usuários a campos predeterminados). Além disso, as **visões** mantêm os dados independentes da estrutura do banco de dados, garantindo flexibilidade para a análise e manipulação de dados.

A instrução para criar uma **visão** é bastante simples, basta adicionar as palavras CREATE VIEW antes da instrução da consulta que se deseja armazenar. A sintaxe de criação de uma **visão** é descrita no destaque abaixo.

```
mysql> CREATE VIEW nome_da_visão AS
      SELECT atributo1, atributo2, ...
      FROM nome_da_tabela1, nome_da_tabela2, ...

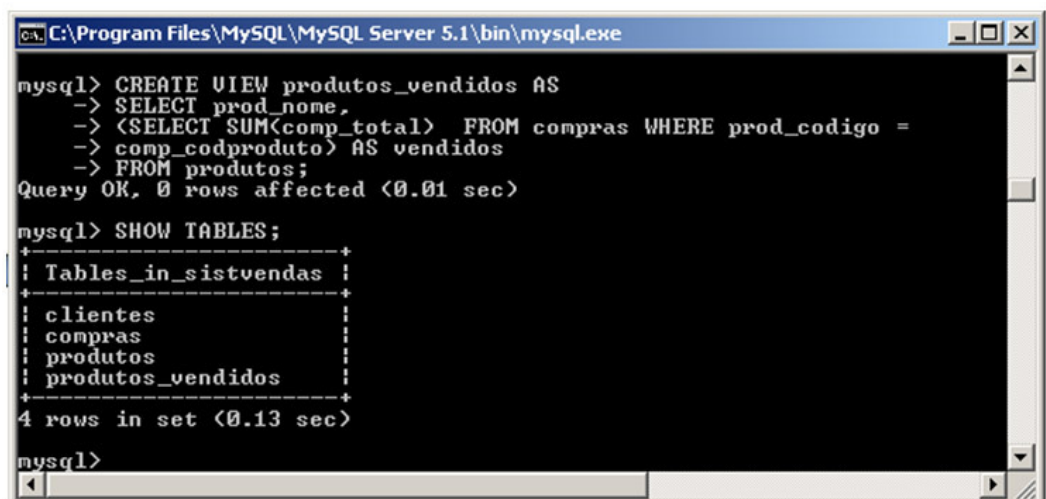
      WHERE condição;
```

Nessa expressão, no campo nome_da_visão deve-se inserir o nome que se deseja atribuir para a visão, nome esse que deve seguir as mesmas regras usadas para os nomes das tabelas, que foi visto na Aula 9. Após a cláusula AS, tem-se qualquer comando SELECT válido.

Vamos exercitar a criação de VIEWS no banco de dados **sistvendas** para entendermos melhor o seu conceito? Na aula anterior sobre subconsultas, um dos exemplos que foram trabalhados foi a consulta aos nomes de todos os produtos e suas quantidades que foram vendidas. Visto que esse tipo de pesquisa tende a ser realizado diariamente num banco de dados de um sistema de vendas, é muito útil criar uma VIEW contendo essa consulta. A criação de uma VIEW neste caso simplifica a consulta e evita que diariamente o usuário tenha de escrever uma consulta complexa, correndo o risco de cometer algum erro. O comando para criar essa VIEW é descrito no destaque abaixo.

```
mysql> CREATE VIEW produtos_vendidos AS
      SELECT prod_nome, (SELECT SUM(comp_total)
      FROM compras WHERE prod_codigo = comp_codproduto)
      AS vendidos FROM produtos;
```

A resposta do sistema SGBD para o comando acima é ilustrada na Figura 1. É interessante notar que a **visão** aparece no esquema do banco de dados como se fosse uma tabela, conforme pode ser verificado usando a instrução SHOW TABLES após a criação da **visão** (Figura 1), que exhibe como resposta o nome das tabelas contidas no banco de dados em questão.



