

# Conceitos de Banco de Dados

## Aula 06 - Linguagem SQL - Alteração de Estrutura de Tabelas e Ambientes de Múltiplas Tabelas

# Apresentação

---

Você já se perguntou como fazer para alterar a estrutura de uma tabela sem precisar excluir e depois recriá-la? A resposta está no comando ALTER, um dos assuntos da aula de hoje, que modifica a estrutura de uma tabela, criando novas colunas ou redefinindo colunas já existentes. Prosseguindo, iremos dar início ao nosso estudo sobre a utilização da linguagem SQL em projetos de banco de dados com múltiplas tabelas. Primeiro, você vai aprender como especificar um atributo como sendo chave primária e chave estrangeira. A seguir, você vai estudar sobre conexões, especialmente, a consulta de dados usando conexões cartesianas CROSS JOIN.



## **Vídeo 01** - Apresentação

### Objetivos

- Modificar a estrutura de uma tabela.
- Criar tabelas com chaves primárias e estrangeiras.
- Consultar dados através de conexões de tabelas.

# Alteração da estrutura de uma tabela

---

Em aulas anteriores, aprendemos a criar, inserir e selecionar dados em tabelas. Mas, como proceder para inserir, por exemplo, uma coluna em uma tabela já existente, no nosso banco de dados? Será que é necessário, para isso, excluir toda a tabela (dados e estrutura)? A resposta para ambas as perguntas é não. Utilizando o comando ALTER, é possível alterar a estrutura de uma tabela, por exemplo, inserindo novas colunas.

Usando o comando ALTER, é possível realizar as seguintes alterações na estrutura de uma tabela:

- Adicionar colunas.
- Excluir colunas.
- Alterar o tipo e o nome de uma coluna já existente.

A sintaxe do comando ALTER para adicionar colunas a uma tabela é descrita no quadro a seguir.

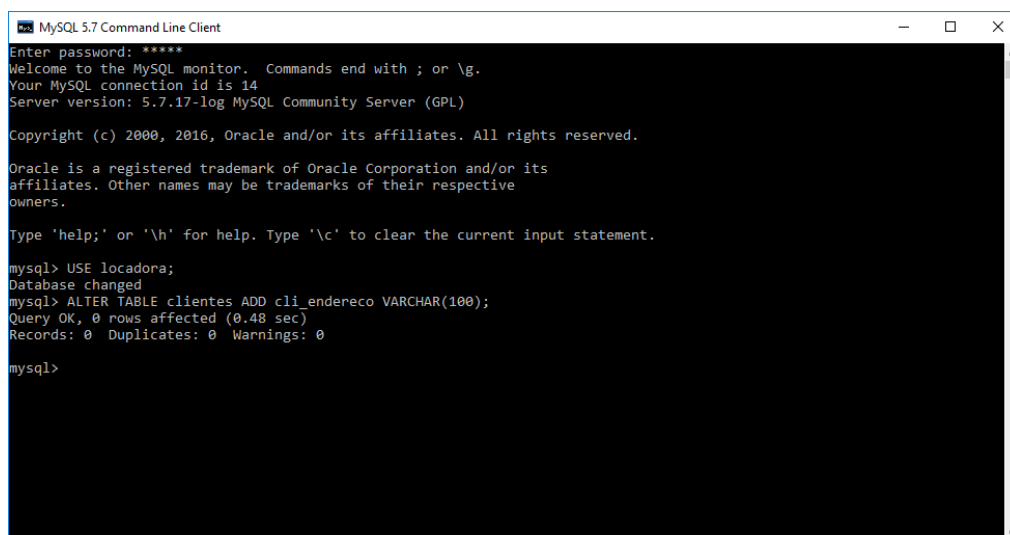
```
1 mysql> ALTER TABLE nome_da_tabela ADD coluna tipo;
```

Observe que no comando ALTER aparece a cláusula ADD indicando que estamos adicionando uma coluna à tabela e definindo o seu tipo. É possível adicionar mais de um campo de uma única vez. Para isso, basta usar uma vírgula para separar os itens a serem inseridos. Examine o exemplo a seguir, no qual adicionamos o campo endereço (*cli\_endereco*) na tabela **clientes** da nossa locadora.

```
1 mysql> ALTER TABLE clientes ADD cli_endereco VARCHAR(100);
```

A resposta do SGBD, no caso o *MySQL*, ao referido comando, é ilustrada na **Figura 1**. A mensagem “*Query OK*” informa que a coluna foi adicionada corretamente.

**Figura 01** - Tela do *MySQL* após o comando `ALTER TABLE clientes ADD cli_endereco VARCHAR(100)`.



```
MySQL 5.7 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

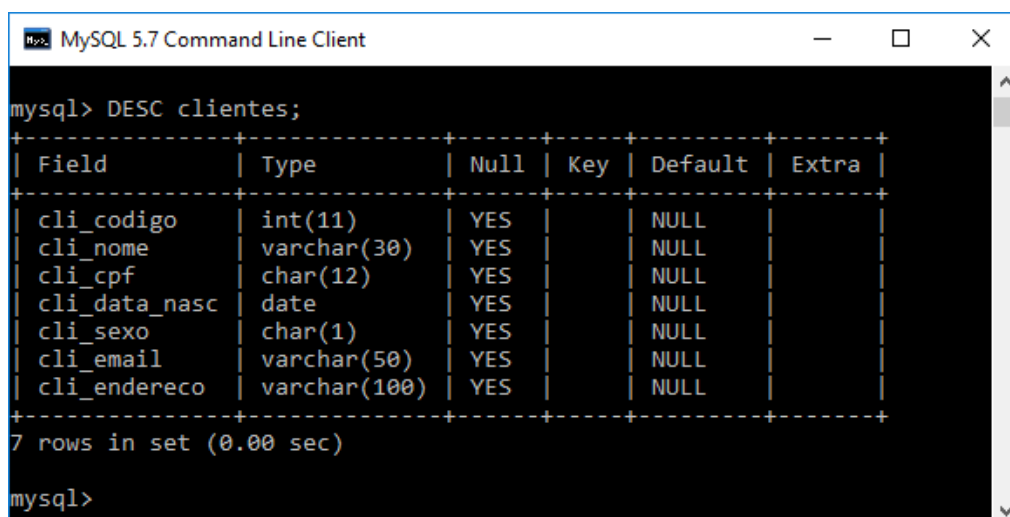
mysql> USE locadora;
Database changed
mysql> ALTER TABLE clientes ADD cli_endereco VARCHAR(100);
Query OK, 0 rows affected (0.48 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

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

Caso você deseje conferir como ficou a estrutura da tabela **clientes**, utilize o comando `DESC`, estudado na Aula 10, conforme visto na **Figura 2**.

**Figura 02** - Tela do *MySQL* mostrando a estrutura da tabela **clientes** após a inserção da coluna `cli_endereco`.



