



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 12 - Linguagem SQL - Alteração da estrutura de tabelas e ambientes de múltiplas tabelas

Professores autores

José Josemar de Oliveira Júnior (josemar@ect.ufrn.br)

Luciana Ribeiro Veloso (luciana.veloso@globocom)

Apresentacao

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, vamos aprender como especificar um atributo como sendo chave primária e chave estrangeira. A seguir, iniciamos o estudo sobre conexões, estudando a consulta de dados usando conexões cartesianas CROSS JOIN.

Objetivo

Ao final desta aula, você será capaz de:

- 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 que o usuário altere 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.

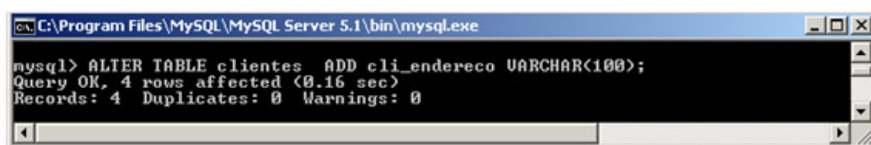
A sintaxe do comando ALTER para adicionar colunas a uma tabela é descrito no quadro abaixo.

```
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.

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

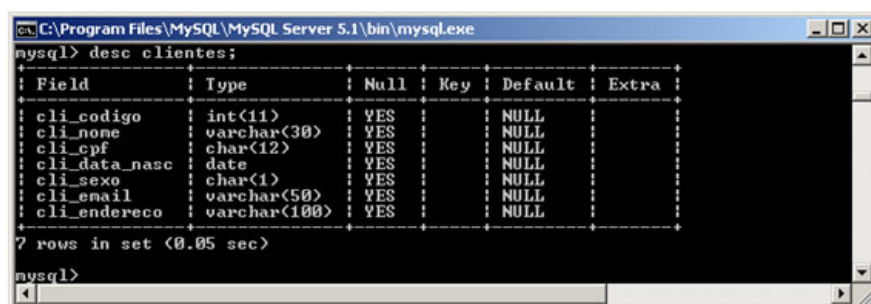
A resposta do SGBD, no caso o MySQL, ao comando acima, é ilustrada na Figura 1. A mensagem "Query OK" informa que a coluna foi adicionada corretamente.



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql> ALTER TABLE clientes ADD cli_endereco VARCHAR(100);
Query OK, 4 rows affected (0.16 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

Figura 1 – Tela do MySQL após o comando ALTER TABLE clientes ADD cli_endereco VARCHAR(100)
Fonte: MySQL Server 5.1

Caso você deseje conferir como ficou a estrutura da tabela **clientes**, utilize o comando DESC, estudado na Aula 11, conforme visto na Figura 2.



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
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 (0.05 sec)

mysql>
```

Figura 2 – Tela do MySQL mostrando a estrutura da tabela **clientes** após a inserção da coluna cli_endereco
Fonte: MySQL Server 5.1

