

# 12 Introdução ao MySQL

## 12.1 Introdução

Neste capítulo é dada uma introdução ao banco de dados MySQL, dando mais ênfase ao aspecto da linguagem SQL. Para quem deseja aprofundar mais sobre SQL, este capítulo não será suficiente. Para aqueles que já sabem SQL, este capítulo será uma revisão e uma introdução às particularidades do banco de dados MySQL. Grande parte deste capítulo faz parte da tradução do tutorial do MySQL encontrado na documentação em sua documentação oficial.

## 12.2 Por que o MySQL?

- O MySQL é um banco de dados cliente-servidor gratuito
- É simples, tem alto desempenho, é disponível para várias plataformas e é robusto
- Possui bom suporte à java (possui driver jdbc)

## 12.3 O que é MySQL?

MySQL é um servidor de banco de dados SQL multi-usuário, com suporte à múltiplas linhas de execução [multi-threaded]. SQL é a linguagem de banco de dados mais popular no mundo. MySQL é uma implementação cliente/servidor que consiste de servidor [server daemon] mysqld e vários programas clientes e bibliotecas.

SQL é a linguagem padronizada que torna fácil armazenar, atualizar e acessar informação. Por exemplo, você pode usar SQL para recuperar informação de produtos e armazenar informação de clientes para um site Internet. MySQL é também suficientemente veloz e flexível para armazenar dados históricos e figuras.

Os principais objetivos do MySQL são: velocidade, robustez e facilidade de uso.

## 12.4 Básico

'mysql' (algumas vezes é referenciado como "monitor") é um programa interativo que permite conectar a um servidor MySQL, executar consultas e ver os resultados. 'mysql' também pode ser usado em modo de execução em lote [batch mode]: você armazena suas consultas em um arquivo, então orienta ao 'mysql' executar os conteúdos do arquivo. Ambas maneiras de usar o 'mysql' são cobertos aqui. Para ver uma lista de opções fornecidas pelo 'mysql', invoque-o com a opção de comando '--help'

```
> mysql --help
```

## 12.5 Conectando e desconectando do servidor

Para conectar ao servidor, você irá necessitar de fornecer um nome de usuário do próprio Linux, quando o 'mysql' é invocado e uma senha. Se o servidor está funcionando em uma máquina diferente da que você está efetuando logon, também será necessário especificar o nome de máquina. Por exemplo:

```
> mysql -u root -p
Enter password: *****
```

Os caracteres `\*\*\*\*\*' representam sua senha; entre quando 'mysql' mostra o prompt 'Enter password'. Se der tudo certo, você deverá ver informações introdutórias seguidas pelo prompt 'mysql>'.

```
shell> mysql -u root
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 459 to server version: 3.22.20a-log
Type 'help' for help.
mysql>
```

O prompt significa que o 'mysql' está pronto para você entrar comandos. Depois de ter conectado com sucesso, você pode desconectar qualquer momento digitando 'QUIT' no prompt 'mysql>':

```
mysql> QUIT
Bye
```

Você também pode desconectar acionando as teclas CTRL+D.

## 12.6 Executando consultas

Tenha a certeza que você está conectado ao servidor, como foi discutido na seção anterior. Aqui é mostrado um comando simples que solicita ao servidor sua versão e a data atual.

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| version() | CURRENT_DATE |
+-----+-----+
| 3.22.20a-log | 1999-03-19 |
+-----+-----+
1 row in set (0.01 sec)
mysql>
```

Esta consulta ilustra vários pontos sobre o 'mysql':

- Um comando normalmente consiste de uma sentença SQL seguida por um ponto-e-vírgula.
- Quando você executa um comando, 'mysql' envia ao servidor para execução e mostra os resultados, então imprime outro 'mysql>' para indicar que está pronto para outro comando.
- 'mysql' mostra a saída de consultas como tabelas (linhas e colunas). A primeira linha contém os rótulos para as colunas. As linhas seguintes são os resultados da consulta.
- As palavras-chave podem ser entradas sem diferenciação de maiúsculas/minúsculas. As seguintes consultas são equivalentes:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

É permitido entrar com múltiplas sentenças em uma única linha. Simplesmente finalize cada sentença com um ponto-e-vírgula:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----+
| version() |
+-----+
| 3.22.20a-log |
+-----+
+-----+
| NOW() |
+-----+
| 1999-03-19 00:15:33 |
+-----+
```