```
MySQL 5.7 Command Line Client

mysql> DESC clientes;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cli_codigo | int(11)       | YES  |     | NULL    |       |
| cli_nome   | varchar(30)   | YES  |     | NULL    |       |
| cli_cpf    | char(12)      | YES  |     | NULL    |       |
| cli_data_nasc | date         | YES  |     | NULL    |       |
| cli_sexo   | char(1)       | YES  |     | NULL    |       |
| cli_email  | varchar(50)   | YES  |     | NULL    |       |
| cli_endereco | varchar(100)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql>
```

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

Observe na **Figura 2** que o campo `cli_endereco` é o último da lista de atributos. Você pode utilizar palavras-chaves como `FIRST` (primeiro), `AFTER` (após), `BEFORE` (antes) e `LAST` (por último) para posicionar a nova coluna na posição que desejar na tabela. Além dessas palavras-chaves, podem ser utilizadas as palavras `SECOND` (segundo), `THIRD` (terceiro) e, assim, por diante.

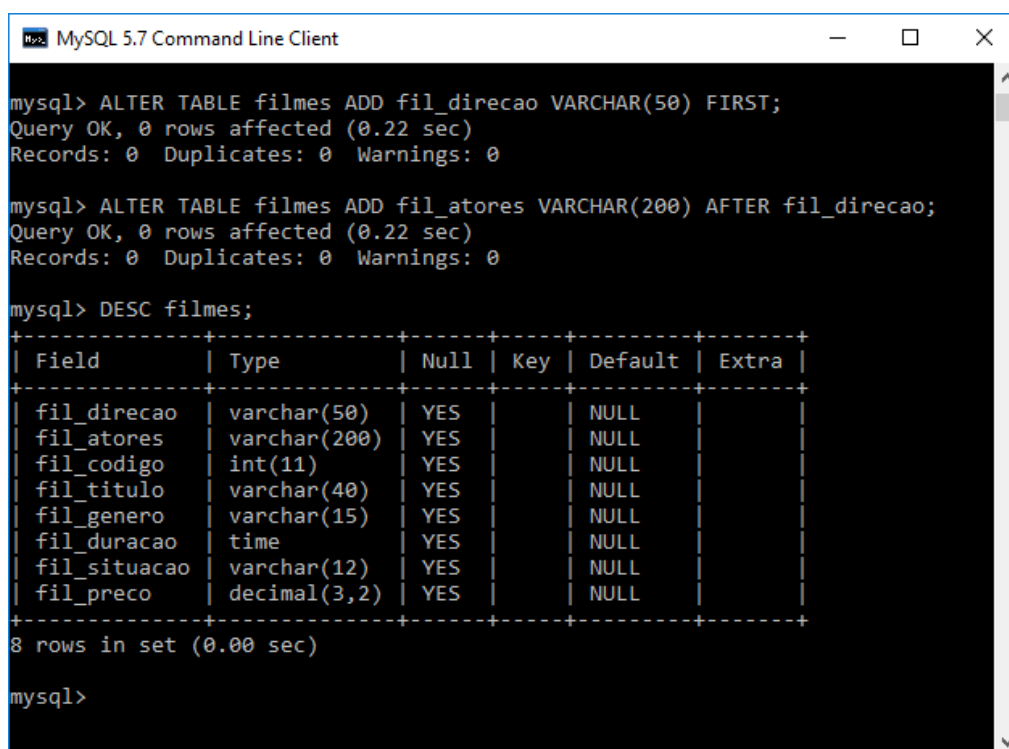
É importante destacar que a utilização dessas palavras-chaves, para o posicionamento das colunas, depende do SGBD adotado, por exemplo, no *MySQL* só é permitido o emprego de `FIRST` e `AFTER`.

Para uma melhor compreensão, analise as seguintes inserções na tabela **filmes** da nossa locadora.

```
1 mysql> ALTER TABLE filmes
2   ADD fil_direcao VARCHAR(50)
3   FIRST;
4
5 mysql> ALTER TABLE filmes
6   ADD fil_atores VARCHAR(200)
7   AFTER fil_direcao;
```

A primeira inserção introduz a coluna *fil\_direção*, do tipo `VARCHAR`, no início da lista de atributos, e a segunda inserção introduz a coluna *fil\_atores* após a coluna *fil\_direção*. As respostas do SGBD às referidas inserções e a nova estrutura da tabela **filmes** são ilustradas na **Figura 3**. Independente da ordem que as colunas sejam definidas na estrutura da tabela, ao fazermos uma consulta com o `SELECT`, podemos especificar a ordem mais adequada para nossa visualização.

**Figura 03** - Tela do MySQL após os comandos ALTER e DESC.



```
mysql> ALTER TABLE filmes ADD fil_direcao VARCHAR(50) FIRST;
Query OK, 0 rows affected (0.22 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE filmes ADD fil_atores VARCHAR(200) AFTER fil_direcao;
Query OK, 0 rows affected (0.22 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC filmes;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| fil_direcao | varchar(50)   | YES  |     | NULL    |       |
| fil_atores  | varchar(200)  | YES  |     | NULL    |       |
| fil_codigo  | int(11)       | YES  |     | NULL    |       |
| fil_titulo  | varchar(40)   | YES  |     | NULL    |       |
| fil_genero  | varchar(15)   | YES  |     | NULL    |       |
| fil_duracao | time          | YES  |     | NULL    |       |
| fil_situacao | varchar(12)   | YES  |     | NULL    |       |
| fil_preco   | decimal(3,2)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql>
```

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

Uma boa prática de programação é ter apenas colunas necessárias nas tabelas de um banco de dados. Atributos (ou colunas) desnecessários ocupam espaço de armazenamento e provocam uma redução no tempo de resposta a consultas no seu SGBD.

Os atributos desnecessários em tabelas podem ser eliminados com o comando ALTER TABLE juntamente com o comando DROP COLUMN. Mas, atenção! Utilize o comando DROP COLUMN com cuidado, pois ele irá excluir todos os dados contidos na coluna em questão. A sintaxe do comando ALTER para excluir colunas em tabelas é descrita no quadro a seguir.

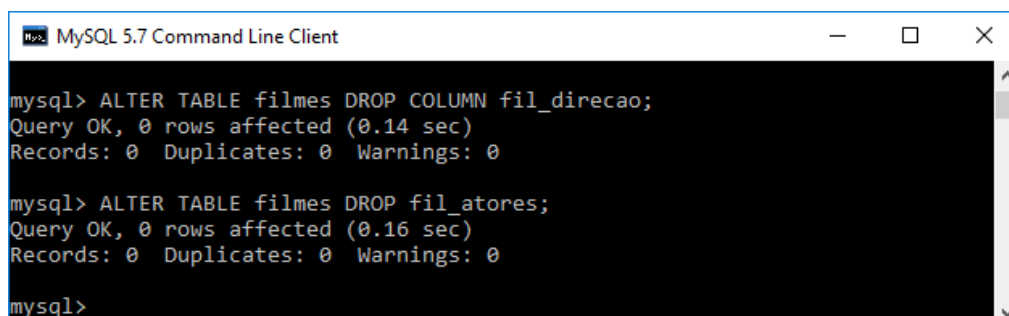
```
1 mysql> ALTER TABLE nome_da_tabela DROP COLUMN atributo;
```

Vamos praticar o comando ALTER TABLE com DROP COLUMN excluindo os atributos *fil\_direcao* e *fil\_atores* na tabela **filmes**?

