

# Conceitos de Banco de Dados

## Aula 07 - Linguagem SQL - Subconsultas e Vis es

# Apresentação

---

Na aula anterior, começamos nossos estudos sobre ambientes de banco de dados com múltiplas tabelas, através da especificação de atributos como sendo chave primária e chave estrangeira (PRIMARY KEY e FOREIGN KEY). Em seguida, estudamos o processo de consulta no contexto multitabelas, usando as conexões cartesianas definidas pela cláusula CROSS JOIN.

Nesta aula, você vai aprender como pegar um resultado de uma consulta e usá-lo como entrada para outra consulta, ou seja, irá trabalhar com consultas aninhadas, denominadas subconsultas. A utilização de subconsultas permite realizar consultas mais dinâmicas e evitar dados duplicados.

Além disso, você vai aprender 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.

## Objetivos

- Consultar dados em tabelas usando subconsultas.
- Diferenciar subconsultas e conexões.
- Realizar subconsultas com uma coluna na instrução SELECT.
- Diferenciar subconsultas correlacionadas e não correlacionadas.
- Realizar subconsultas usando as cláusulas EXISTS e NOT EXISTS.
- Criar visões e visualizar os dados contidos em uma visão.
- Manipular os dados de uma tabela por meio de visões associadas a elas.

# Subconsulta

---

Para você realizar os estudos desta aula, vamos considerar um banco de dados, denominado **sistvendas**, que representa um sistema de controle de vendas de uma determinada loja de departamentos com as seguintes tabelas:

- produtos (codigo\_Produto [chave primária], nome, marca e preço);
- clientes (codigo\_Clientes [chave primária], nome, CPF, sexo e dataNascimento);
- compras (codigo\_Compras [chave primária], codigo\_Produto [chave estrangeira], codigo\_Clientes [chave estrangeira] e total\_de\_unidades).

As estruturas e os dados das tabelas **produtos**, **clientes** e **compras** são ilustradas na **Figura 1** e na **Figura 2**. Analise com cuidado essas tabelas e não se esqueça de implementá-las em seu SGBD e inserir dados nelas para posterior utilização, essa é uma ótima maneira de fixar os conceitos aprendidos.

Figura 01 - Tela do MySQL após os comandos DESC produtos, DESC clientes e DESC compras.

```
MySQL 5.7 Command Line Client
mysql> USE sistvendas;
Database changed
mysql> DESC produtos;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| prod_codigo | int(11)       | NO   | PRI | NULL    |      |
| prod_nome   | varchar(100)  | NO   |     | NULL    |      |
| prod_marca  | varchar(40)   | NO   |     | NULL    |      |
| prod_preco  | decimal(9,2)  | NO   |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.02 sec)

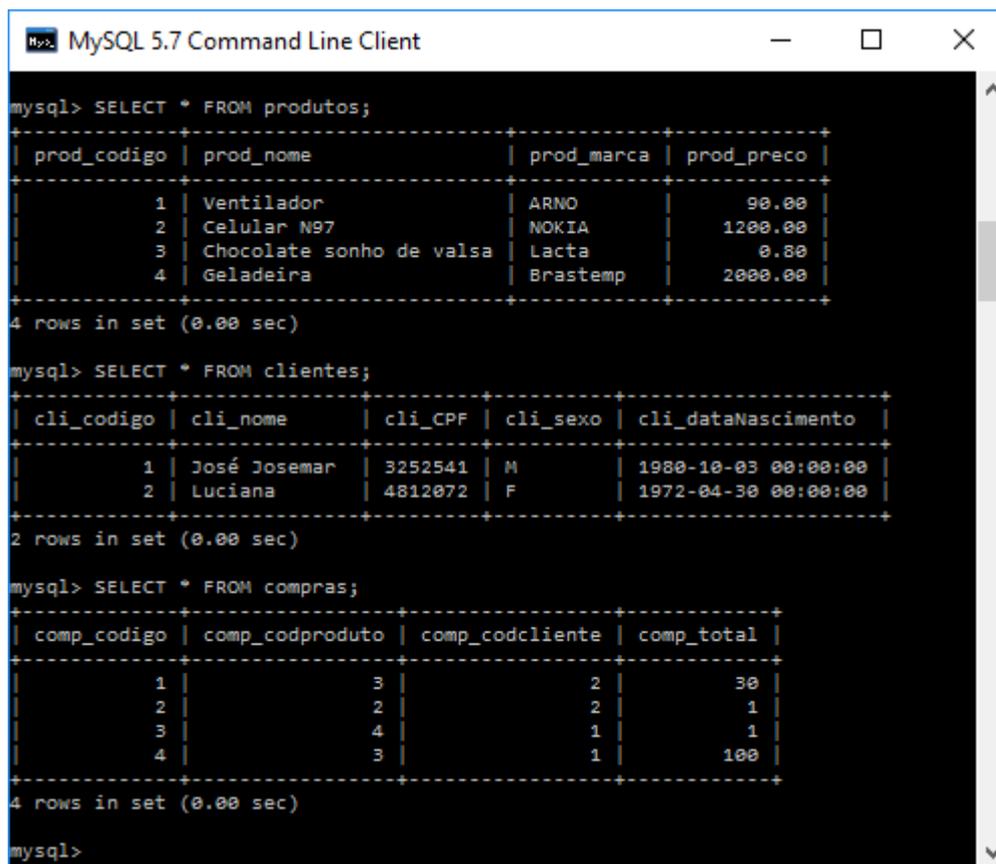
mysql> DESC clientes;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cli_codigo     | int(11)       | NO   | PRI | NULL    |      |
| cli_nome       | varchar(50)   | NO   |     | NULL    |      |
| cli_CPF        | varchar(11)   | NO   |     | NULL    |      |
| cli_sexo       | char(1)       | NO   |     | NULL    |      |
| cli_dataNascimento | datetime      | NO   |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> DESC compras;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| comp_codigo    | int(11)       | NO   | PRI | NULL    |      |
| comp_codproduto | int(11)       | NO   | MUL | NULL    |      |
| comp_codcliente | int(11)       | NO   | MUL | NULL    |      |
| comp_total     | int(11)       | NO   |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

Fonte: MySQL Server 5.7 Command Line Client

**Figura 02** - Tela do MySQL mostrando os registros presentes nas tabelas **produtos**, **clientes** e **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>
```

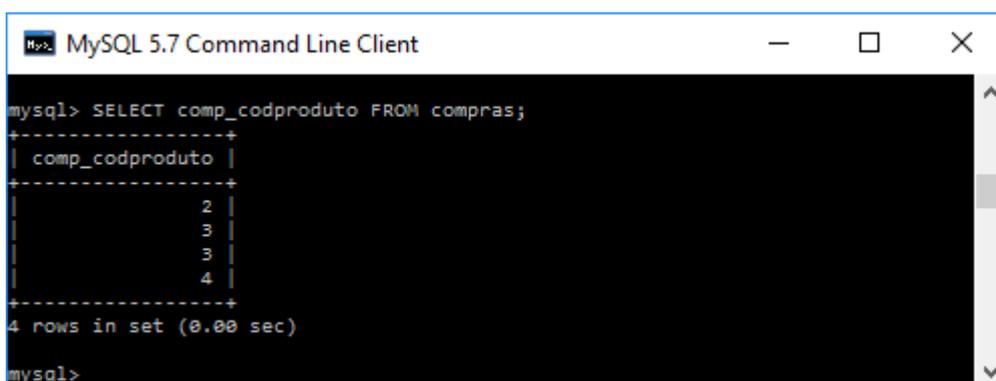
**Fonte:** MySQL Server 5.7 Command Line Client

Suponha que precisamos saber quais são os códigos dos produtos que foram vendidos. O comando utilizado, conforme visto em aulas anteriores, para realizar essa simples pesquisa é mostrado a seguir.

```
1 mysql> SELECT comp_codproduto
2   FROM compras;
```

A resposta do sistema SGBD a essa pesquisa é 2, 3, 3 e 4, conforme é ilustrado na **Figura 3**.