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

mysql> CREATE VIEW produtos_vendidos AS
-> SELECT prod_nome,
-> (SELECT SUM(comp_total) FROM compras WHERE prod_codigo =
-> comp_codproduto) AS vendidos
-> FROM produtos;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_sistvendas |
+-----+
| clientes              |
| compras               |
| produtos              |
| produtos_vendidos     |
+-----+
4 rows in set (0.13 sec)

mysql>
```

Figura 1 – Tela do MySQL após os comandos CREATE VIEW e SHOW TABLES. Fonte: MySQL Server 5.1

Mas, como fazer para visualizar a estrutura de uma **visão**, ou seja, a estrutura de uma tabela virtual? A resposta a essa pergunta é o comando DESC. Para verificar a estrutura de uma **visão**, é necessário utilizar o comando DESC conforme apresentado no destaque a seguir.

```
mysql>DESC nome_da_visão;
```

A resposta do SGBD ao comando DESC produtos_vendidos é ilustrada na Figura 2. Observe que a tabela virtual **produtos_vendidos** contém dois campos: **prod_nome** e **vendidos**. O campo **vendidos** é o resultado da seguinte subconsulta:

```
(SELECT SUM(comp_total) FROM compras  
WHERE prod_codigo = comp_codproduto)
```

Essa subconsulta ou consulta interna acessa os dados da tabela **compras**, enquanto a consulta externa acessa as informações da tabela **produtos**. Portanto, a tabela virtual **produtos_vendidos** contém informações pertencentes às tabelas **produtos** e **compras**, permitindo flexibilidade e simplicidade para a análise dos dados.

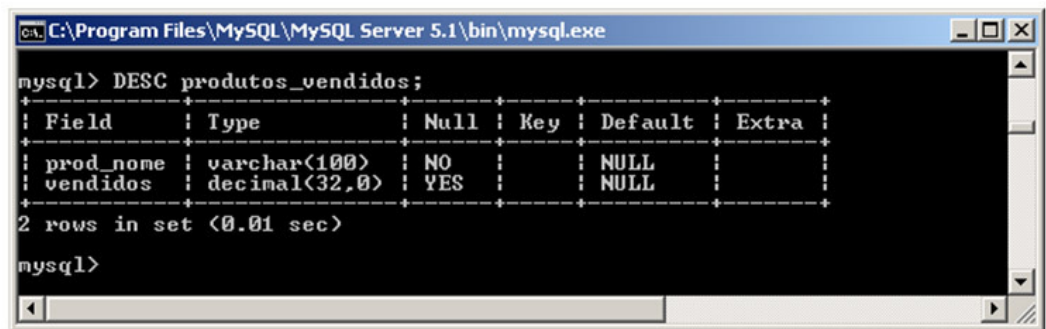


Figura 2 – Tela do MySQL após a visualização da estrutura da **visão** usando o comando DESC. Fonte: MySQL Server 5.1

Você deve estar se perguntado como fazer para visualizar os dados da consulta que foi armazenada como uma VIEW. Só faz sentido criar uma VIEW se a visualização dos seus dados for realizada de forma simplificada. E é exatamente isso que acontece. A sintaxe para visualizar todos os dados de uma VIEW é descrita no destaque abaixo.

```
mysql> SELECT * FROM nome_da_visão;
```

Sendo assim, para visualizarmos todos os dados da VIEW **produtos_vendidos**, basta digitar o seguinte comando:

```
mysql>SELECT * FROM produtos_vendidos;
```

A resposta do sistema ao comando acima é ilustrado na Figura 3. Observe que a resposta do sistema é exatamente a mesma dada pelo comando da consulta em si, conforme é mostrado na Figura 3. A diferença é que com a utilização de VISÕES, o comando necessário para visualizar os dados é simples, tornando a consulta muito mais simples e prática.

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql>
mysql> SELECT * FROM produtos_vendidos;
+-----+-----+
| prod_nome      | vendidos |
+-----+-----+
| ventilador     | NULL     |
| Celular N97    | 1        |
| chocolate sonho de valsa | 130      |
| geladeira      | 1        |
+-----+-----+
4 rows in set (0.02 sec)

mysql> SELECT prod_nome,
-> (SELECT SUM(comp_total) FROM compras WHERE prod_codigo =
-> comp_codproduto) as vendidos
-> FROM produtos;
+-----+-----+
| prod_nome      | vendidos |
+-----+-----+
| ventilador     | NULL     |
| Celular N97    | 1        |
| chocolate sonho de valsa | 130      |
| geladeira      | 1        |
+-----+-----+
4 rows in set (0.00 sec)

mysql>