```
1 mysql> ALTER TABLE filmes DROP COLUMN fil_direcao;
2 mysql> ALTER TABLE filmes DROP COLUMN fil_atores;
```

Observe com cuidado as exclusões das colunas *fil\_direcao* e *fil\_atores* na **Figura 4**. Percebeu que na última exclusão não foi colocado o termo COLUMN (coluna)? No MySQL, o termo COLUMN é opcional.

**Figura 04** - Tela do MySQL após os comandos DROP e DROP COLUMN.



```
mysql> ALTER TABLE filmes DROP COLUMN fil_direcao;
Query OK, 0 rows affected (0.14 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE filmes DROP fil_atores;
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

É possível ainda modificar colunas existentes em uma tabela. Você pode alterar, por exemplo, a definição da coluna ou mesmo o nome da coluna.

Para modificar o nome de uma coluna, devemos utilizar o comando CHANGE COLUMN. A sintaxe para modificar uma coluna é apresentada no quadro a seguir:

```
1 mysql> ALTER TABLE nome_da_tabela CHANGE COLUMN nome_atual novo_nome tipo;
```

Vejamos esse comando na prática:

```
1 mysql> ALTER TABLE clientes CHANGE COLUMN cli_email cli_endereco_eletronico
2 varchar(80);
```

No exemplo acima, modificamos o nome do campo cli\_email da tabela clientes para cli\_endereco\_eletronico e alteramos ainda a definição da coluna de varchar(50) para varchar(80).

E se quiséssemos apenas modificar a definição de uma coluna? Será que é necessário modificar também o nome da coluna? A resposta é não. O comando MODIFY COLUMN permite modificar a definição de uma coluna sem precisarmos alterar o nome da coluna.

Veja a sintaxe:

```
1 mysql> ALTER TABLE nome_da_tabela MODIFY COLUMN nome_da_coluna tipo;
```

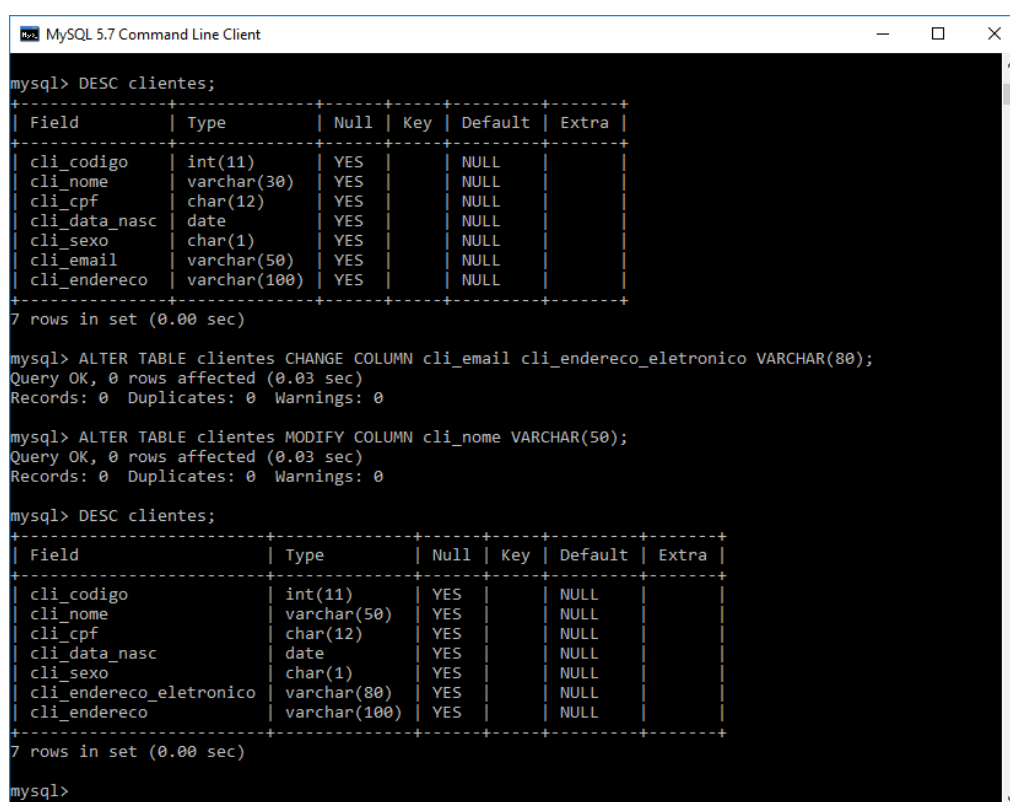
Vejamos um exemplo prático para que seja possível entender melhor o comando acima:

```
1 mysql> ALTER TABLE clientes MODIFY COLUMN cli_nome varchar(50);
```

No exemplo acima, alteramos a definição da coluna `cli_nome` da tabela `clientes`. Trocamos de `varchar(30)` para `varchar(50)`.

A **Figura 5** abaixo mostra como ficou a tabela `clientes` após modificarmos as colunas `cli_nome` e `cli_email`.