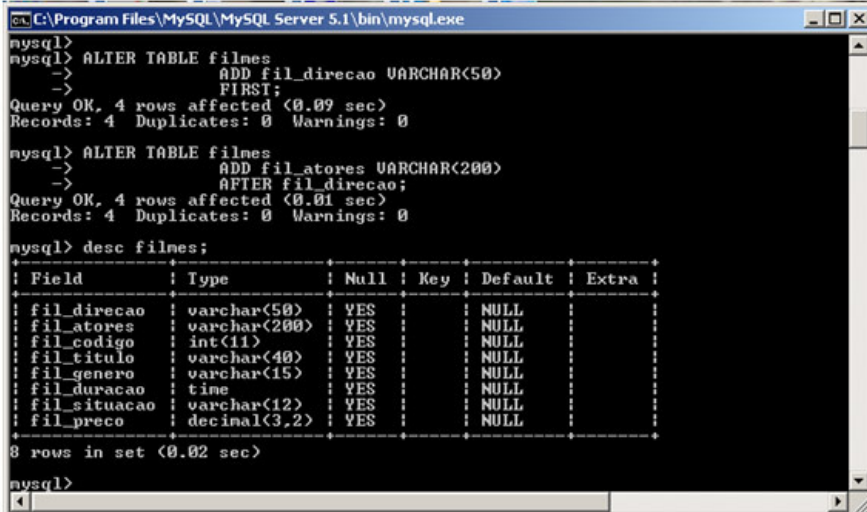
Observe na Figura 2 que o campo `cli_endereco` é o último da lista de atributos. Você pode utilizar palavras-chaves como `FIRST` (primeiro), `AFTER` (após), `BEFORE` (antes) e `LAST` (por último) para posicionar a nova coluna na posição que desejar na tabela. Além dessas palavras-chaves, podem ser utilizadas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

```
mysql> ALTER TABLE filmes  
ADD fil_direcao VARCHAR(50)  
FIRST;
```

```
mysql> ALTER TABLE filmes  
ADD fil_atores VARCHAR(200)  
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 inserções acima 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.



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe  
mysql>  
mysql> ALTER TABLE filmes  
-> ADD fil_direcao VARCHAR(50)  
-> FIRST;  
Query OK, 4 rows affected (0.09 sec)  
Records: 4 Duplicates: 0 Warnings: 0  
  
mysql> ALTER TABLE filmes  
-> ADD fil_atores VARCHAR(200)  
-> AFTER fil_direcao;  
Query OK, 4 rows affected (0.01 sec)  
Records: 4 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 (0.02 sec)  
  
mysql>
```

Figura 3 – Tela do MySQL após os comandos `ALTER` e `DESC`
Fonte: MySQL Server 5.1

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 descrito no quadro abaixo.

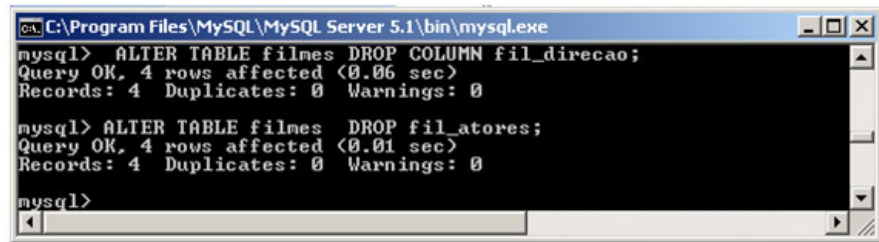
```
mysql> ALTER TABLE nome_da_tabela DROP COLUMN atributo;
```

Vamos praticar o comando `ALTER TABLE` com `DROP COLUMN` excluindo os atributos `fil_direcao` e `fil_atores` na tabela **filmes**?

```
mysql> ALTER TABLE filmes DROP COLUMN fil_direcao;
```

```
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.



```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysql.exe
mysql> ALTER TABLE filmes DROP COLUMN fil_direcao;
Query OK, 4 rows affected (0.06 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE filmes DROP fil_atores;
Query OK, 4 rows affected (0.01 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql>
```

Figura 4 – Tela do MySQL após os comandos `ALTER TABLE ... DROP COLUMN`
Fonte: MySQL Server 5.1

Para alterar o tipo e o nome de uma determinada coluna existente, utiliza-se o comando `ALTER TABLE` com a seguinte sintaxe:

```
mysql> ALTER TABLE nome_da_tabela
CHANGE COLUMN nome_da_coluna novo_nome_da_coluna tipo;
```

Nessa sintaxe, o primeiro argumento após `CHANGE COLUMN` é o nome da coluna que se deseja renomear (nome antigo da coluna), o segundo argumento é o novo nome que a coluna (atributo) deverá ter e o terceiro argumento informa qual o novo tipo de dados dessa coluna. Mas atenção!!! Se o tipo de dados que você estiver alterando não for compatível com os dados já existentes nessa coluna, seu comando não será considerado. Entretanto, se os tipos forem compatíveis, seus dados serão convertidos para o novo tipo, o que poderá provocar a necessidade de algum truncamento nos dados, ocasionando perda de informação. Caso seja necessário alterar o tipo de um atributo, certifique-se de que o novo tipo de dado não vai truncar seus dados antigos, e caso isso ocorra, que sejam perdas inexpressivas.

Se desejar alterar apenas o tipo dos dados de uma coluna sem modificar o nome do atributo, deve-se utilizar o comando `ALTER` com a cláusula `MODIFY`, com a seguinte sintaxe:

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

Vamos praticar o comando `ALTER TABLE` com as palavras-chaves `CHANGE COLUMN` e `MODIFY COLUMN`, alterando o nome da coluna `cli_email` para `cli_endereco_eletronico` com o tipo `VARCHAR(80)` na tabela **clientes** e modificando o tipo da coluna `cli_nome` para `VARCHAR(50)`.