```

Figura 3 – Tela do MySQL após os comandos SELECTs para visualização da quantidade total de produtos vendidos. Fonte: MySQL Server 5.1

É importante ressaltar que o SGBD armazena apenas o nome da **visão** e o comando SELECT associado. Os dados visualizados nas **visões** são gerados dinamicamente toda vez que é solicitada uma consulta sobre a VIEW. Isso implica que uma **visão** está sempre atualizada, ou seja, ao se modificar dados nas tabelas referenciadas na descrição da **visão**, uma consulta a **visão** reflete automaticamente essas alterações.

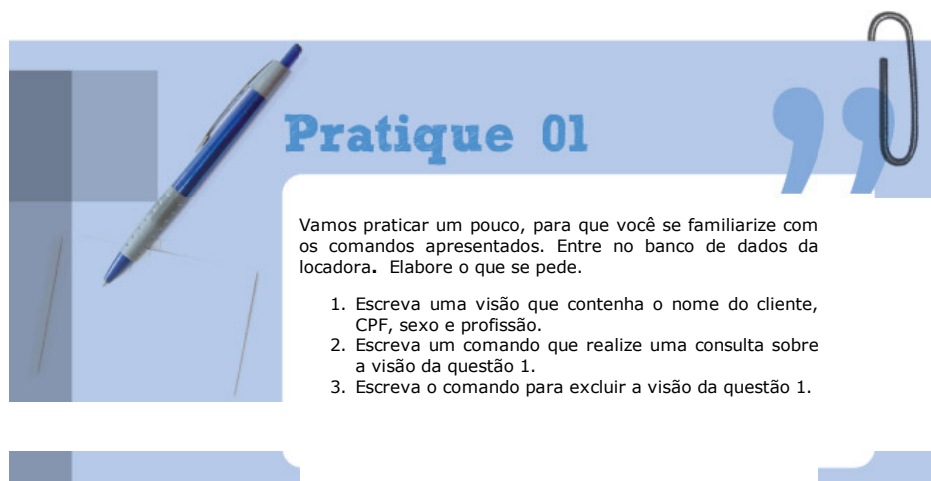
Vale salientar que uma consulta a uma **visão** pode ser realizada da mesma forma que uma consulta a uma tabela. O segredo é que a **visão** se comporta como uma tabela de verdade, no entanto, sem guardar os dados, por isso chamada de uma tabela virtual. Vejamos o seguinte exemplo: realizar uma consulta à **visão** com o objetivo de determinar quais os produtos que já foram vendidos.

```
mysql> SELECT prod_nome FROM produtos_vendidos WHERE vendidos>0;
```

Conforme pode ser observado nessa consulta, na cláusula SELECT tem-se apenas o atributo **prod_nome**, na cláusula FROM tem-se o nome da **visão** e na cláusula WHERE tem-se a condição em questão, no caso que o atributo **vendidos** deve ser maior que 0.

Quando uma **visão** não é mais necessária, pode-se excluí-la utilizando o comando descrito abaixo.

```
mysql> DROP VIEW nome_da_visão;
```



Pratique 01

Vamos praticar um pouco, para que você se familiarize com os comandos apresentados. Entre no banco de dados da locadora. Elabore o que se pede.

1. Escreva uma visão que contenha o nome do cliente, CPF, sexo e profissão.
2. Escreva um comando que realize uma consulta sobre a visão da questão 1.
3. Escreva o comando para excluir a visão da questão 1.