**Figura 05** - Tela do MySQL após os comandos `DESC clientes`, `ALTER TABLE ... CHANGE COLUMN ...` e `ALTER TABLE ... MODIFY COLUMN ...`, respectivamente.



```
mysql> DESC clientes;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cli_codigo | int(11) | YES | | NULL | |
| cli_nome | varchar(30) | YES | | NULL | |
| cli_cpf | char(12) | YES | | NULL | |
| cli_data_nasc | date | YES | | NULL | |
| cli_sexo | char(1) | YES | | NULL | |
| cli_email | varchar(50) | YES | | NULL | |
| cli_endereco | varchar(100) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> ALTER TABLE clientes CHANGE COLUMN cli_email cli_endereco_eletronico VARCHAR(80);
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE clientes MODIFY COLUMN cli_nome VARCHAR(50);
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC clientes;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cli_codigo | int(11) | YES | | NULL | |
| cli_nome | varchar(50) | YES | | NULL | |
| cli_cpf | char(12) | YES | | NULL | |
| cli_data_nasc | date | YES | | NULL | |
| cli_sexo | char(1) | YES | | NULL | |
| cli_endereco_eletronico | varchar(80) | YES | | NULL | |
| cli_endereco | varchar(100) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql>
```

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



**Vídeo 02** - Alteração da Estrutura de uma Tabela



# Atividade 01

---

1. Qual a diferença entre o comando ALTER TABLE com CHANGE COLUMN e o comando ALTER TABLE com MODIFY COLUMN?
2. Descubra outras possibilidades do comando ALTER TABLE acessando o HELP do seu sistema SGBD. Se a página acessada estiver em inglês, será uma ótima oportunidade de você treinar o que aprendeu na disciplina de Inglês Técnico.

## Ambientes multitabelas

---

A natureza do projeto de um banco de dados relacional significa que, muitas vezes, dispomos de dados relacionados que são armazenados em tabelas diferentes. Até agora, trabalhamos com tabelas não relacionadas, chegou a hora de aprendermos a trabalhar em banco de dados com várias tabelas relacionadas entre si.

Para realizarmos os nossos estudos em um banco de dados multitabelas, vamos implementar um novo banco de dados que representa um sistema de venda de ingressos para filmes em uma cadeia de cinemas. Nesse banco, temos as seguintes tabelas:

- Cinema (codigo\_Cinema [chave primária], nome, sala, capacidade, cidade);
- Filmes (codigo\_Filme [chave primária], titulo, genero, censura);
- Sessao (codigo\_Sessao [chave primária], codigo\_Filme [chave estrangeira], codigo\_Cinema [chave estrangeira], dataHora, preco);
- Clientes (codigo\_Clientes [chave primária], nome, CPF, sexo, dataNascimento);

- Compras (codigo\_Compra [chave primária], codigo\_Clientes [chave estrangeira], codigo\_Sessao [chave estrangeira], quantidadeInteira, quantidadeMeia, formaPagamento [dinheiro, crédito ou débito]).

Vamos aproveitar a oportunidade para revisar os conceitos aprendidos?

Inicialmente, devemos criar o banco de dados chamado **cineOnline** no qual posteriormente vamos criar nossas tabelas. Para criarmos o banco de dados chamado **cineOnline**, digitamos o comando a seguir:

```
1 mysql>CREATE DATABASE cineOnline;
2   mysql>CREATE TABLE cinema
3   (
4       cinema_codigo int NOT NULL,
5       cinema_nome varchar(40) NOT NULL,
6       cinema_sala varchar(2) NOT NULL,
7       cinema_capacidade int NOT NULL,
8       cinema_cidade varchar(50) NOT NULL,
9       PRIMARY KEY(cinema_codigo)
10  );
```

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

```
1 mysql>USE cineOnline;
```

A seguir, podemos criar as nossas tabelas. Observe, que além de definir os atributos e seus tipos é necessário, neste projeto, identificar os atributos que serão chaves primárias e chaves estrangeiras. Mas, como definir um atributo como sendo chave primária ou como chave estrangeira?

A especificação de uma chave primária de uma tabela tem sintaxe bem simples:

```
1 PRIMARY KEY (nome_da_chave_primária)
```

Para entendermos melhor a utilização do comando CREATE TABLE com a restrição PRIMARY KEY, analise o seguinte exemplo de criação da tabela **cinema** no nosso banco de dados **cineOnline**.

```

1  mysql>CREATE TABLE cinema
2  (
3      cinema_codigo int NOT NULL,
4      cinema_nome varchar(40) NOT NULL,
5      cinema_sala varchar(2) NOT NULL,
6      cinema_capacidade int NOT NULL,
7      cinema_cidade varchar(50) NOT NULL,
8          PRIMARY KEY(cinema_codigo)
9  );
10

```

Nessa tabela, o atributo **cinema\_codigo** foi definido como sendo uma chave primária. Observe que todos os atributos da tabela **cinema** foram definidos com a restrição NOT NULL, que especifica que essas colunas não podem ser deixadas em branco. Essa restrição é especificamente útil para o atributo cinema\_codigo, visto que se a chave primária contém um valor NULL, ou nenhum valor, não se pode garantir que ele (atributo) identificará de forma única cada linha da tabela. O MySQL, como a maioria dos SGBDs, implicitamente já gerenciam as colunas de chave primária como NOT NULL, não sendo obrigatório explicitamente identificar o campo com este atributo.

A criação de uma chave estrangeira tem procedimento similar à criação de uma chave primária. A especificação formal de uma chave estrangeira possui a seguinte sintaxe:

```

1  FOREIGN KEY (nome_da_chave_estrangeira)
2  REFERENCE nome_da_tabela_origem
3  (nome_da_chave primaria_na_tabela_origem)

```

A linha REFERENCE (nome\_da\_tabela\_origem nome\_da\_chave\_primaria\_na\_tabela\_origem) especifica de onde a chave estrangeira veio e como ela é chamada na tabela de origem.

Agora, estamos aptos a criar as tabelas **filmes**, **sessão**, **clientes** e **compras** do nosso banco de dados de vendas de ingressos, no **cineOnline**. Examine com cuidado e não deixe de praticar em seu banco de dados. Lembre-se de que a prática leva à perfeição!

Criação da tabela **filmes**:

```
1 mysql>CREATE TABLE filmes
2 (
3     fil_codigo int NOT NULL,
4     fil_titulo varchar(50) NOT NULL,
5     fil_genero varchar(30) NOT NULL,
6     fil_censura char(8) NOT NULL DEFAULT 'Livre',
7     PRIMARY KEY(fil_codigo)
8 );
```

Nessa tabela, o atributo **fil\_censura** foi declarado como tendo as restrições NOT NULL e DEFAULT 'Livre'. O valor 'Livre' é automaticamente atribuído à coluna **fil\_censura** cada vez que um registro é inserido sem que nenhum valor seja informado.