**Figura 03** - Tela do *MySQL* após o comando `SELECT` para determinar os códigos dos produtos vendidos.



```
mysql> SELECT comp_codproduto FROM compras;
+-----+
| comp_codproduto |
+-----+
|                2 |
|                3 |
|                3 |
|                4 |
+-----+
4 rows in set (0.00 sec)

mysql>
```

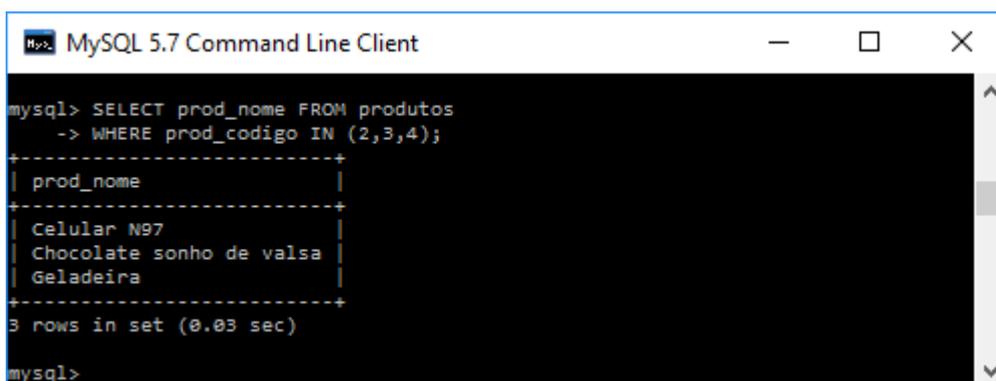
**Fonte:** *MySQL Server 5.7 Command Line Client*

Esses são os códigos dos produtos que foram vendidos, mas essa resposta não fornece muita informação acerca de quais são esses produtos. Então, uma nova pesquisa é necessária para identificar quais são os nomes dos produtos que possuem código 2, 3 e 4. Para tanto, podemos utilizar o seguinte comando.

```
1 mysql> SELECT prod_nome
2   FROM produtos
3   WHERE prod_codigo IN (2,3,4);
```

A resposta do sistema SGBD a essa pesquisa é ilustrada na **Figura 4**.

**Figura 04** - Tela do *MySQL* após o comando `SELECT` para determinar o nome dos produtos vendidos.



```
mysql> SELECT prod_nome FROM produtos
-> WHERE prod_codigo IN (2,3,4);
+-----+
| prod_nome |
+-----+
| Celular N97 |
| Chocolate sonho de valsa |
| Geladeira |
+-----+
3 rows in set (0.03 sec)

mysql>
```

**Fonte:** *MySQL Server 5.7 Command Line Client*

Será que não é possível combinar essas duas consultas para que não se tenha de escrever e solicitar duas consultas separadas ao banco de dados? A resposta a essa pergunta é a **subconsulta**, o assunto que estudaremos agora.

Subconsulta é uma instrução SELECT na forma (SELECT... FROM... WHERE...) adicionada dentro de outra instrução SELECT. Podendo também ser utilizada nas instruções INSERT, DELETE e UPDATE como parâmetro da cláusula WHERE.

Vamos combinar as duas consultas realizadas anteriormente ao banco de dados **sistvendas** em única consulta usando uma subconsulta? A consulta que retorna os códigos dos produtos que foram vendidos será a consulta interna ou subconsulta. E a consulta que identifica os nomes dos produtos vendidos será a consulta externa. Para ver como isso funciona, analise o seguinte comando, que realiza uma consulta com a subconsulta, descrito no quadro a seguir.

```
1 mysql> SELECT prod_nome
2   FROM produtos
3   WHERE prod_codigo IN (SELECT comp_codproduto FROM compras);
```

O que a instrução SELECT mais interna faz é retornar os códigos dos produtos que já foram vendidos. Esses códigos são utilizados pela expressão da consulta mais externa para filtrar os nomes dos produtos que devem ser visualizados.

Observe no exemplo anterior alguns pontos importantes. Em primeiro lugar, a subconsulta está entre parênteses "()". Sempre devemos colocar uma subconsulta entre parênteses, pois é assim que o SGBD consegue fazer sua identificação. Segundo, a subconsulta não tem seu próprio sinal de ponto e vírgula. O sinal de ponto e vírgula é utilizado somente ao final de uma consulta completa (consulta externa + consulta interna). Terceiro, a subconsulta em questão está retornando uma lista de valores. Por isso, utilizamos o operador IN, que é um operador de agregação que indica que o sistema está procurando um conjunto de valores, conforme foi visto em aulas anteriores. Se tivéssemos utilizado na cláusula WHERE um operador de comparação, como o sinal de igualdade (=), a subconsulta deve retornar apenas um valor simples.

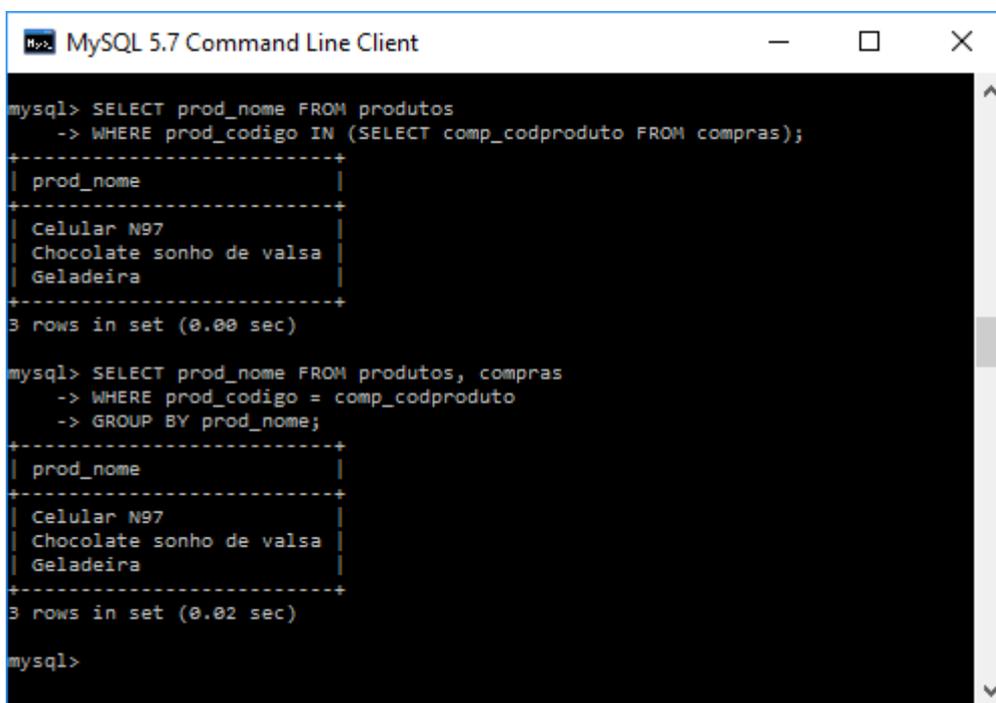
Uma pergunta que pode surgir nesse momento é se a consulta feita não poderia ser realizada utilizando conexões. A verdade é que uma consulta contendo uma subconsulta não é o único jeito de realizar a pesquisa em questão. Poderíamos ter

realizado a mesma consulta utilizando conexão cruzada, conforme é descrito no quadro a seguir.