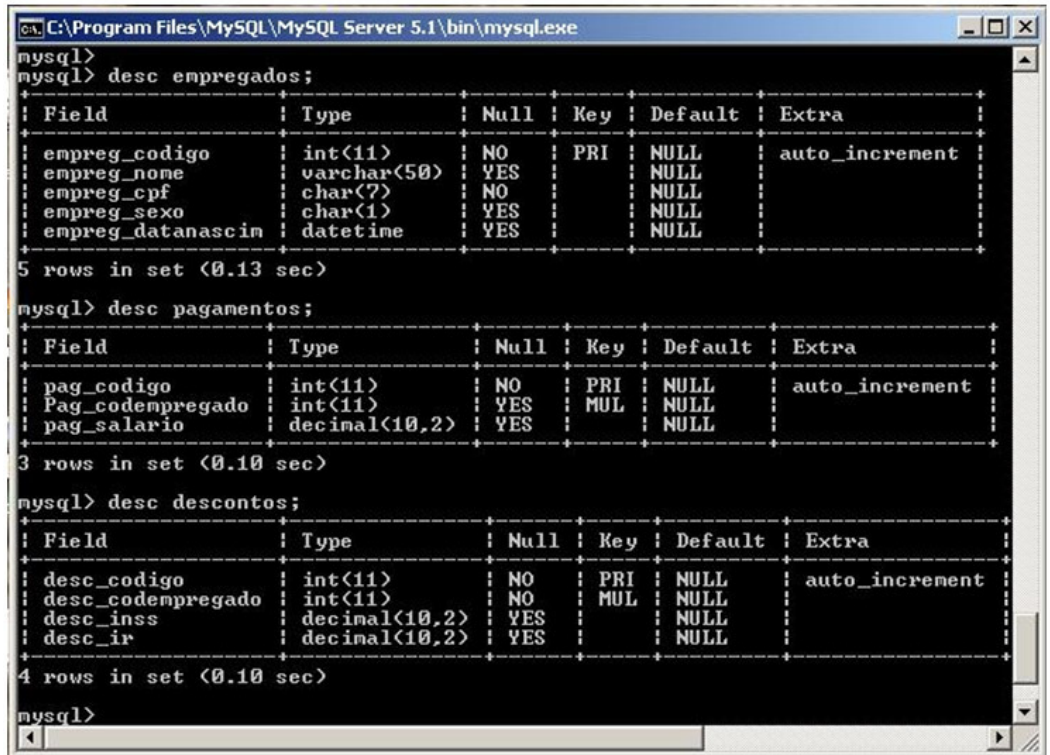
Inserindo, atualizando e apagando dados com visões

Você deve ter notado que os comandos de consulta, descrição e de exclusão de uma VIEW são iguais aos de uma tabela real. Mas, será que podemos inserir, atualizar e apagar dados através das **visões**? Em alguns casos, pode-se realmente inserir, atualizar e excluir os dados através das **visões**, desde que na **visão** não tenha valores agregados, tais como SUM, COUNT e AVG, e nem cláusulas como GROUP BY e DISTINCT.

Para continuarmos os nossos estudos, considere um banco de dados, denominado **sispagamentos**, que representa um sistema de pagamentos dos funcionários de uma determinada empresa contendo as seguintes tabelas:

- empregados (codigo_empregado [chave primária], nome, CPF, sexo e dataNascimento);
- pagamentos (codigo_pagamento [chave primária], codigo_empregado [chave estrangeira], salario);
- descontos (codigo_desconto [chave primária], codigo_empregado [chave estrangeira], INSS, IR).