Existe uma forma bastante livre para digitar sentenças. É permitido pular linhas entre sentenças:

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+-----+-----+
| USER() | CURRENT_DATE |
+-----+-----+
| joesmith@localhost | 1999-03-18 |
+-----+-----+
```

Se você decidir que você não quer executar um comando que você está em processo de digitação, cancele-o digitando '\c':

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

Lembre-se que para finalizar um comando, finalize-o com ponto-e-vírgula.

```
mysql> SELECT USER()
-> ;
+-----+
| USER() |
+-----+
| joesmith@localhost |
+-----+
```

## 12.7 Criando e usando um banco de dados

Agora que você já sabe entrar comandos, estamos no momento certo de acessar um banco de dados. Suponha que você deseje criar um banco de dados de contatos de clientes. Esta seção mostra como recuperar dados de tabelas e também:

- Como criar um banco de dados
- Como criar uma tabela
- Como carregar dados para a tabela
- Como recuperar dados da tabela de várias formas
- Como utilizar tabelas múltiplas

O banco de dados de clientes será simples (deliberadamente), mas não será difícil imaginar situações de mundo real no qual um tipo de banco de dados similar poderá ser usado.

Use a sentença 'SHOW' para checar quais bancos de dados existem atualmente no servidor:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
```

O banco de dados 'mysql' é necessário porque descreve os privilégios de acesso dos usuários. O banco de dados 'test' é geralmente fornecido como um espaço de trabalho para usuários para testes. Tente acessar o banco de dados teste:

```
mysql> USE test
Database changed
```

O banco de dados test permite que todos que tenham acesso a ele criar e remover as suas tabelas. Portanto, é recomendável utilizar um banco de dados separado para cada aplicação. Desta maneira, você (ou o administrador do banco de dados) poderá configurar o acesso de uma maneira mais apropriada.

### 12.7.1 Criando e selecionando um banco de dados

Para criar um banco de dados:

```
mysql> CREATE DATABASE contatos;
```

Sob sistemas Unix, nomes de banco de dados são sensíveis a caixa (diferente das palavras-chave SQL). Portanto, você deve sempre referenciar seu banco de dados como 'contatos', não como 'Contatos', 'CONTATOS' ou qualquer outra variante. Isto também é aplicável para nomes de tabelas. Para conectar no banco de dados:

```
mysql> USE contatos Database changed
```

## 12.7.2 Criando uma tabela

Neste ponto o banco de dados está vazio, como é mostrado no comando 'SHOW TABLES':

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

A parte complexa é decidir como será a estrutura de seu banco de dados: quais tabelas serão necessárias, e quais colunas terão cada uma. Este trabalho é um trabalho de análise de banco de dados, que foge do escopo deste curso. Suponhamos que este trabalho já foi realizado, e tenhamos a seguinte estrutura: Tabelas do sistema de contatos de clientes.

### Pessoa

Nome da coluna	Tipo	Nulo?	Descrição
cpf	char(11)	não	Cpf da pessoa
Nome	char(40)	não	Nome completo da pessoa
telefone1	char(15)	sim	Telefone residencial
telefone2	char(15)	sim	Telefone comercial
email	char(30)	sim	Email da pessoa
data_nascimento	Date	sim	Data de nascimento
Endereco	char(50)	sim	Endereço residencial
bairro	int unsigned	sim	Código do Bairro residencial
cidade	int unsigned	sim	Código da Cidade residencial
uf	char(2)	sim	Unidade de Federação da residência
cep	char(8)	sim	CEP da residência

### Bairro

Nome da coluna	Tipo	Nulo?	Descrição
cod_bairro	int unsigned	não	Identificador do bairro
descricao	char(40)	não	Descrição do bairro