```
mysql> ALTER TABLE clientes
CHANGE COLUMN cli_email cli_endereco_eletronico
VARCHAR(80);

mysql> ALTER TABLE clientes
CHANGE MODIFY cli_nome
VARCHAR(50);
```

As respostas do SGBD, no caso o MySQL, aos comandos acima são ilustrados na Figura 5. Observe na resposta ao comando `DESC clientes` as alterações nas colunas `cli_nome` e `cli_email`. Por exemplo, o atributo `cli_email` era definido como `VARCHAR(50)` e foi alterado para um tipo `VARCHAR(80)`. Isso não provoca perdas, pois não ocorreu necessidade de truncamento.

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql>
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 | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>
mysql> ALTER TABLE clientes
-> CHANGE COLUMN cli_email cli_endereco_eletronico
-> VARCHAR(80);
Query OK, 4 rows affected (0.02 sec)
Records: 4 Duplicates: 0 Warnings: 0

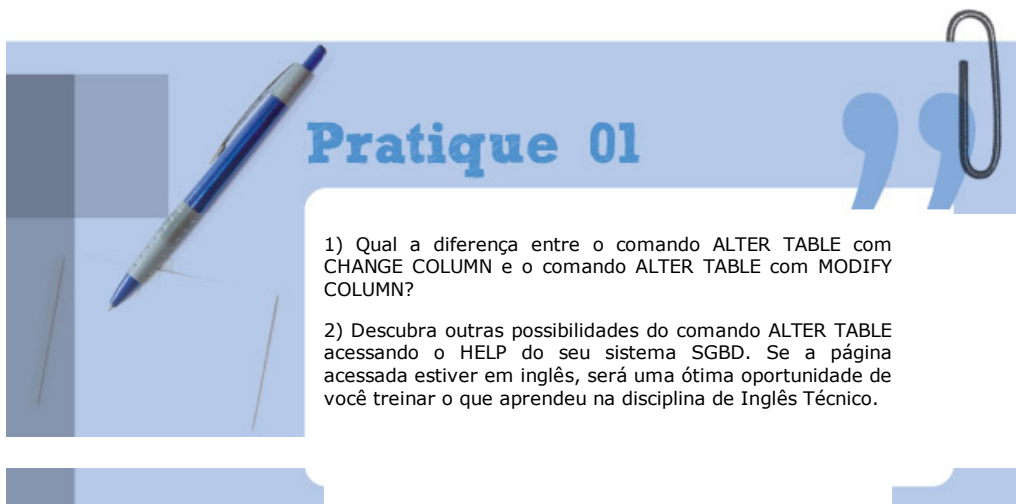
mysql>
mysql> ALTER TABLE clientes
-> MODIFY COLUMN cli_nome
-> VARCHAR(50);
Query OK, 4 rows affected (0.02 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql>
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 | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>

```

Figura 5 – Tela do MySQL após os comandos DESC clientes, ALTER TABLE ... CHANGE COLUMN ... e ALTER TABLE ... MODIFY COLUMN ..., respectivamente Fonte: MySQL Server 5.1



Pratique 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 (Aula 10):

```
mysql>CREATE DATABASE cineOnline;
```

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

```
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:

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**.