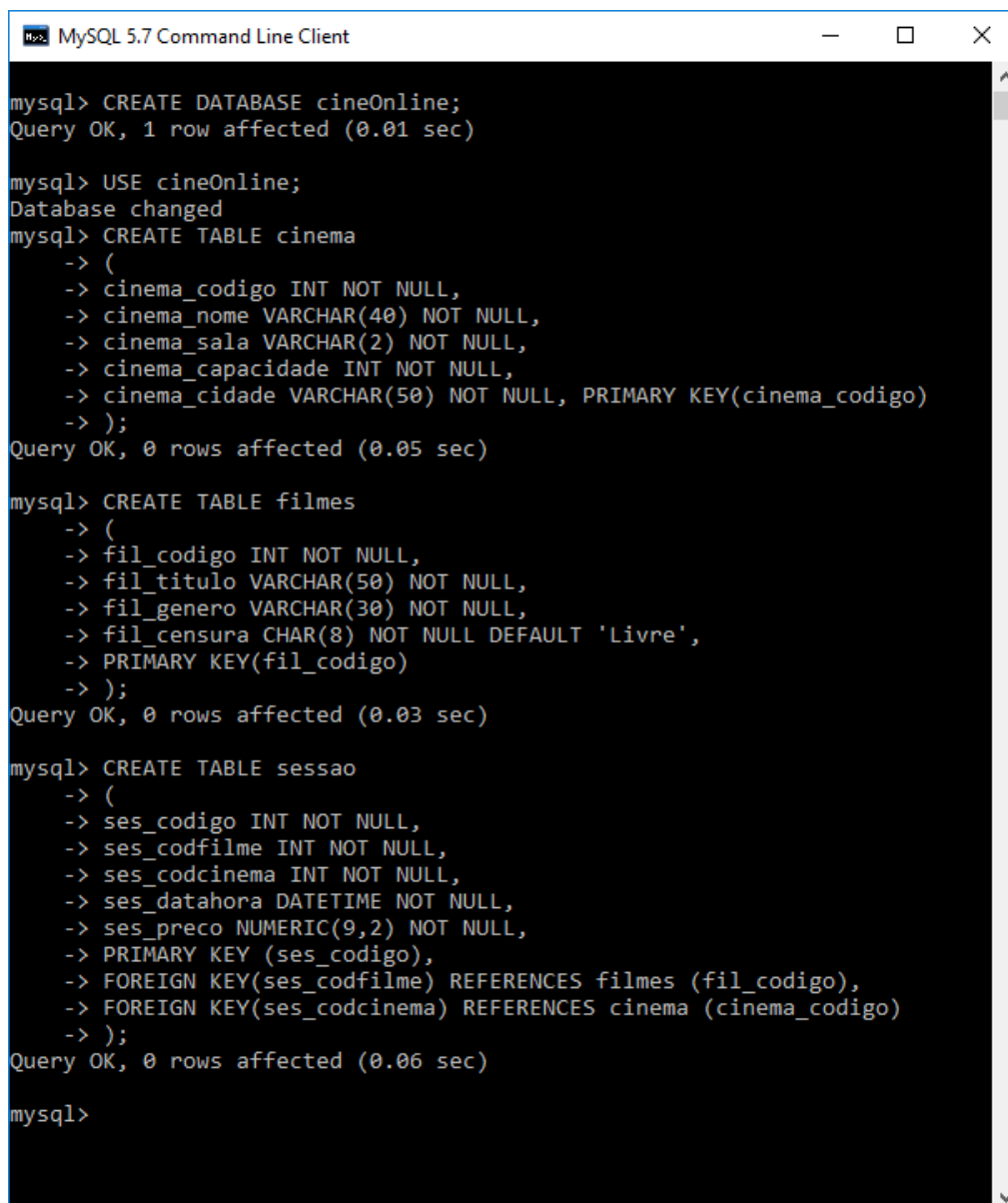
Criação da tabela **sessao**:

```
1 CREATE TABLE sessao
2 (
3     ses_codigo int NOT NULL,
4     ses_codfilme int NOT NULL,
5     ses_codcinema int NOT NULL,
6     ses_datahora datetime NOT NULL,
7     ses_preco numeric(9,2) NOT NULL,
8     PRIMARY KEY (ses_codigo),
9     FOREIGN KEY (ses_codfilme)
10    REFERENCES filmes (fil_codigo),
11    FOREIGN KEY (ses_codcinema)
12    REFERENCES cinema (cinema_codigo)
13 );
```

Observe que o atributo **ses\_codcinema** foi definido como uma chave estrangeira, ele faz referência ao atributo **cinema\_codigo** da tabela **cinema**, linha de baixo.

A criação do banco de dados **cineOnline** e das tabelas **cinema**, **filmes** e **sessao** utilizando os comandos CREATE DATABASE, USE e CREATE TABLE estão ilustradas na **Figura 6**.

**Figura 06** - Tela do MySQL após os comandos CREATE DATABASE, USE e CREATE TABLE.



```
mysql> CREATE DATABASE cineOnline;
Query OK, 1 row affected (0.01 sec)

mysql> USE cineOnline;
Database changed
mysql> CREATE TABLE cinema
-> (
-> cinema_codigo INT NOT NULL,
-> cinema_nome VARCHAR(40) NOT NULL,
-> cinema_sala VARCHAR(2) NOT NULL,
-> cinema_capacidade INT NOT NULL,
-> cinema_cidade VARCHAR(50) NOT NULL, PRIMARY KEY(cinema_codigo)
-> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE filmes
-> (
-> fil_codigo INT NOT NULL,
-> fil_titulo VARCHAR(50) NOT NULL,
-> fil_genero VARCHAR(30) NOT NULL,
-> fil_censura CHAR(8) NOT NULL DEFAULT 'Livre',
-> PRIMARY KEY(fil_codigo)
-> );
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE sessao
-> (
-> ses_codigo INT NOT NULL,
-> ses_codfilme INT NOT NULL,
-> ses_codcinema INT NOT NULL,
-> ses_datahora DATETIME NOT NULL,
-> ses_preco NUMERIC(9,2) NOT NULL,
-> PRIMARY KEY (ses_codigo),
-> FOREIGN KEY(ses_codfilme) REFERENCES filmes (fil_codigo),
-> FOREIGN KEY(ses_codcinema) REFERENCES cinema (cinema_codigo)
-> );
Query OK, 0 rows affected (0.06 sec)

mysql>
```

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



**Vídeo 03** - Criação de Chaves

## Atividade 02

---

1. Vamos praticar um pouco para que você se familiarize com o comando de criação de tabelas contendo chave primária e chave estrangeira. Entre no banco de dados da nossa empresa de venda de ingressos (**cineOnline**) e crie as tabelas **clientes** e **compras**, coloque as restrições que achar necessárias. Não se esqueça de visualizar todas as estruturas das tabelas criadas. Caso alguma tabela tenha algum problema, utilize o comando ALTER para alterar a estrutura da tabela.
2. A seguir, insira alguns dados nas tabelas **cinema**, **filmes**, **sessao**, **clientes** e **compras**. É importante nesse momento de aprendizagem que você certifique-se de que os dados foram inseridos corretamente, utilizando o comando SELECT. Se tiver dúvidas, consulte o material das aulas anteriores nas quais você aprendeu a criar, apagar tabelas e selecionar dados em tabelas. A realização desta atividade irá ajudá-lo a compreender e praticar os próximos comandos que serão mostrados nesta aula.