As estruturas das tabelas **empregados**, **pagamentos** e **descontos** são ilustradas na Figura 4. Analise com cuidado essas estruturas e não se esqueça de implementá-las em seu SGBD, essa é uma ótima maneira de fixar os conceitos aprendidos. Inicialmente, não será necessário incluir dados nessas tabelas.



```

mysql> desc empregados;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| empreg_codigo | int(11)       | NO   | PRI | NULL    | auto_increment |
| empreg_nome   | varchar(50)   | YES  |     | NULL    |                |
| empreg_cpf    | char(7)       | NO   |     | NULL    |                |
| empreg_sexo   | char(1)       | YES  |     | NULL    |                |
| empreg_datanascim | datettime    | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.13 sec)

mysql> desc pagamentos;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| pag_codigo   | int(11)       | NO   | PRI | NULL    | auto_increment |
| Pag_codempregado | int(11)      | YES  | MUL | NULL    |                |
| pag_salario  | decimal(10,2) | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.10 sec)

mysql> desc descontos;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| desc_codigo  | int(11)       | NO   | PRI | NULL    | auto_increment |
| desc_codempregado | int(11)      | NO   | MUL | NULL    |                |
| desc_inss    | decimal(10,2) | YES  |     | NULL    |                |
| desc_ir      | decimal(10,2) | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.10 sec)

mysql>

```

Figura 4 – Tela do MySQL após os comandos de visualização das estruturas das tabelas do banco de dados sispagamentos.
Fonte: MySQL Server 5.1

Vamos criar duas **visões**. A primeira, denominada **funcionarios** exibirá os nomes dos empregados e seus respectivos CPFs. A segunda, denominada **salario**, exibirá os nomes dos empregados, o salário bruto, o desconto de INSS, o desconto de IR e o salário líquido. Os comandos para as criações das **visões funcionarios** e **salários** são apresentados no quadro abaixo. Analise com cuidado esses comandos e não deixe de implementá-los no seu SGBD.

Lembre-se, a prática leva a perfeição!

```

mysql> CREATE VIEW funcionario AS
      SELECT empreg_nome, empreg_cpf
      FROM empregados;

mysql> CREATE VIEW salario AS
      SELECT empreg_nome AS nome,
      (SELECT pag_salario FROM pagamentos
      WHERE empreg_codigo = pag_codempregado) AS salario_bruto,
      (SELECT desc_inss FROM descontos
      WHERE empreg_codigo = desc_codempregado) AS INSS,
      (SELECT desc_ir FROM descontos
      WHERE empreg_codigo = desc_codempregado) AS IR,
      (SELECT pag_salario FROM pagamentos
      WHERE empreg_codigo = pag_codempregado) -
      (SELECT desc_inss FROM descontos
      WHERE empreg_codigo = desc_codempregado) -
      (SELECT desc_ir FROM descontos
      WHERE empreg_codigo = desc_codempregado)
      AS salario_liquido FROM empregados;

```

A criação das **visões funcionario** e **salario** estão ilustradas na Figura 5 e suas respectivas estruturas são ilustradas na Figura 6.

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql>
mysql> CREATE VIEW funcionario AS
-> SELECT empreg_nome, empreg_cpf
-> FROM empregados;
Query OK, 0 rows affected (0.06 sec)

mysql>
mysql> CREATE VIEW salario AS
-> SELECT empreg_nome AS nome,
-> (SELECT pag_salario FROM pagamentos WHERE empreg_codigo = pag_codempregado) AS salario_bruto,
-> (SELECT desc_inss FROM descontos WHERE empreg_codigo = desc_codempregado) AS INSS,
-> (SELECT desc_ir FROM descontos WHERE empreg_codigo = desc_codempregado) AS IR,
-> (SELECT pag_salario FROM pagamentos WHERE empreg_codigo = pag_codempregado) -
-> (SELECT desc_inss FROM descontos WHERE empreg_codigo = desc_codempregado) -
-> (SELECT desc_ir FROM descontos WHERE empreg_codigo = desc_codempregado) AS salario_liquido
-> FROM empregados;
Query OK, 0 rows affected (0.01 sec)

mysql>

```

Figura 5 - Tela do MySQL após os comandos de criação das **visões funcionario** e **salário**. Fonte: MySQL Server 5.1

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql> DESC funcionario;
+----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| empreg_nome | varchar(50) | YES  |     | NULL    |       |
| empreg_cpf  | char(7)    | NO   |     | NULL    |       |
+----+-----+-----+-----+-----+-----+
2 rows in set (0.03 sec)

mysql>
mysql> DESC salario;
+----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| nome       | varchar(50) | YES  |     | NULL    |       |
| salario_bruto | decimal(10,2) | YES  |     | NULL    |       |
| INSS       | decimal(10,2) | YES  |     | NULL    |       |
| IR         | decimal(10,2) | YES  |     | NULL    |       |
| salario_liquido | decimal(12,2) | YES  |     | NULL    |       |
+----+-----+-----+-----+-----+-----+
5 rows in set (0.03 sec)

mysql>