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

O termo GROUP BY agrupa as linhas com valores iguais de uma determinada coluna, conforme apresentado anteriormente. Observe na **Figura 5** que ambas as consultas produzem o mesmo resultado impresso na tela.

**Figura 05** - Tela do MySQL após diversos comandos SELECT para listar o nome dos produtos vendidos.



```
mysql> SELECT prod_nome FROM produtos
  -> WHERE prod_codigo IN (SELECT comp_codproduto FROM compras);
+-----+
| prod_nome |
+-----+
| Celular N97 |
| Chocolate sonho de valsa |
| Geladeira |
+-----+
3 rows in set (0.00 sec)

mysql> SELECT prod_nome FROM produtos, compras
  -> WHERE prod_codigo = comp_codproduto
  -> GROUP BY prod_nome;
+-----+
| prod_nome |
+-----+
| Celular N97 |
| Chocolate sonho de valsa |
| Geladeira |
+-----+
3 rows in set (0.02 sec)

mysql>
```

Fonte: MySQL Server 5.7 Command Line Client

## Subconsultas versus Conexões

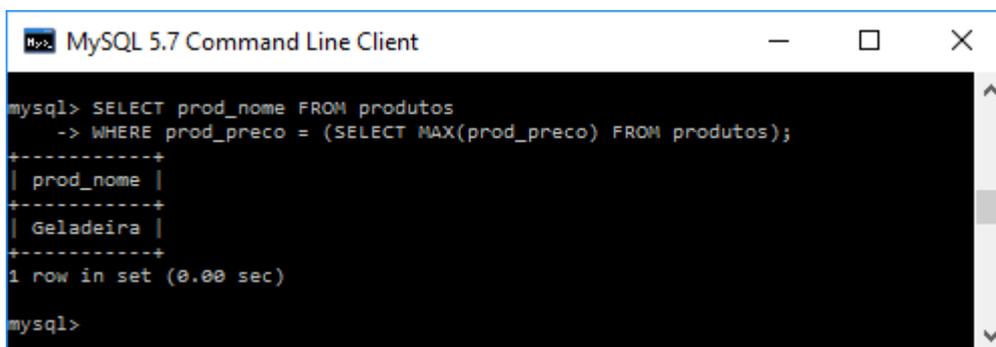
Visto que é possível realizar uma pesquisa, como, por exemplo, o nome dos produtos que já foram vendidos, usando uma consulta com subconsulta ou com conexão, você saberia dizer qual é a melhor estratégia?

De uma maneira geral, consultas usando conexões são mais eficientes que consultas com subconsultas, pois o *Query Optimizer* (mecanismo interno do SGBD que otimiza as instruções enviadas para o banco de dados) pode executar algumas operações a mais quando se utiliza uma subconsulta degradando assim a performance da execução. Ou seja, uma consulta com subconsulta pode ser mais demorada do que uma consulta com conexão. Entretanto, as consultas com subconsultas são apropriadas para pesquisas que envolvam comparações com agregações, como a consulta a seguir:

```
1 mysql> SELECT prod_nome
2   FROM produtos
3   WHERE prod_preco = (SELECT MAX(prod_preco) FROM produtos);
```

Nessa consulta, a instrução SELECT mais interna retorna o preço máximo dos produtos da tabela PRODUTOS, através da função de agregação MAX(). A seguir, esse valor é comparado com todos os produtos e somente aqueles que possuírem o **prod\_preco** com valor máximo serão retornados pela consulta mais externa. A resposta do SGBD a essa consulta é ilustrada na **Figura 6**.

**Figura 06** - Tela do MySQL após o comando SELECT para listar os produtos com preço máximo.



```
MySQL 5.7 Command Line Client
mysql> SELECT prod_nome FROM produtos
-> WHERE prod_preco = (SELECT MAX(prod_preco) FROM produtos);
+-----+
| prod_nome |
+-----+
| Geladeira |
+-----+
1 row in set (0.00 sec)

mysql>
```

**Fonte:** MySQL Server 5.7 Command Line Client



**Vídeo 01** - Subconsultas

## Subconsultas como uma coluna na instrução SELECT

---

Podemos também utilizar o resultado de uma subconsulta no lugar de uma coluna que será retornada na instrução SELECT, conforme é descrito a seguir.

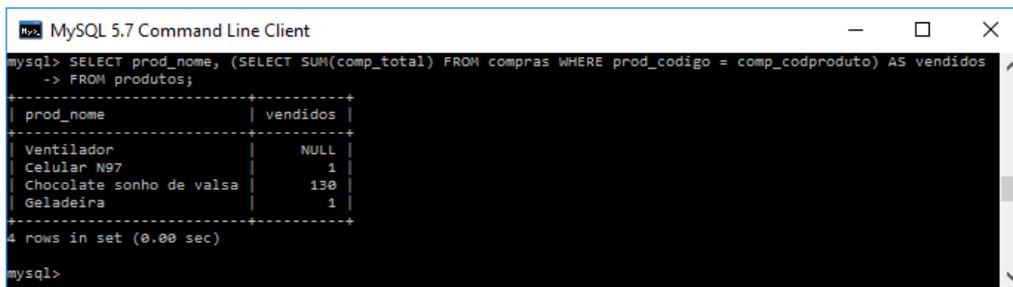
```
1 mysql> SELECT tabela1.atributo1,  
2   (SELECT tabela2.atributoX FROM tabela2 WHERE condição), tabela1.atributo2 ...  
3   FROM tabela1;
```

Observe que os atributos *tabela1.atributo1* e *tabela1.atributo2* são atributos pertencentes à tabela 1 que se deseja visualizar. Além desses atributos, deseja-se visualizar o resultado da subconsulta que retorna um determinado atributo (*tabela2.atributoX*) da tabela 2. É importante destacar que mesmo que a tabela 2 possua mais de uma coluna (atributo), somente pode-se retornar uma única coluna. Para um melhor entendimento acerca desse tópico, examine o exemplo a seguir que lista todos os nomes dos produtos e quantidade de itens vendidos.

```
1 mysql> SELECT prod_nome,  
2   (SELECT SUM(comp_total) FROM compras WHERE prod_codigo= comp_codproduto) AS vendidos  
3   FROM produtos;
```

Em termos simples, a consulta feita está exibindo em cada linha o nome de um produto e a quantidade de itens vendidos desse produto. Para ser executada, a consulta interna necessita da informação do atributo **prod\_codigo**, porém, essa informação pertence à tabela **produtos**, que é acessada a partir da consulta externa. O termo AS é utilizado no SELECT externo para nomear o resultado da subconsulta como sendo vendidos com a finalidade de deixar a consulta mais legível. A função SUM retorna a soma dos itens vendidos de um determinado produto, conforme pode ser verificado na **Figura 7**. Lembre-se de que a subconsulta nesse caso deve retornar um valor único. Então, cada vez que ela é executada, uma única linha é retornada.