## Consultas em ambientes multitabelas

---

Já vimos que em um banco de dados relacional, muitas vezes, os dados são armazenados em tabelas diferentes, entretanto, eles estão relacionados entre si, sendo essa relação definida entre determinadas colunas dessas tabelas. Mas, como fazer para realizar uma consulta que combine os registros de duas ou mais tabelas em um banco de dados?

A solução para essa pergunta está no uso de conexões definidas pela cláusula JOIN (conexões). A cláusula JOIN em SQL permite combinar os registros de duas ou mais tabelas em um banco de dados. Por meio dessa cláusula, é possível pesquisar dados em duas ou mais tabelas, com base nas relações entre determinadas colunas nessas tabelas.

Vamos começar nosso estudo sobre conexões estudando o tipo mais simples de conexão denominada de conexão cartesiana ou de produto cartesiano.

Uma consulta a duas tabelas utilizando a cláusula CROSS JOIN vai retornar todos os registros resultantes da combinação de cada linha da primeira tabela (**A**) com cada linha da segunda tabela (**B**). A conexão cruzada combina cada linha da tabela **A**, com cada linha da tabela **B**. O número total de linhas no conjunto de resultados será o número de linhas da tabela **A** vezes o número de linhas da tabela **B**. Ou seja, a cláusula CROSS JOIN retorna o produto cartesiano dos conjuntos de linhas das tabelas.

No comando SELECT, a conexão cruzada pode ser especificada inserindo a palavra CROSS JOIN entre os nomes das tabelas na cláusula FROM, ou simplesmente utilizando uma vírgula (,) entre os nomes das tabelas. A sintaxe do comando SELECT com conexão cruzada é descrita no quadro a seguir.

```
1 mysql>SELECT atributo1_da_tabela1, atributo1_da_tabela2, ...  
2 FROM nome_da_tabela1 CROSS JOIN nome_da_tabela2, ...;
```

No comando SELECT, os valores representados por atributo1\_da\_tabela1, atributo1\_da\_tabela2, ... compõem a lista de atributos das diferentes tabelas que você deseja consultar. A cláusula FROM informa de quais tabelas os dados serão recuperados.

Uma forma de identificar o atributo de uma determinada tabela é utilizando a sintaxe **nome\_da\_tabela.nome\_da\_coluna**, dessa forma, o sistema sabe exatamente a qual tabela aquele determinado atributo pertence. Essa sintaxe é especialmente útil quando atributos com nomes idênticos estiverem presentes em mais de uma tabela.

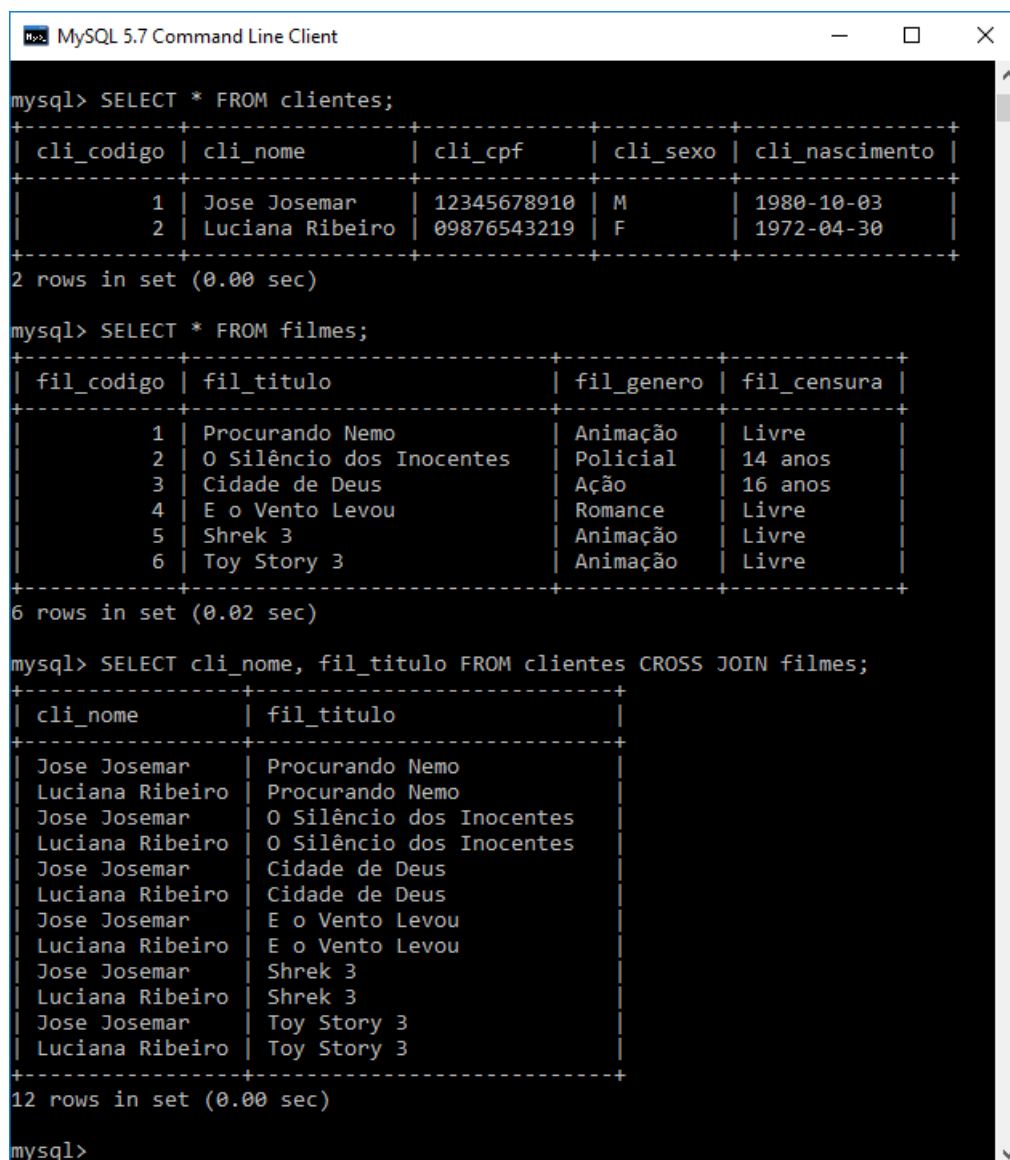
Vamos analisar a seguinte consulta, no nosso banco de dados **cineOnline**, para fundamentar melhor o conceito de conexão cruzada.

```
1 mysql>SELECT cli_nome, fil_titulo  
2 FROM clientes CROSS JOIN filmes;
```

Observe que não foi necessária a utilização da sintaxe **nome\_da\_tabela.nome\_da\_coluna**, pois estamos usando um sistema de nomenclatura de atributos que possui uma abreviação do nome da tabela antes de cada atributo, de forma a identificar a qual tabela pertence aquele atributo.