```

Figura 6 - Tela do MySQL após os comandos DESC **funcionario** e DESC **salário**. Fonte: MySQL Server 5.1

Agora, vamos analisar os exemplos a seguir para entendermos como funciona os comandos INSERT, UPDATE e DELETE com **visões**.

Exemplo 1

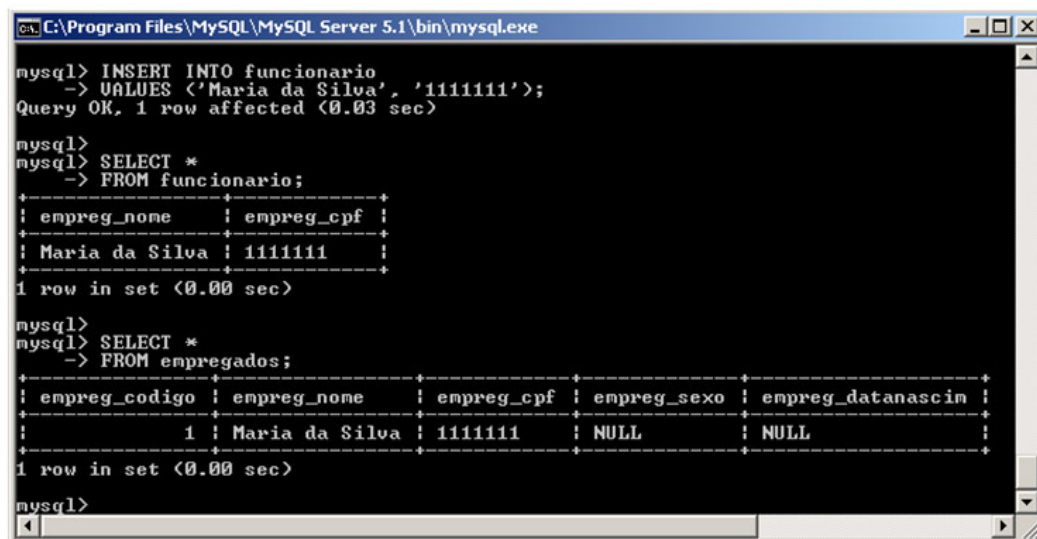
Inserir o nome do funcionário Maria da Silva e o seu CPF 1111111 na tabela **empregados** por meio da **visão funcionario**.

```

mysql> INSERT INTO funcionario
VALUES ('Maria da Silva', '1111111');