### Cidade

Nome da coluna	Tipo	Nulo?	Descrição
cod_cidade	int unsigned	não	Identificador da cidade
descricao	char(40)	não	Descrição da cidade

### Empresa

Nome da coluna	Tipo	Nulo?	Descrição
cgc	char(20)	não	Cgc/cnpj
nome_fantasia	char(40)	não	Nome fantasia
razao_social	char(40)	sim	Razão social
responsavel	char(20)	sim	Cpf do responsável pela empresa
site_internet	char(50)	sim	Site Internet
email	char(30)	sim	Email do contato com a empresa
telefone	char(15)	sim	Telefone de contato
fax	char(15)	sim	Fax
endereco	char(50)	sim	Endereço
bairro	int unsigned	sim	Código do Bairro
cidade	int unsigned	sim	Código da Cidade

uf	char(2)	sim	Unidade de Federação
cep	char(8)	sim	CEP

### Contato

Nome da coluna	Tipo	Nulo?	Descrição
num_contato	int unsigned	não	Identifica o contato (valor sequencial)
tipo_cliente	char(1)	não	p=pessoa; e=empresa
cpf	char(20)	sim	Cpf (caso seja pessoa)
cgc	char(20)	sim	Cgc (caso seja empresa)
data	date	não	Data de contato com o cliente
observacao	text	sim	Observações, informações sobre o contato com o cliente

Através desta base de dados, além de você ter o cadastro de clientes, empresas ou pessoas, que poderá utilizá-la em uma mala direta. Note que com o uso de tabelas para cidade e bairro, você poderá utilizar informações para classificar área de maior concentração geográfica de seus clientes. Através da tabela de contatos você poderá cadastrar os contatos realizados com cada cliente, possuindo um histórico dos contatos realizados.

Você pode forçar que determinada coluna não aceita nulos, através da cláusula NOT NULL. Use um comando 'CREATE TABLE' para especificar o layout de cada tabela:

```
mysql> CREATE TABLE PESSOA (
-> cpf char(11) not null,
-> nome char(40),
-> telefone1 char(15),
-> telefone2 char(15),
-> email char(30),
-> data_nascimento date,
-> endereco char(50),
-> bairro int unsigned,
-> cidade int unsigned,
-> uf char(2),
-> cep char(8));
```

Agora que voce criou uma tabela, 'SHOW TABLES' deve produzir a saída:

```
mysql> SHOW TABLES;
+-----+
| Tables in contatos |
+-----+
| PESSOA |
+-----+
```

Para verificar que sua tabela foi criada da maneira que você espera, use o comando 'DESCRIBE':

```
mysql> DESCRIBE PESSOA;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| cpf | char(20) | YES | | NULL | |
| nome | char(40) | YES | | NULL | |
| telefone1 | char(15) | YES | | NULL | |
| telefone2 | char(15) | YES | | NULL | |
| email | char(30) | YES | | NULL | |
| data_nascimento | date | YES | | NULL | |
| endereco | char(50) | YES | | NULL | |
| bairro | int(11) | YES | | NULL | |
| uf | char(2) | YES | | NULL | |
| cep | char(8) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
```

Por enquanto, iremos apenas criar a tabela EMPRESA:

```
mysql> CREATE TABLE EMPRESA (
```

```

-> cgc char(20) not null,
-> nome_fantasia char(40) not null,
-> razao_social char(40),
-> responsavel char(11),
-> site_internet char(50),
-> email char(30),
-> telefone char(15),
-> fax char(15),
-> endereco char(50),
-> bairro int unsigned,
-> cidade int unsigned,
-> uf char(2),
-> cep char(8));

```

### 12.7.3 Carregando dados para uma tabela

Depois de criar suas tabelas, você precisa populá-las. Os comandos 'LOAD DATA' e o 'INSERT' são úteis para isto. Suponha que seus registros podem ser descritos como mostrado abaixo. (Observe que o *MySQL* aceita datas no formato 'YYYY-MM-DD'; isto pode ser diferente do que você está acostumado.