**Figura 07** - Tela do MySQL após o comando SELECT para listar a quantidade total de produtos vendidos.



```
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>
```

Fonte: MySQL Server 5.7 Command Line Client

## Atividade 01

---

1. Vamos praticar um pouco para que você se familiarize com o comando de consulta com subconsulta. Entre no banco de dados do sistvendas. Elabore as seguintes consultas utilizando subconsultas.
  - a. Nome dos clientes que efetuaram compras.
  - b. Nome do cliente com código 1 e suas compras.
  - c. Nome do cliente com código 2 e suas compras.
  - d. Nome de todos os clientes e suas respectivas compras.
2. Elabore uma consulta ao banco de dados do sistvendas usando subconsulta e faça o mesmo usando conexões. Em sua opinião, qual é a mais fácil de implementar? Justifique.

## Subconsulta correlacionada e subconsulta não correlacionada

---

No exemplo anterior, que lista todos os nomes dos produtos e quantidade de itens vendidos, vimos que a consulta interna necessita da informação sobre o atributo **prod\_codigo** da tabela **produtos** para ser executada, que está disponível para acesso a partir da consulta externa. Sendo assim, a subconsulta não pode ser

executada como uma consulta independente. A consulta externa tem que ser executada antes para sabermos qual é o valor de **prod\_codigo**. Quando isso ocorre, dizemos que a subconsulta é correlacionada.

Subconsulta correlacionada é quando a consulta interna depende dos valores retornados pela consulta externa para ser processada. Na subconsulta não correlacionada, a consulta interna funciona sozinha, não necessitando de nenhuma informação da consulta externa, podendo ser executada como uma consulta independente.

Para entendermos melhor a diferença entre subconsulta correlacionada e subconsulta não correlacionada, vamos analisar os seguintes exemplos. Examine com cuidado e não deixe que nenhuma dúvida fique sem ser esclarecida.

## 1º exemplo

### Subconsulta correlacionada

Pesquisar o nome e o total de produtos comprados por cada cliente.

```
1 mysql> SELECT cli_nome,  
2   (SELECT SUM(comp_total)  
3   FROM compras  
4   WHERE cli_codigo= comp_codcliente) AS itenscomprados  
5   FROM clientes;
```

Nesse exemplo, a subconsulta necessita, para ser executada, da informação de qual é o código do cliente, que será informada pela consulta externa a cada linha que a consulta externa processe. O resultado dessa pesquisa é ilustrado na **Figura 8**.

## 2º exemplo

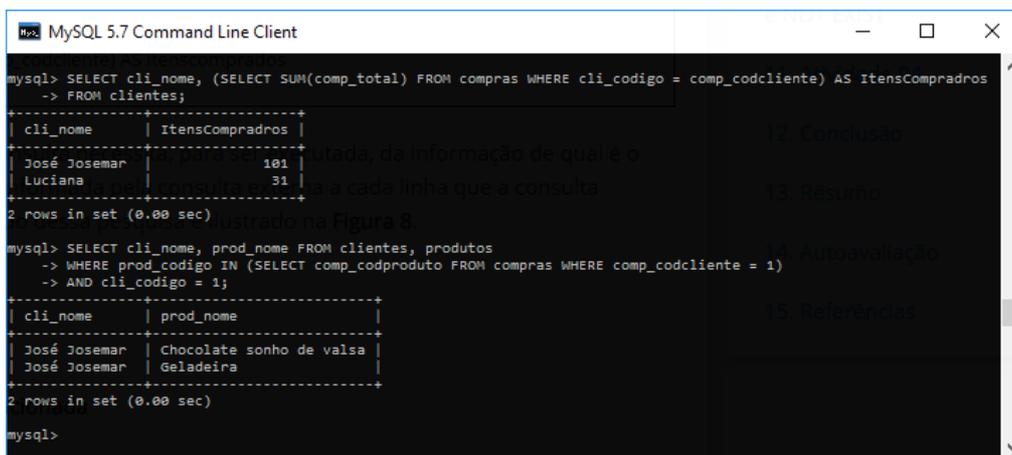
### Subconsulta não correlacionada

Pesquisar o nome do cliente com o atributo **cli\_codigo=1** e os produtos por ele comprados.

```
1 mysql> SELECT cli_nome, prod_nome
2   FROM clientes, produtos
3   WHERE prod_codigo IN (SELECT comp_codproduto FROM compras
4   WHERE comp_codcliente=1) and cli_codigo=1;
```

Nesse exemplo, tem-se uma junção de conexões e subconsulta não correlacionada em uma única consulta. A subconsulta não depende de nenhuma informação da consulta externa para ser executada, trabalhando apenas com as informações provenientes da tabela **compras**, sendo, portanto, uma subconsulta não correlacionada. A consulta externa realiza uma pesquisa empregando conexão cartesiana. O resultado dessa pesquisa é ilustrado na **Figura 8**. Observe que as linhas listadas correspondem ao produto cartesiano entre as tabelas clientes e produtos que possuem o código do produto pertencente ao conjunto retornado pela subconsulta.

**Figura 08** - Tela do MySQL após os comandos SELECTs dos exemplos 1 e 2.



```
mysql> SELECT cli_nome, (SELECT SUM(comp_total) FROM compras WHERE cli_codigo = comp_codcliente) AS ItensComprados
-> FROM clientes;
+-----+-----+
| cli_nome | ItensComprados |
+-----+-----+
| José Josemar | 101 |
| Luciana | 31 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT cli_nome, prod_nome FROM clientes, produtos
-> WHERE prod_codigo IN (SELECT comp_codproduto FROM compras WHERE comp_codcliente = 1)
-> AND cli_codigo = 1;
+-----+-----+
| cli_nome | prod_nome |
+-----+-----+
| José Josemar | Chocolate sonho de valsa |
| José Josemar | Geladeira |
+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Fonte: MySQL Server 5.7 Command Line Client



**Vídeo 02** - Subconsultas como Coluna

## Subconsulta com EXIST e NOT EXIST

---

Uma das grandes aplicações da subconsulta correlacionada é a realização de testes de existência e inexistência, que são realizados com as cláusulas EXISTS e NOT EXISTS, respectivamente, juntamente com a subconsulta.

A realização de um teste de inexistência permite encontrar todas as linhas de uma tabela (referenciada na consulta externa), que não contém os dados fornecidos pela subconsulta.

Vamos supor que precisamos determinar quais foram os produtos que não foram vendidos no nosso banco de dados **sistvendas**. Uma forma de realizar essa pesquisa é utilizar um NOT EXISTS para encontrar os produtos.