Nessa consulta às tabelas **clientes** e **filmes**, é solicitada a visualização do nome do cliente e o título do filme de todas as combinações possíveis entre esses dois campos, conforme é ilustrado na **Figura 7**. Além da resposta à consulta anterior, a **Figura 7** contém a visualização de todos os dados contidos nas tabelas **clientes** e **filmes** do nosso banco de dados **cineOnline**, para que você compreenda melhor a resposta do sistema à consulta realizada.

**Figura 07** - Tela do MySQL mostrando o conteúdo das tabelas **clientes** e **filmes** e o comando SELECT com conexão cruzada.



```
mysql> SELECT * FROM clientes;
+-----+-----+-----+-----+-----+
| cli_codigo | cli_nome      | cli_cpf      | cli_sexo | cli_nascimento |
+-----+-----+-----+-----+-----+
| 1          | Jose Josemar  | 12345678910  | M        | 1980-10-03     |
| 2          | Luciana Ribeiro | 09876543219  | F        | 1972-04-30     |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM filmes;
+-----+-----+-----+-----+
| fil_codigo | fil_titulo      | fil_genero | fil_censura |
+-----+-----+-----+-----+
| 1          | Procurando Nemo | Animação   | Livre       |
| 2          | O Silêncio dos Inocentes | Policial   | 14 anos     |
| 3          | Cidade de Deus  | Ação       | 16 anos     |
| 4          | E o Vento Levou | Romance    | Livre       |
| 5          | Shrek 3         | Animação   | Livre       |
| 6          | Toy Story 3     | Animação   | Livre       |
+-----+-----+-----+-----+
6 rows in set (0.02 sec)

mysql> SELECT cli_nome, fil_titulo FROM clientes CROSS JOIN filmes;
+-----+-----+
| cli_nome      | fil_titulo      |
+-----+-----+
| Jose Josemar  | Procurando Nemo |
| Luciana Ribeiro | Procurando Nemo |
| Jose Josemar  | O Silêncio dos Inocentes |
| Luciana Ribeiro | O Silêncio dos Inocentes |
| Jose Josemar  | Cidade de Deus  |
| Luciana Ribeiro | Cidade de Deus  |
| Jose Josemar  | E o Vento Levou |
| Luciana Ribeiro | E o Vento Levou |
| Jose Josemar  | Shrek 3         |
| Luciana Ribeiro | Shrek 3         |
| Jose Josemar  | Toy Story 3     |
| Luciana Ribeiro | Toy Story 3     |
+-----+-----+
12 rows in set (0.00 sec)

mysql>
```

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



# Exemplos

---

Para entendermos melhor a utilização de pesquisas com conexão cruzada, vamos analisar os seguintes exemplos. Examine com cuidado e não deixe de praticar em seu banco de dados. Lembre-se de que a prática leva à perfeição!

## Exemplo 1

Pesquisar os nomes dos cinemas e as respectivas salas que estão exibindo o filme Procurando Nemo.

```
1 mysql> SELECT cinema_nome, cinema_sala
2   FROM filmes, sessao, cinema
3   WHERE fil_titulo = 'Procurando Nemo'
4     AND ses_codfilme = fil_codigo
5     AND cinema_codigo = ses_codcinema;
```

O resultado dessa pesquisa é ilustrado na **Figura 8**. Observe com cuidado esses resultados. É interessante notar que a consulta é formulada fazendo uma associação entre chaves primárias e estrangeiras das diversas tabelas. Note que inicialmente são selecionadas as linhas do produto cartesiano cujo título do filme é Procurando Nemo, em seguida, dentro desse conjunto são selecionadas as sessões que correspondem ao código do filme e, finalmente, os cinemas que estão associados a essas sessões. Lembre-se: o uso da cláusula WHERE implica que só são mostradas as linhas que satisfaçam as condições de consulta.

**Figura 08** - Tela do MySQL mostrando o conteúdo das tabelas **clientes** e **filmes** e o comando SELECT com conexão cruzada.

```

mysql> SELECT * FROM cinema;
+-----+-----+-----+-----+-----+
| cinema_codigo | cinema_nome | cinema_sala | cinema_capacidade | cinema_cidade |
+-----+-----+-----+-----+-----+
| 1 | Midway | 1 | 50 | Natal |
| 2 | Midway | 2 | 50 | Natal |
| 3 | Midway | 3 | 50 | Natal |
| 4 | Midway | 4 | 120 | Natal |
| 5 | UCI Recife | 1 | 150 | Recife |
| 6 | UCI Recife | 2 | 150 | Recife |
| 7 | UCI Recife | 3 | 150 | Recife |
| 8 | UCI Recife | 4 | 220 | Recife |
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM filmes;
+-----+-----+-----+-----+
| fil_codigo | fil_titulo | fil_genero | fil_censura |
+-----+-----+-----+-----+
| 1 | Procurando Nemo | Animação | Livre |
| 2 | O Silêncio dos Inocentes | Policial | 14 anos |
| 3 | Cidade de Deus | Ação | 16 anos |
| 4 | E o Vento Levou | Romance | Livre |
| 5 | Shrek 3 | Animação | Livre |
| 6 | Toy Story 3 | Animação | Livre |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> SELECT cinema_nome, cinema_sala FROM filmes, sessao, cinema
-> WHERE fil_titulo='Procurando Nemo'
-> AND ses_codfilme = fil_codigo
-> AND cinema_codigo = ses_codcinema;
+-----+-----+
| cinema_nome | cinema_sala |
+-----+-----+
| Midway | 1 |
| Midway | 2 |
| Midway | 3 |
| Midway | 4 |
| UCI Recife | 1 |
| UCI Recife | 2 |
+-----+-----+
6 rows in set (0.00 sec)

mysql>

```

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

## Exemplo 2

Pesquisar os filmes que estão sendo exibidos fora o filme Procurando Nemo, disponibilizando o nome do cinema e suas respectivas salas.