```

A resposta do SGBD, no caso, o MySQL, ao comando acima é ilustrada na Figura 7. Visando uma melhor compreensão do que foi realizado pelo sistema ao processar o comando acima, são exibidos, também, na Figura 7 os dados existentes na **visão funcionario** e na tabela **empregados** após o referido comando de inclusão (INSERT). Observe que os valores inseridos foram corretamente introduzidos na tabela **empregados**. No campo **empreg_codigo**, não foi inserido nenhum valor, entretanto, esse campo apresenta valor igual a 1, pois foi definido com sendo AUTO_INCREMENT (Figura 4) e ao introduzir um novo registro nesta tabela ele é automaticamente incrementado. Nos campos **empreg_sexo** e **empreg_datanascimento**, não foram introduzidos nenhum valor, assumindo o seu valor padrão NULL.



```

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

mysql> INSERT INTO funcionario
-> VALUES ('Maria da Silva', '1111111');
Query OK, 1 row affected (0.03 sec)

mysql>
mysql> SELECT *
-> FROM funcionario;
+-----+-----+
| empreg_nome | empreg_cpf |
+-----+-----+
| Maria da Silva | 1111111 |
+-----+-----+
1 row in set (0.00 sec)

mysql>
mysql> SELECT *
-> FROM empregados;
+-----+-----+-----+-----+-----+
| empreg_codigo | empreg_nome | empreg_cpf | empreg_sexo | empreg_datanascim |
+-----+-----+-----+-----+-----+
| 1 | Maria da Silva | 1111111 | NULL | NULL |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

Figura 7 – Tela do MySQL após os comandos INSERT e SELECT. Fonte: MySQL Server 5.1

Mas, será que podemos inserir o nome de um funcionário na tabela **empregados** através da **visão salario**? Para responder a essa pergunta, vamos analisar a estrutura da tabela **empregados** (Figura 4). A tabela **empregados** contém 5 campos, entre os quais tem o campo **empreg_cpf** que foi definido como sendo do tipo CHAR e não aceita o valor NULL, ou seja, não se pode inserir um registro nesta tabela sem que seja inserido um valor no campo **empreg_cpf**. Portanto, não podemos atualizar a tabela **empregados** através da **visão salario**.

Quando uma **visão** incluir todas as colunas que possuem a restrição NOT NULL de todas as tabelas a qual ela faz referência, então, dizemos que a **visão** é atualizável. Uma **visão** não atualizável é aquela que não inclui todas as colunas NOT NULL das tabelas à qual ela faz referência.

Portanto, a **visão salario** não é atualizável, pois ela não contém os campos empreg_CPF (tabela **empregados**) e desc_codempregados (tabela **descontos**). Observe que os atributos empreg_codigo (tabela **empregados**), pag_codigo (tabela **pagamentos**) e desc_codigo (tabela **descontos**) não são considerados nessa discussão, pois mesmo sendo NOT NULL são do tipo AUTO INCREMENT.

Exemplo 2

Atualize o nome da funcionária Maria de Silva para Maria da Silva Fernandes através da **visão funcionario**.

```
mysql> UPDATE funcionario
      SET empreg_nome = 'Maria da Silva Fernandes';
```

Observe que o comando para atualizar dados de uma tabela a partir de uma **visão** tem a mesma sintaxe de um comando para atualização de dados em tabelas (como visto na Aula 9). Vale ressaltar que apenas os campos observados através da **visão** podem ser atualizados e apenas nas **visões** atualizáveis. A resposta do SGBD, no caso o MySQL, é ilustrada na Figura 8.

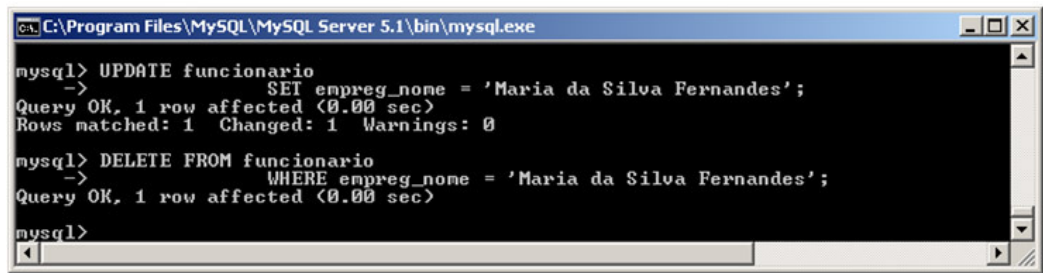
Exemplo 3

Apague o registro da funcionária Maria de Silva Fernandes através da **visão funcionario**.

A sintaxe do comando DELETE para excluir dados de uma tabela a partir de uma **visão** é exatamente o mesmo utilizado para apagar dados em tabelas, conforme pode ser verificado no quadro abaixo, que exclui todos os campos do registro de Maria da Silva Fernandes.