```
1 mysql> SELECT prod_nome
2   FROM produtos
3   WHERE NOT EXISTS (SELECT * FROM compras
4   WHERE prod_codigo= comp_codproduto);
```

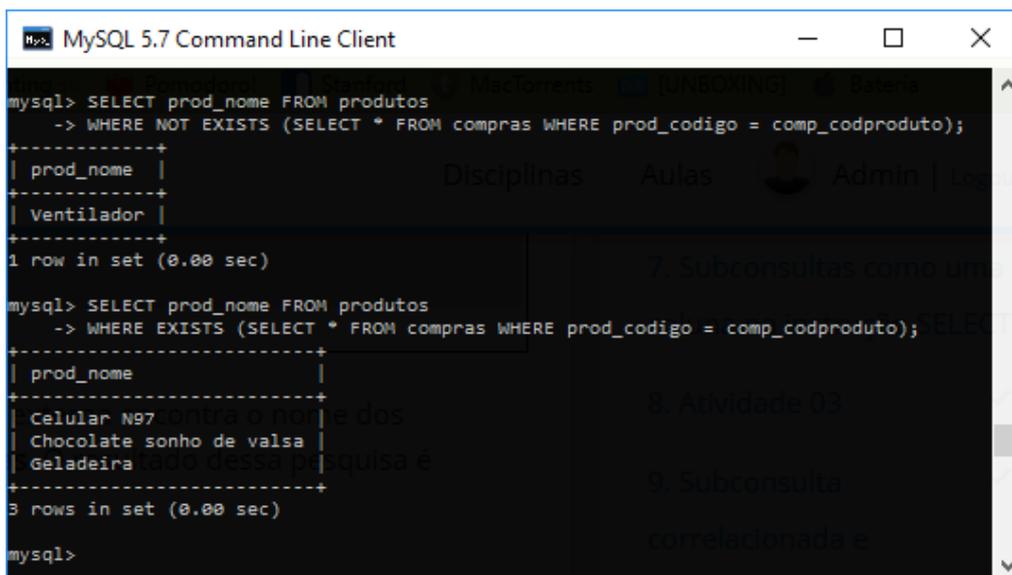
Com a utilização da cláusula NOT EXISTS, a consulta externa encontra o nome dos produtos que ainda não estão listados na tabela **compras**. O resultado dessa pesquisa é ilustrado na **Figura 9**.

De maneira semelhante, podemos listar os nomes dos produtos que foram vendidos utilizando o termo EXISTS.

```
1 mysql> SELECT prod_nome
2   FROM produtos
3   WHERE EXISTS (SELECT * FROM compras
4   WHERE prod_codigo= comp_codproduto);
```

A consulta feita retorna os nomes dos produtos em que o atributo **prod\_codigo** aparece pelo menos uma vez na tabela **compras**. O resultado dessa pesquisa encontra-se na **Figura 9**.

**Figura 09** - Tela do MySQL após os testes de inexistência e existência .



```
mysql> SELECT prod_nome FROM produtos
-> WHERE NOT EXISTS (SELECT * FROM compras WHERE prod_codigo = comp_codproduto);
+-----+
| prod_nome |
+-----+
| Ventilador |
+-----+
1 row in set (0.00 sec)

mysql> SELECT prod_nome FROM produtos
-> WHERE EXISTS (SELECT * FROM compras WHERE prod_codigo = comp_codproduto);
+-----+
| prod_nome |
+-----+
| Celular N97 |
| Chocolate sonho de valsa |
| Geladeira |
+-----+
3 rows in set (0.00 sec)

mysql>
```

**Fonte:** MySQL Server 5.7 Command Line Client



**Vídeo 03** - EXIST e NOT EXIST

## Atividade 02

1. Seu SGBD avisa a você quando algo está errado no seu código, mas às vezes a resposta é um tanto vaga. Examine, a seguir, os comandos SELECT e tente descobrir o que há de errado com ele. Em seguida, digite no seu sistema e observe a mensagem exibida.
  - a. SELECT prod\_nome, (SELECT cli\_nome FROM clientes) FROM produtos;
  - b. SELECT prod\_nome, (SELECT cli\_nome, cli\_cpf FROM clientes WHERE cli\_codigo=1) AS cliente1 FROM produtos;
  - c. SELECT prod\_nome, (SELECT cli\_nome, FROM clientes WHERE cli\_codigo=1;) AS cliente1 FROM produtos;

2. Acesse o banco de dados do **cineOnline**. Elabore as seguintes consultas utilizando subconsultas com teste de existência e inexistência, conforme for o caso.

a. Nome dos filmes que tiveram ingressos vendidos.

b. Nome dos clientes que não compraram ingressos

## VIEWS

---

**Visões** (*views*) 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.

```
1 mysql> CREATE VIEW nome_da_visão AS
2   SELECT atributo1, atributo2, ...
3   FROM nome_da_tabela1, nome_da_tabela2, ...
4   WHERE condição;
```

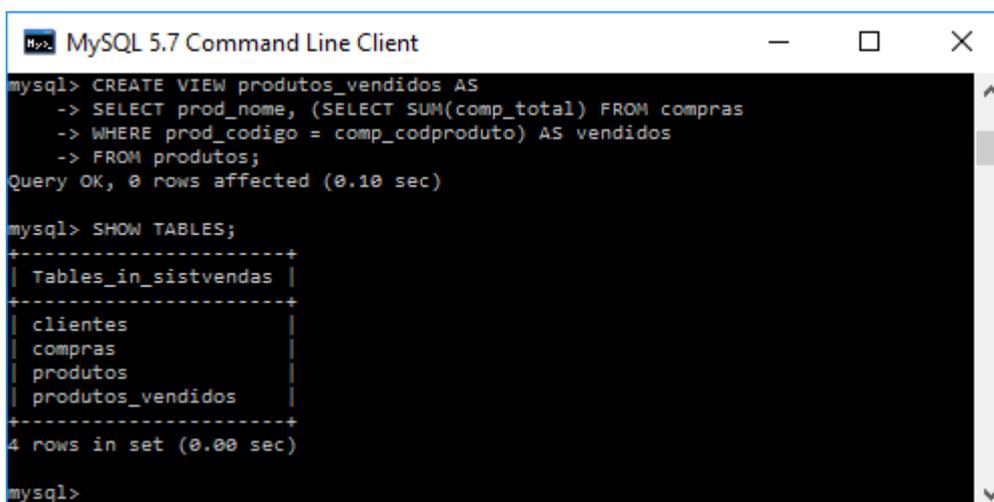
Nessa expressão, no campo *nome\_da\_visão* deve-se inserir o nome que se deseja atribuir para a **visão**, nome esse que deve seguir as mesmas regras usadas para os nomes das tabelas. Após a cláusula AS, tem-se qualquer comando SELECT válido.

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 nesse 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 a seguir.

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

A resposta do sistema SGBD para o referido comando é ilustrada na **Figura 10**. 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**, que exibe como resposta o nome das tabelas contidas no banco de dados em questão.

**Figura 10** - Tela do MySQL após os comandos CREATE VIEW e SHOW TABLES.



```
MySQL 5.7 Command Line Client
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 (0.10 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_sistvendas |
+-----+
| clientes              |
| compras               |
| produtos              |
| produtos_vendidos    |
+-----+
4 rows in set (0.00 sec)

mysql>
```

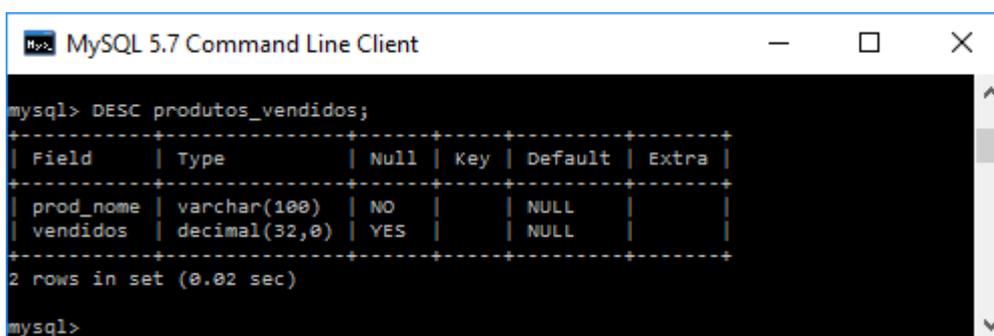
Fonte: MySQL 5.7 Command Line Client

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.