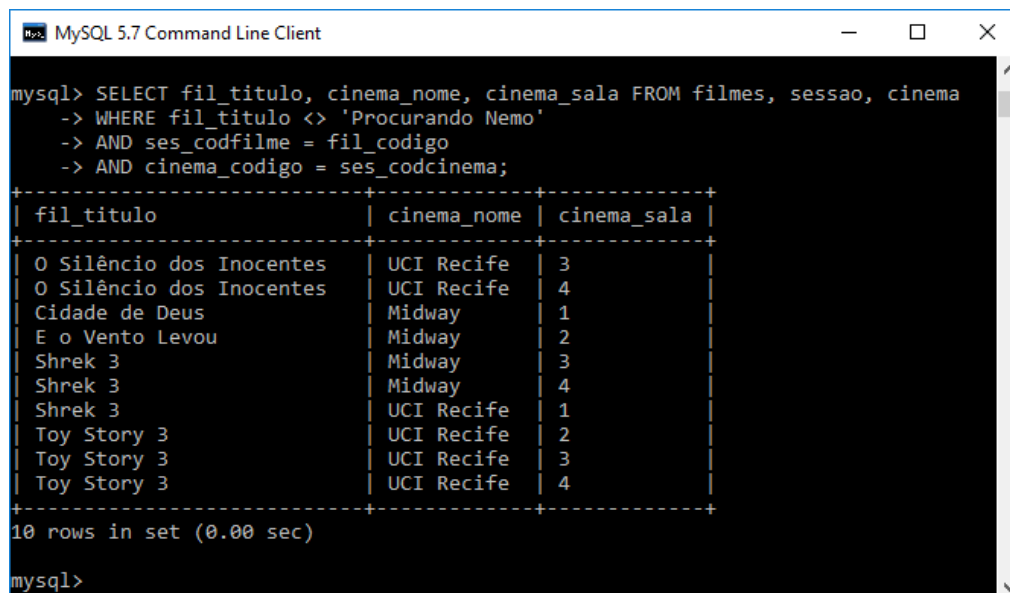
```

1 mysql> SELECT fil_titulo, cinema_nome, cinema_sala
2   FROM filmes, sessao, cinema
3   WHERE fil_titulo <> 'Procurando Nemo'
4   AND ses_codfilme = fil_codigo
5   AND cinema_codigo = ses_codcinema;

```

O resultado dessa pesquisa é ilustrado na **Figura 9**. Observe que as linhas listadas correspondem ao produto cartesiano entre as tabelas **filmes** e **cinema** que não possuem o filme Procurando Nemo.

**Figura 09** - Tela do MySQL mostrando o comando SELECT com conexão cruzada.



```
mysql> SELECT fil_titulo, cinema_nome, cinema_sala FROM filmes, sessao, cinema
-> WHERE fil_titulo <> 'Procurando Nemo'
-> AND ses_codfilme = fil_codigo
-> AND cinema_codigo = ses_codcinema;
+-----+-----+-----+
| fil_titulo | cinema_nome | cinema_sala |
+-----+-----+-----+
| O Silêncio dos Inocentes | UCI Recife | 3 |
| O Silêncio dos Inocentes | UCI Recife | 4 |
| Cidade de Deus | Midway | 1 |
| E o Vento Levou | Midway | 2 |
| Shrek 3 | Midway | 3 |
| Shrek 3 | Midway | 4 |
| Shrek 3 | UCI Recife | 1 |
| Toy Story 3 | UCI Recife | 2 |
| Toy Story 3 | UCI Recife | 3 |
| Toy Story 3 | UCI Recife | 4 |
+-----+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

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



**Vídeo 04** - Consultas Relacionadas

## Atividade 03

1. Qual a diferença entre o comando SELECT com conexão cartesiana e o comando SELECT simples que utilizamos nas aulas anteriores?
2. Qual o resultado da seguinte consulta: `SELECT * FROM clientes CROSS JOIN filmes`?
3. Qual o resultado da seguinte consulta: `SELECT cinema_nome, cinema_sala, fil_titulo FROM cinema, filmes WHERE fil_titulo = 'Procurando Nemo' GROUP BY cinema_nome`?

4. Compare o resultado obtido na consulta realizada na Questão 3 com a consulta do Exemplo 1.
5. Você pode utilizar funções como SUM e AVG nas consultas com conexão? Cite exemplos.

## Conclusão

---

Encerramos por aqui nossa quinta aula sobre a linguagem SQL. Na próxima aula, aprenderemos a selecionar um resultado de uma consulta e usá-lo como entrada para outra consulta, ou seja, iremos trabalhar com consultas aninhadas, denominadas subconsultas. Lembre-se de fazer sua autoavaliação. E se precisar, pare e reflita mais um pouco sobre o que estudamos.

Bons estudos e boa sorte!

# Resumo

---

Nesta aula, você estudou o comando ALTER, que é utilizado para alterar a estrutura de uma tabela em um banco de dados. Aprendeu a trabalhar com múltiplas tabelas, especificando atributos como sendo chave primária e chave estrangeira, utilizando para isso as palavras chaves PRIMARY KEY e FOREIGN KEY. Estudou ainda o processo de consulta no contexto multitabelas, usando as conexões cartesianas definidas pela cláusula CROSS JOIN.

## Autoavaliação

---

1. Crie um banco de dados chamado SistemaFidelizacao.
2. Nesse banco, crie as tabelas a seguir e insira dados nelas, de acordo com as seguintes informações:
  - a. clientes (codigoCliente [chave primária], nome, CPF, profissao, saldoPontos)
  - b. compras (codigoCompra [chave primária], codigoCliente [chave estrangeira], data, valor, pontosGanhos)
  - c. premios (codigoPremio [chave primária], descricao, valorPontos, quantEstoque)
  - d. trocas (codigoTroca [chave primária], codigoCliente [chave estrangeira], codigoPremio [chave estrangeira], quantidade, data)
3. Acrescente na tabela clientes um atributo para conter o e-mail do cliente.
4. Considerando o banco de dados do sistema de Fidelização, escreva comandos SQL para exibir, para cada troca realizada, o nome do cliente, a descrição do prêmio e a quantidade trocada, ordenadas por data, usando conexões.
5. Considerando o banco de dados do sistema de Fidelização, escreva comandos SQL para exibir uma lista dos prêmios que já podem ser resgatados pelos clientes de acordo com o seu saldo atual de pontos.

## 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.