```
mysql> DELETE FROM funcionario
      WHERE empreg_nome = 'Maria da Silva Fernandes';
```

A resposta do SGBD, no caso o MySQL, ao comando acima é ilustrada na Figura 8.



```

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

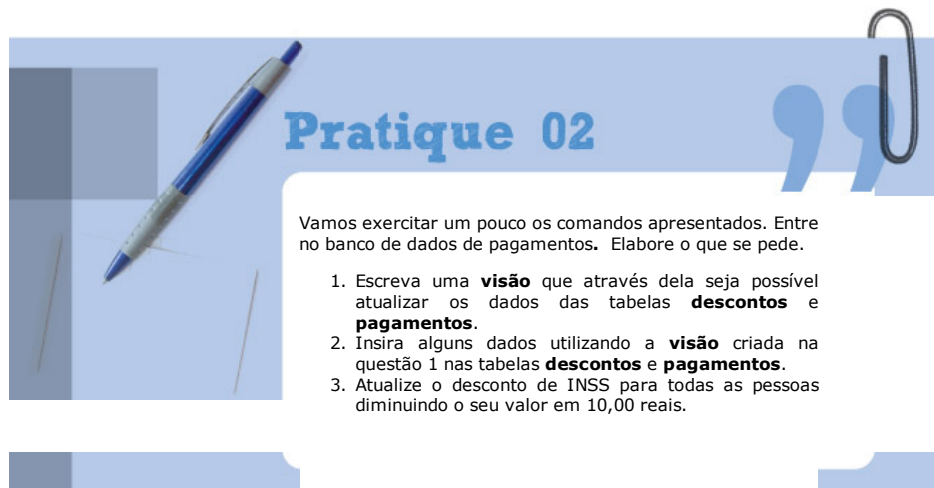
mysql> UPDATE funcionario
->      SET empreg_nome = 'Maria da Silva Fernandes';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> DELETE FROM funcionario
->      WHERE empreg_nome = 'Maria da Silva Fernandes';
Query OK, 1 row affected (0.00 sec)

mysql>

```

Figura 8 – Tela do MySQL após os comandos UPDATE e DELETE aplicados a **visão funcionario**. Fonte: MySQL Server 5.1



Pratique 02

Vamos exercitar um pouco os comandos apresentados. Entre no banco de dados de pagamentos. Elabore o que se pede.

1. Escreva uma **visão** que através dela seja possível atualizar os dados das tabelas **descontos** e **pagamentos**.
2. Insira alguns dados utilizando a **visão** criada na questão 1 nas tabelas **descontos** e **pagamentos**.
3. Atualize o desconto de INSS para todas as pessoas diminuindo o seu valor em 10,00 reais.

Encerramos por aqui mais uma aula sobre a linguagem SQL. Na próxima aula, aprenderemos a criar procedimentos armazenados (*stored procedures*), que é um recurso muito útil no gerenciamento de sistemas de banco de dados. Bons estudos e boa sorte!

Resumo

Nesta aula, estudamos uma forma alternativa de olhar os dados contidos em uma ou mais tabelas através das **visões** ou VIEWS. Com **visões**, é possível tratar os resultados de uma consulta como uma tabela. É ótimo para transformar as consultas complexas em consultas simples. Estudamos como criar uma **visão** utilizando o comando CREATE VIEW. A seguir, foi estudado como visualizar os dados da **visão** utilizando o comando SELECT. Estudamos, também, como inserir, atualizar e excluir dados nas tabelas referenciadas em uma **visão** através dos comandos INSERT, UPDATE e DELETE aplicados na **visão**.



Evolucao

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

- Crie uma **visão** que mostre os nomes dos professores que moram em Natal. Consulte a **visão** criada.
- Crie uma **visão** que mostre a quantidade de alunos que moram em cada cidade. Consulte a **visão** criada.
- Crie uma **visão** que mostre o nome das disciplinas e seus respectivos professores. Consulte a **visão** criada.
- Crie uma **visão** que mostre o nome dos professores e a quantidade de disciplinas por ele ministradas. Consulte a **visão** criada.

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](#)