```
1 | mysql> DESC nome_da_visão;
```

A resposta do SGBD ao comando DESC produtos\_vendidos é ilustrada na **Figura 11**.

**Figura 11** - Tela do MySQL após a visualização da estrutura da **visão** usando o comando DESC.



```
mysql> DESC produtos_vendidos;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| prod_nome  | varchar(100)  | NO   |     | NULL    |       |
| vendidos  | decimal(32,0) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)

mysql>
```

Fonte: MySQL 5.7 Command Line Client

Observe que a tabela virtual **produtos\_vendidos** contém dois campos: **prod\_nome** e **vendidos**. O campo **vendidos** é o resultado da seguinte subconsulta:

```
1 | (SELECT SUM(comp_total) FROM compras
2 | 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.

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 a seguir.

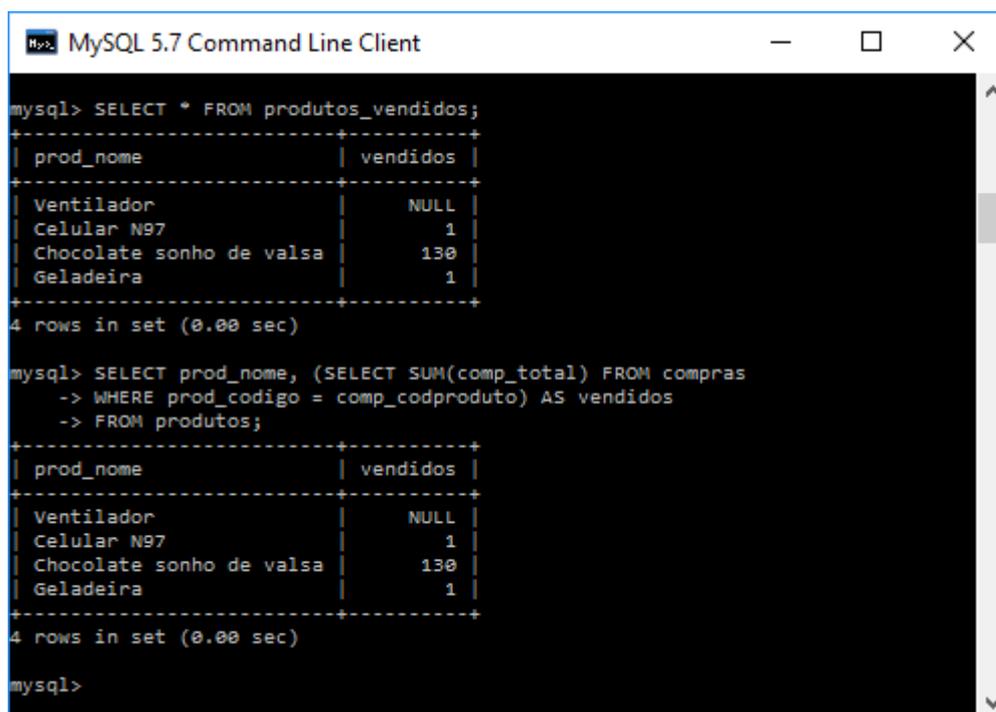
```
1 mysql> SELECT * FROM nome_da_visão;
```

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

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

A resposta do sistema ao comando acima é ilustrada na **Figura 12**.

**Figura 12** - Tela do MySQL após os comandos SELECTs para visualização da quantidade total de produtos vendidos.



```
mysql> SELECT * FROM produtos_vendidos;
+-----+-----+
| prod_nome | vendidos |
+-----+-----+
| Ventilador | NULL     |
| Celular N97 | 1       |
| Chocolate sonho de valsa | 130    |
| Geladeira  | 1       |
+-----+-----+
4 rows in set (0.00 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>
```

**Fonte:** MySQL 5.7 Command Line Client

Observe que a resposta do sistema é exatamente a mesma dada pelo comando da consulta em si. 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.

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 em uma **visão** estar 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.

```
1 | 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 a seguir.

```
1 | mysql> DROP VIEW nome_da_visão;
```



**Vídeo 04** - Introdução a Visões

## 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.

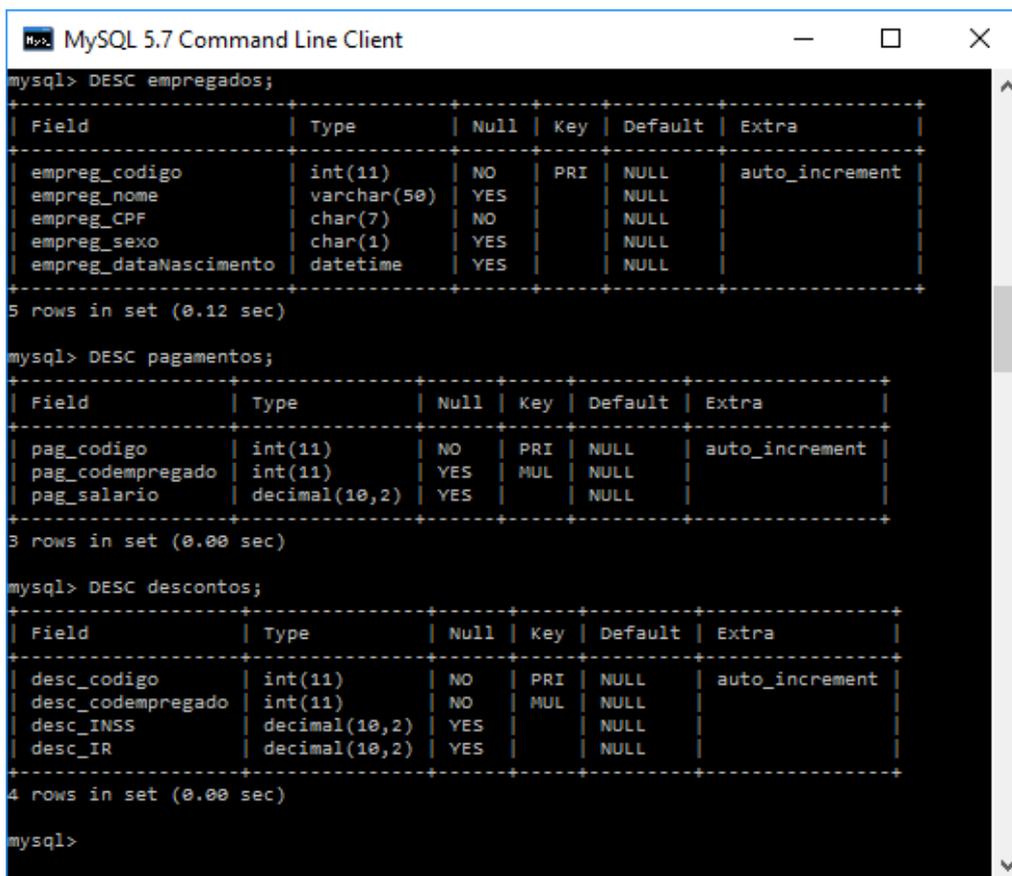
Para continuarmos os nossos estudos, considere um banco de dados, denominado **sispagamentos**, que representa um sistema de pagamentos dos funcionários de uma determinada empresa contendo as seguintes tabelas:

- 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 13**. 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.

Perceba, conforme **Figura 13**, que as chaves primárias estão com informação de campo extra chamada `auto_increment`. O `auto_increment` é uma opção que pode ser adicionada a um atributo para que o banco de dados MySQL gere automaticamente valores únicos para esta coluna. Sendo assim, com essa opção habilitada, não é necessário informar o valor da chave primária no comando de `INSERT` para essas tabelas. Além disso, o próprio banco garante que os valores para esse campo serão únicos.

**Figura 13** - Tela do MySQL após os comandos de visualização das estruturas das tabelas do banco de dados sispagamentos.