```
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)
);
```

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.

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:

FOREIGN KEY (nome_da_chave_estrangeira)
REFERENCE nome_da_tabela_origem
(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**, **sessao**, **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**:

```
mysql>CREATE TABLE filme
(
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)
);
```

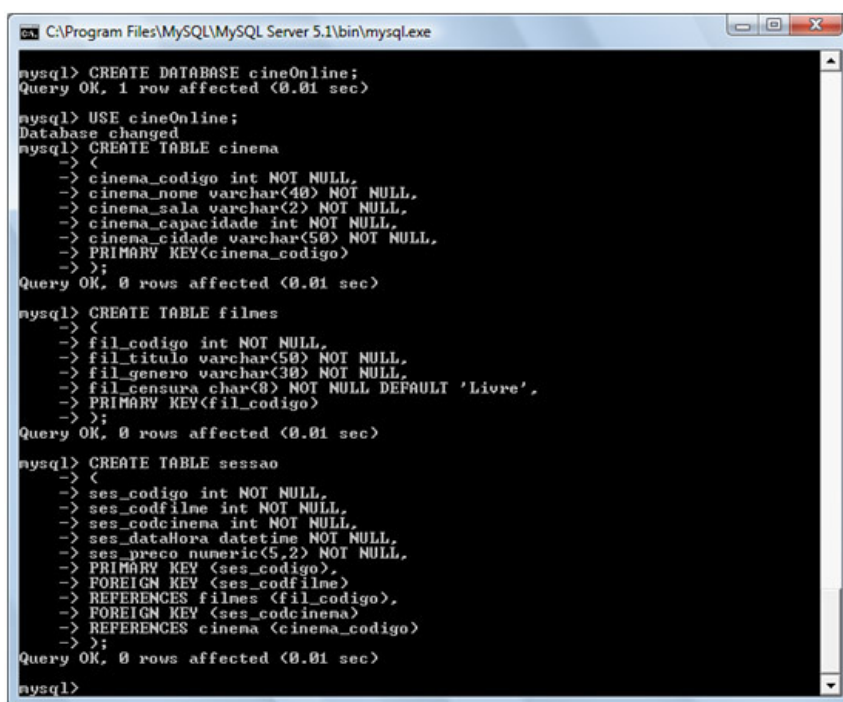
Nessa tabela, o atributo **fil_censura** foi declarado como tendo as restrições NOT NULL e DEFAULT 'Livre'. O valor 'Livre' é automaticamente atribuído à coluna **fil_censura** cada vez que um registro é inserido sem que nenhum valor seja informado.

Criação da tabela **sessao**:

```
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)
);
```

Observe que o atributo **ses_codcinema** foi definido como uma chave estrangeira, ele faz referência ao atributo **cinema_codigo** da tabela **cinema**.

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.



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

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

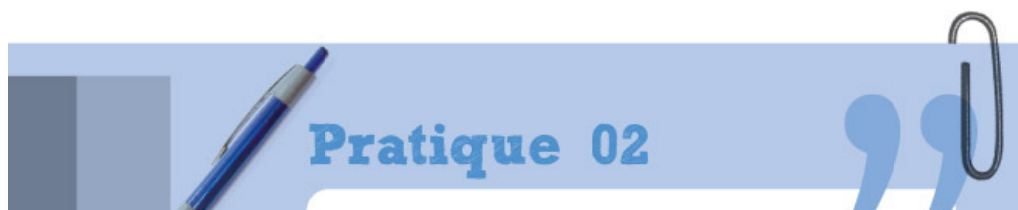
mysql> USE cineOnline;
Database changed
mysql> CREATE TABLE cinema
-> (
->  cinema_codigo int NOT NULL,
->  cinema_nome varchar(40) NOT NULL,
->  cinema_sala varchar(2) NOT NULL,
->  cinema_capacidade int NOT NULL,
->  cinema_cidade varchar(50) NOT NULL,
->  PRIMARY KEY(cinema_codigo)
-> );
Query OK, 0 rows affected (0.01 sec)

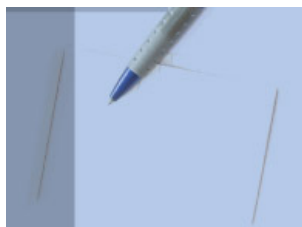
mysql> CREATE TABLE filmes
-> (
->  fil_codigo int NOT NULL,
->  fil_titulo varchar(50) NOT NULL,
->  fil_genero varchar(30) NOT NULL,
->  fil_censura char(8) NOT NULL DEFAULT 'Livre',
->  PRIMARY KEY(fil_codigo)
-> );
Query OK, 0 rows affected (0.01 sec)

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(5,2) 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 (0.01 sec)

mysql>
```

Figura 6 –Tela do MySQL após os comandos CREATE DATABASE, USE e CREATE TABLE
Fonte: MySQL Server 5.1





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.