Tabela Pessoa

cpf	nome	telefone1	telefone2	email	data_nascimento	endereco	bairro	cidade	uf	cep
1111	João da Silva	111-1111	111-1112	<a href="mailto:joao@uol.com.br">joao@uol.com.br</a>	1984-11-11	Rua 11, no. 11	1	1	AM	11111111
2222	Maria Nunes	222-2222	222-2223	<a href="mailto:maria@bol.com.br">maria@bol.com.br</a>	1971-11-22	Rua 22, no. 22	2	2	GO	22222222
3333	José Pereira	333-3333	null	<a href="mailto:jose@sol.com.br">jose@sol.com.br</a>	1961-03-03	Rua 33, no. 33	2	2	BA	22222222

Desde que você está iniciando com uma tabela vazia, uma forma fácil de populá-la é criar um arquivo texto contendo uma linha para cada pessoa, então carregar o conteúdo do arquivo para a tabela com um único comando.

Você pode criar um arquivo texto 'pessoa.txt' contendo um registro por linha, com valores separados por tabs, e fornecendo dados na ordem na qual as colunas foram listadas no comando 'CREATE TABLE'. Para valores não aplicáveis ou não existentes (tal como segundo telefone), você pode usar valores 'NULL'. Para representar estes valores em seu arquivo texto, use '\N'. Por exemplo, o registro para José Pereira seria (onde há espaço em branco entre valores existe um caractere tab):

cpf	nome	telefone1	telefone2	email	data_nascimento	endereco	bairro	cidade	uf	cep
3333	José Pereira	333-3333	\N j	<a href="mailto:ose@sol.com.br">ose@sol.com.br</a>	1968-03-03	Rua 33, no. 33	2	2	BA	22222222

Para carregar o arquivo texto 'pessoa.txt' na tabela 'PESSOA', use este comando:

```
mysql> LOAD DATA LOCAL INFILE "pessoa.txt" INTO TABLE PESSOA;
```

Quando você desejar adicionar novos registro um por vez, o comando 'INSERT' é útil. Nesta forma mais simples, você fornece valores para cada coluna, na ordem na qual as colunas foram listadas no comando 'CREATE TABLE':

```
mysql> INSERT INTO PESSOA
-> VALUES ('4444', 'Ana Souza', '444-4444', NULL, 'ana@aol.com.br',
-> '1955-04-04', 'Rua 44, no. 44', 1, 1, 'GO', '44444444');
```



```
| 2222 | Maria Nunes          | 222-2222 | 222-2223 | maria@bol.com.br |
1971-11-22 | Rua 22,
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

Você pode combinar condições, por exemplo, para localizar pessoas que moram em Goiás e que nasceram depois de 1970:

```
mysql> SELECT * FROM PESSOA WHERE DATA_NASCIMENTO >= "1970-1-1" AND UF = "GO";
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| cpf      | nome          | telefone1 | telefone2 | email          |
data_nascimento | endereco
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| 2222     | Maria Nunes  | 222-2222  | 222-2223  | maria@bol.com.br |
1971-11-22 | Rua 22,
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

A consulta anterior usou o operador lógico 'AND'. Há também o operador 'OR':

```
mysql> SELECT * FROM PESSOA WHERE DATA_NASCIMENTO >= "1970-1-1" OR UF = "GO";
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| cpf      | nome          | telefone1 | telefone2 | email          |
data_nascimento | endereco
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| 1111     | João da Silva | 111-1111  | 111-1112  | joao@uol.com.br |
1984-11-11 | Rua 11,
| 2222     | Maria Nunes  | 222-2222  | 222-2223  | maria@bol.com.br |
1971-11-22 | Rua 22,
| 4444     | Ana Souza    | 444-4444  | NULL      | ana@aol.com.br  |
1955-04-04 | Rua 44,
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

### 12.7.4.3 Selecionando colunas particulares

Se você não desejar ver todas colunas de sua tabela, simplesmente forneça os nomes das colunas que você tiver interesse, separados por vírgulas. Por exemplo, se você desejar obter uma lista de nomes com seus respectivos e-mails, selecione as colunas 'nome' e 'email':