```
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_dataNascimento | datetime     | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.12 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 (0.00 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 (0.00 sec)

mysql>
```

Fonte: MySQL 5.7 Command Line Client

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!

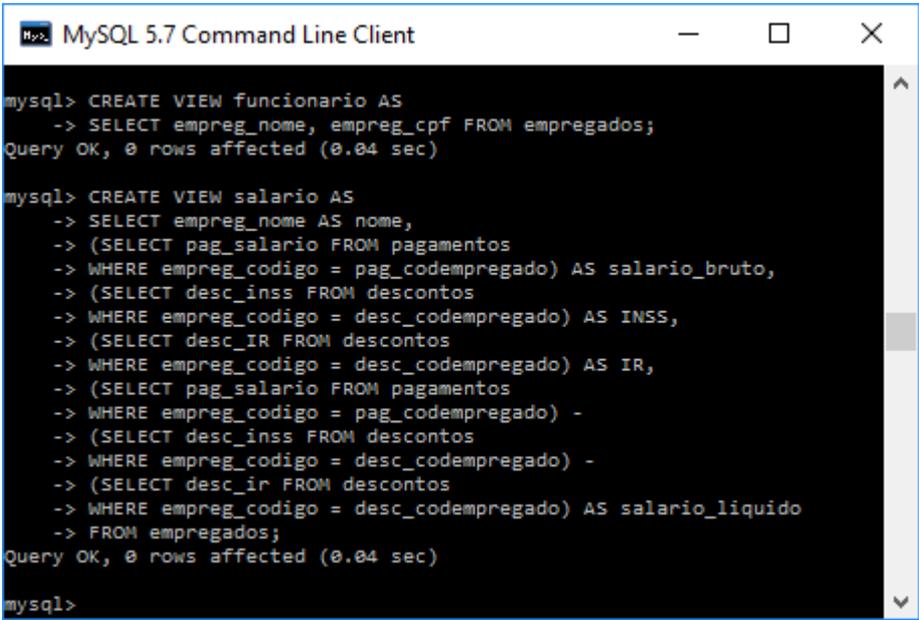
```

1 mysql> CREATE VIEW funcionario AS
2   SELECT empreg_nome, empreg_cpf
3   FROM empregados;
4
5 mysql> CREATE VIEW salario AS
6   SELECT empreg_nome AS nome,
7   (SELECT pag_salario FROM pagamentos
8   WHERE empreg_codigo = pag_codempregado) AS salario_bruto,
9   (SELECT desc_inss FROM descontos
10  WHERE empreg_codigo = desc_codempregado) AS INSS,
11  (SELECT desc_ir FROM descontos
12  WHERE empreg_codigo = desc_codempregado) AS IR,
13  (SELECT pag_salario FROM pagamentos
14  WHERE empreg_codigo = pag_codempregado) -
15  (SELECT desc_inss FROM descontos
16  WHERE empreg_codigo = desc_codempregado) -
17  (SELECT desc_ir FROM descontos
18  WHERE empreg_codigo = desc_codempregado)
19  AS salario_liquido FROM empregados;

```

A criação das **visões funcionario** e **salario** está ilustrada na **Figura 14** e suas respectivas estruturas estão na **Figura 15**.

**Figura 14** - Tela do *MySQL* após os comandos de criação das **visões funcionario** e **salário**.



```

MySQL 5.7 Command Line Client
mysql> CREATE VIEW funcionario AS
-> SELECT empreg_nome, empreg_cpf FROM empregados;
Query OK, 0 rows affected (0.04 sec)

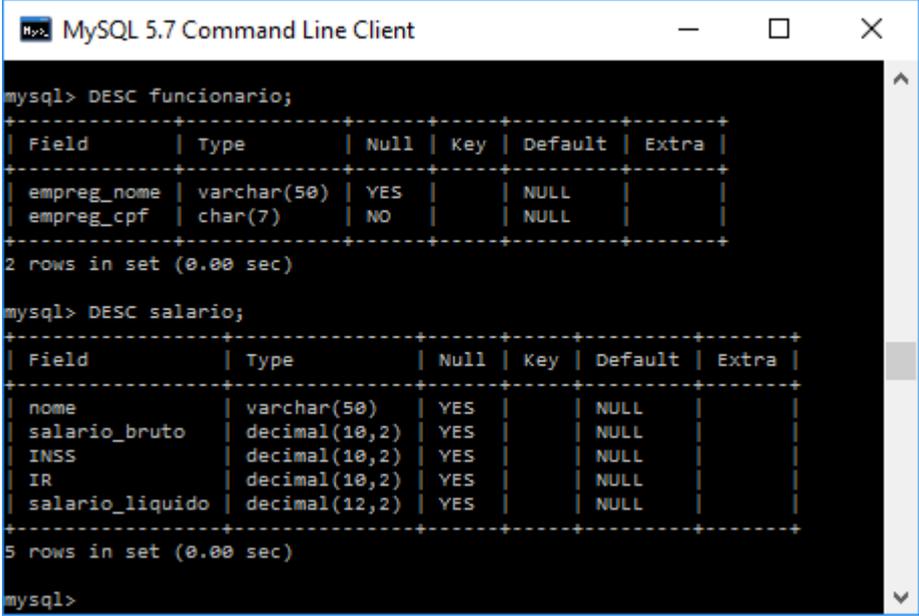
mysql> CREATE VIEW salario AS
-> SELECT empreg_nome AS nome,
-> (SELECT pag_salario FROM pagamentos
-> WHERE empreg_codigo = pag_codempregado) AS salario_bruto,
-> (SELECT desc_inss FROM descontos
-> WHERE empreg_codigo = desc_codempregado) AS INSS,
-> (SELECT desc_IR FROM descontos
-> WHERE empreg_codigo = 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 (0.04 sec)

mysql>

```

Fonte: *MySQL 5.7 Command Line Client*

Figura 15 - Tela do MySQL após os comandos DESC **funcionario** e DESC **salário**.