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 (Aulas 11 e 12). 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.

```
mysql>SELECT atributo1_da_tabela1, atributo1_da_tabela2, ...  
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 à 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.

```
mysql>SELECT cli_nome, fil_titulo  
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 à 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.

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
mysql> SELECT * FROM clientes;
+----+-----+-----+-----+-----+
| cli_codigo | cli_nome | cli_cpf | cli_sexo | cli_datanascim |
+----+-----+-----+-----+-----+
| 1 | Jose Josemar | 3252541 | M | 1980-10-03 |
| 2 | Luciana Ribeiro | 4812072 | 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 | Animacao | Livre |
| 2 | O silencio dos inocentes | Policial | 14 anos |
| 3 | Cidade de Deus | Acao | 16 anos |
| 4 | E o vento levou | Romance | Livre |
| 5 | Shrek 3 | Animacao | Livre |
| 6 | Toy Store 3 | Animacao | Livre |
+----+-----+-----+-----+
6 rows in set (0.00 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 silencio dos inocentes |
| Luciana Ribeiro | O silencio 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 Store 3 |
| Luciana Ribeiro | Toy Store 3 |
+-----+-----+
12 rows in set (0.00 sec)

mysql> _

```

Figura 7 – Tela do MySQL mostrando o conteúdo das tabelas **clientes** e **filmes** e o comando SELECT com conexão cruzada
Fonte: MySQL Server 5.1

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.

```

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;

```

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 titulo 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.

```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
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 | Animacao | Livre |
| 2 | O silencio dos inocentes | Policial | 14 anos |
| 3 | Cidade de Deus | Acao | 16 anos |
| 4 | E o vento levou | Romance | Livre |
| 5 | Shrek 3 | Animacao | Livre |
| 6 | Toy Store 3 | Animacao | 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>

```

Figura 8 – Tela do MySQL mostrando o conteúdo das tabelas **cinema** e **filmes** e o comando SELECT com conexão cruzada
Fonte: MySQL Server 5.1

Exemplo 2

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

```

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;

```

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.


```

C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
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 silencio dos inocentes | UCI Recife | 3 |
| O silencio 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 Store 3 | UCI Recife | 2 |
| Toy Store 3 | UCI Recife | 3 |
| Toy Store 3 | UCI Recife | 4 |
+-----+-----+-----+
10 rows in set (0.00 sec)

mysql>

```

Figura 9 – Tela do MySQL mostrando o comando SELECT com conexão cruzada
Fonte: MySQL Server 5.1



Pratique 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.

Encerramos por aqui nossa quinta aula sobre a linguagem SQL. Na próxima aula, aprenderemos a pegar 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 auto-avaliação. E se precisar, pare e reflita mais um pouco sobre o que estudamos. Bons estudos e boa sorte!

Resumo

Nesta aula, apresentamos o comando ALTER que é utilizado para alterar a estrutura de uma tabela em um banco de dados. Também aprendemos 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. A seguir, ilustramos o processo de consulta no contexto multitabelas usando as conexões cartesianas definidas pela cláusula CROSS JOIN.



Evolucao



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.

- **clientes** (codigoCliente [chave primária], nome, CPF, profissao, saldoPontos)
- **compras** (codigoCompra [chave primária], codigoCliente [chave estrangeira], data, valor, pontosGanhos)
- **premios** (codigoPremio [chave primária], descricao, valorPontos, quantEstoque)
- **trocas** (codigoTroca [chave primária], codigoCliente [chave estrangeira], codigoPremio [chave estrangeira], quantidade, data)

3) Acrescente na tabela **clientes** um atributo para conter o email 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.

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