```
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 (0.00 sec)

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 (0.00 sec)

mysql>
```

Fonte: MySQL 5.7 Command Line Client

## Exemplos pt1

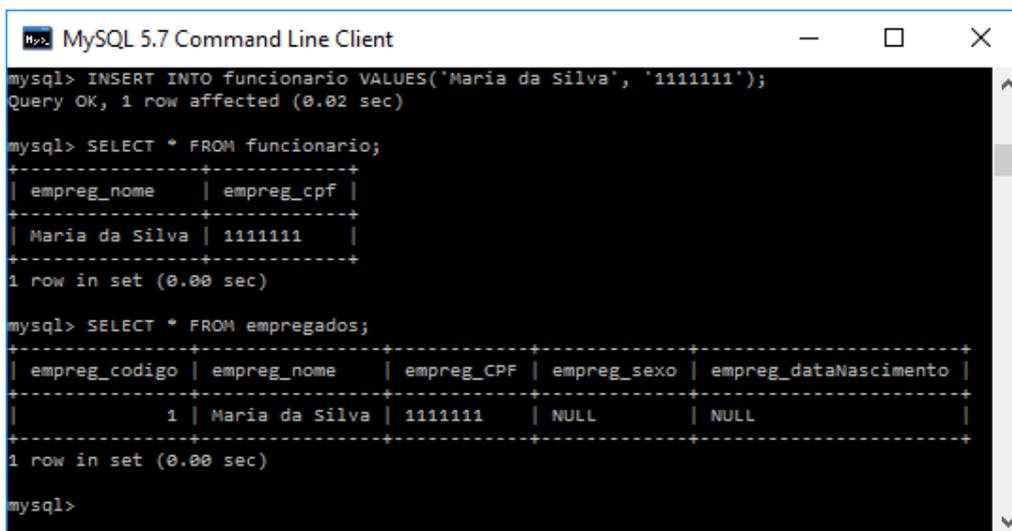
Agora, vamos analisar os exemplos a seguir para entendermos como funciona os comandos INSERT, UPDATE e DELETE com **visões**.

Inserir o nome do funcionário Maria da Silva e o seu CPF 11111111 na tabela **empregados** por meio da visão **funcionario**.

```
1 | mysql>INSERT INTO funcionario VALUES ('Maria da Silva','11111111');
```

A resposta do SGBD, no caso, o MySQL, ao comando apresentado é ilustrada na **Figura 16**. Visando uma melhor compreensão do que foi realizado pelo sistema ao processar o referido comando, são exibidos, também, na **Figura 16**, os dados existentes na **visao funcionario** e na tabela **empregados** após o referido comando de inclusão (INSERT).

**Figura 16** - Tela do MySQL após os comandos INSERT e SELECT.



```
mysql> INSERT INTO funcionario VALUES('Maria da Silva', '1111111');
Query OK, 1 row affected (0.02 sec)

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

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

mysql>
```

**Fonte:** MySQL 5.7 Command Line Client

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 13**) 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.

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 13**). A tabela contém 5 campos, entre os quais 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 que 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.

## Exemplos pt2

---

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

```
1 | 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. 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 17**.

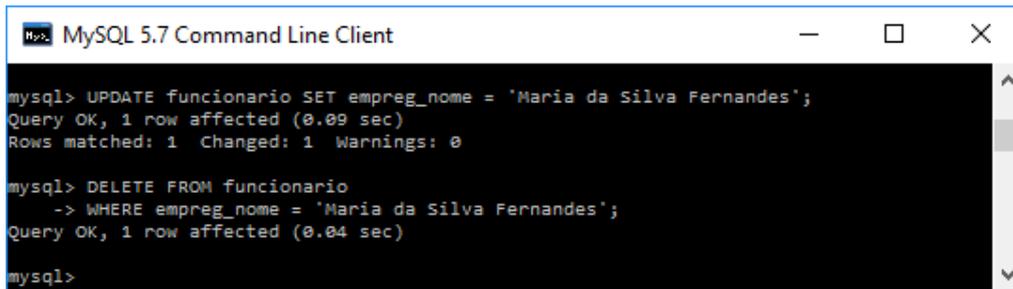
Apague o registro da funcionária “Maria de Silva Fernandes” através da visão funcionario.

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.

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

A resposta do SGBD, no caso o MySQL, ao comando acima é ilustrada na **Figura 17**.

**Figura 17** - Tela do MySQL após os comandos UPDATE e DELETE aplicados a **visão funcionario**.



```
mysql> UPDATE funcionario SET empreg_nome = 'Maria da Silva Fernandes';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> DELETE FROM funcionario
-> WHERE empreg_nome = 'Maria da Silva Fernandes';
Query OK, 1 row affected (0.04 sec)

mysql>
```

Fonte: MySQL 5.7 Command Line Client



**Vídeo 05** - Atualizando Visões

## Atividade 03

1. 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.
  - a. Escreva uma visão que contenha o nome do cliente, CPF, sexo e profissão.
  - b. Escreva um comando que realize uma consulta sobre a visão da questão anterior.
2. Vamos exercitar um pouco os comandos apresentados. Entre no banco de dados de pagamentos. Elabore o que se pede.
  - a. Escreva uma visão que através dela seja possível atualizar os dados das tabelas descontos e pagamentos.
  - b. Insira alguns dados utilizando a visão criada na questão anterior nas tabelas descontos e pagamentos.
  - c. Atualize o desconto de INSS para todas as pessoas diminuindo o seu valor em 10,00 reais.

# Tabelas conectadas com Inner Join

---



**Vídeo 06** - Inner Join

## Conclusão

---

Encerramos por aqui mais uma aula sobre a linguagem SQL. Na próxima aula, você vai aprender a criar procedimentos armazenados (*stored procedures*) e Funções, que são recursos úteis no gerenciamento de sistemas de banco de dados. Bons estudos e boa sorte!

# Autoavaliação

---

1. Qual(ais) a(s) diferença(s) entre subconsultas e conexões?
2. O que você entendeu por subconsultas correlacionadas e não correlacionadas?
3. Altere as seguintes tabelas do banco de dados locadora e insira dados nelas:
  - a. Clientes (codigo [chave primária], nome, cpf, sexo, profissao, salario)
  - b. Filmes (codigo [chave primária], titulo, genero, duracao, ano, situacao, preco)
  - c. Locacoes (codigo, codigo cliente [chave estrangeira], codigo do filme [ chave estrangeira], data)
4. Resolva as seguintes consultas utilizando a linguagem SQL:
  - a. Qual o nome de todos os clientes que já alugaram filmes?
  - b. Qual o título e o gênero de todos os filmes alugados?
  - c. Qual a profissão e o sexo de todos os clientes que alugaram filmes de comédia?
  - d. Qual o gênero dos filmes alugados por estudantes?
  - e. Qual a quantidade de pessoas de cada sexo que alugaram filmes de suspense?
  - f. Qual a média salarial das pessoas que alugaram filmes de aventura?
  - g. Para cada locação do filme E O VENTO LEVOU, liste o nome do cliente.
  - h. Para cada locação, exiba o nome e o CPF do cliente, o título, o gênero, o preço e a data da locação.

i. Para cada locação, exiba o nome do cliente, a data da locação, a quantidade de filmes alugados e o preço total.

5. Considere o banco de dados CursoX criado na autoavaliação cuja estrutura de tabelas é mostrada a seguir:

| <b>ATRIBUTO</b> | <b>TIPO</b>    | <b>DESCRIÇÃO</b>  |
|-----------------|----------------|-------------------|
| 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:** Alunos

| <b>ATRIBUTO</b> | <b>TIPO</b>    | <b>DESCRIÇÃO</b>            |
|-----------------|----------------|-----------------------------|
| 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:** Disciplina

| 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   |

**TABELA:** Professores

- 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.

## 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: 15 jan. 2017.