```
mysql> SELECT nome, email FROM PESSOA;
+-----+-----+
| nome          | email          |
+-----+-----+
| João da Silva | joao@uol.com.br |
| Maria Nunes   | maria@bol.com.br |
| José Pereira  | jose@sol.com.br  |
| Ana Souza     | ana@aol.com.br  |
+-----+-----+
```

Para selecionar quais estados (unidades de federação) as pessoas se encontram, use esta consulta:

```
mysql> SELECT UF FROM PESSOA;
+-----+
| UF  |
+-----+
| AM  |
| GO  |
| BA  |
| GO  |
+-----+
```

No entanto, note que a consulta simplesmente recupera o campo 'uf' de cada registro, e alguns deles aparecem mais do que uma vez. Para minimizar a saída, recupere cada saída de registro unicamente adicionando a palavra-chave 'DISTINCT':

```
mysql> SELECT DISTINCT UF FROM PESSOA;
+-----+
| UF    |
+-----+
| AM    |
| BA    |
| GO    |
+-----+
```

#### 12.7.4.4 Ordenando linhas

Você deve ter notado nos exemplos anteriores que as linhas resultantes são mostradas em uma ordem particular. No entanto, é mais fácil examinar a saída de uma consulta quando as linhas são ordenadas em uma forma significativa. Para ordenar um resultado, use uma cláusula 'ORDER BY'. Aqui estão os aniversários das pessoas, ordenadas pela data:

```
mysql> SELECT nome, data_nascimento FROM PESSOA ORDER BY data_nascimento;
+-----+-----+
| nome          | data_nascimento |
+-----+-----+
| Ana Souza     | 1955-04-04     |
| José Pereira  | 1961-03-03     |
| Maria Nunes   | 1971-11-22     |
| João da Silva | 1984-11-11     |
+-----+-----+
```

Para ordenar em ordem decrescente, adicione a palavra-chave 'DESC' (descendente) para o nome da coluna que você está ordenando:

```
mysql> SELECT nome, data_nascimento FROM PESSOA ORDER BY data_nascimento DESC;
+-----+-----+
| nome          | data_nascimento |
+-----+-----+
| João da Silva | 1984-11-11     |
| Maria Nunes   | 1971-11-22     |
| José Pereira  | 1961-03-03     |
| Ana Souza     | 1955-04-04     |
+-----+-----+
```

#### 12.7.4.5 Casamento de padrões

Para encontrar nomes começando com 'J':

```
mysql> SELECT cpf, nome FROM PESSOA WHERE nome LIKE "J%";
+-----+-----+
| cpf    | nome          |
+-----+-----+
| 1111   | João da Silva |
| 3333   | José Pereira  |
+-----+-----+
```

Para encontrar nomes que terminam com 'a':

```
mysql> SELECT cpf, nome FROM PESSOA WHERE nome LIKE "%a";
+-----+-----+
| cpf    | nome          |
+-----+-----+
| 1111   | João da Silva |
| 3333   | José Pereira  |
| 4444   | Ana Souza     |
+-----+-----+
```

#### 12.7.4.6 Contando linhas

Banco de dados são geralmente usados para responder a questão, "quanto um determinado tipo de dados ocorrem em uma tabela?". Por exemplo, você pode querer saber quantas pessoas você possui, ou quantas pessoas existem por um determinado tipo de estado ou bairro. Para contar o número total de pessoas:

```
mysql> SELECT COUNT(*) FROM PESSOA;
+-----+
```

```
| COUNT(*) |
+-----+
| 4 |
+-----+
```

Para selecionar o número de pessoas, agrupado por estado:

```
mysql> SELECT UF, COUNT(*) FROM PESSOA GROUP BY UF;
```

```
+-----+-----+
| UF | COUNT(*) |
+-----+-----+
| AM | 1 |
| BA | 1 |
| GO | 2 |
+-----+-----+
```

Note o uso de 'GROUP BY' para agrupar todos registros para cada estado.

#### 12.7.4.7 Restrições

Suponha que você deseje evitar que uma pessoa seja cadastrada duas vezes. Isto pode ser evitado elegendo uma coluna como chave. Em termos técnicos, devemos eleger uma chave primária. A chave primária por si deve refletir algum dado que não se repete. Por exemplo, se elegermos a coluna nome como chave primária podemos estar cometendo um erro, pois podem ter pessoas homônimas (mesmo nome). Por outro lado, se escolhermos a coluna cpf será uma boa escolha, pois sabemos que nunca teremos um cpf repetido.

Por definição, uma chave primária nunca deve admitir nulos. Para definir uma coluna como chave primária, você pode utilizar a cláusula PRIMARY KEY na criação da tabela:

```
CREATE TABLE PESSOA (
CPF CHAR(11) NOT NULL PRIMARY KEY,
...
```

Para definir uma coluna como chave primária, para uma tabela já existente, execute o comando ALTER TABLE com a cláusula PRIMARY KEY.

```
mysql> ALTER TABLE PESSOA ADD PRIMARY KEY (CPF);
mysql> ALTER TABLE EMPRESA ADD PRIMARY KEY (CGC);
```

#### 12.7.4.8 Cláusula auto\_increment

Na tabela PESSOA, você deve ter notado que as colunas CIDADE e BAIRRO contém valores inteiros. Na verdade, estas colunas fazem referência a outros valores de outras tabelas; respectivamente às tabelas

CIDADE e BAIRRO.

Execute os próximos comandos. Nestes comandos é introduzido a cláusula auto\_increment, que indica que o valor será auto incrementado quando for criado uma nova linha:

```
mysql> CREATE TABLE BAIRRO (
-> cod_bairro int unsigned not null auto_increment primary key,
-> descricao char(40) not null);
mysql> CREATE TABLE CIDADE (
-> cod_cidade int unsigned not null auto_increment primary key,
-> descricao char(40) not null);
mysql> CREATE TABLE CONTATO (
-> num_contato int unsigned not null auto_increment primary key,
-> tipo_cliente char(1),
-> cpf_ou_rg char(20),
-> cgc char(20),
-> data date,
-> observacao text);
```

Para popular as tabelas CIDADE, BAIRRO, EMPRESA e CONTATO, execute os comandos seguintes, que irão carregar valores para o banco de dados a partir dos arquivos cidade.txt, bairro.txt, empresa.txt e contato.txt respectivamente . Caso não tenha sido criado todos os arquivos , crie-os , conforme já foi explicado , e faça o carregamento dos mesmos no banco da seguinte maneira.

```
mysql> LOAD DATA LOCAL INFILE "bairro.txt" INTO TABLE BAIRRO;
mysql> LOAD DATA LOCAL INFILE "cidade.txt" INTO TABLE CIDADE;
mysql> LOAD DATA LOCAL INFILE "empresa.txt" INTO TABLE EMPRESA;
mysql> LOAD DATA LOCAL INFILE "contato.txt" INTO TABLE CONTATO;
```

Para experimentar a propriedade de auto incremento, execute:

```
mysql> select num_contato, tipo_cliente, cpf, cgc, data FROM CONTATO;
```

num_contato	tipo_cliente	cpf	cgc	data
1	P	1111	NULL	2000-04-21
2	P	2222	NULL	2000-04-21
3	E	NULL	121212	2000-04-21
4	P	1111	NULL	2000-04-22

```
mysql> INSERT INTO CONTATO ('P', '6666', NULL, '2000-04-23', 'Precisa do
programa do curso de mysql> select num_contato, tipo_cliente, cpf_ou_rg, cgc,
data FROM CONTATO;
```

num_contato	tipo_cliente	cpf_ou_rg	cgc	data
1	P	1111	NULL	2000-04-21
2	P	2222	NULL	2000-04-21
3	E	NULL	121212	2000-04-21
4	P	1111	NULL	2000-04-22
5	P	6666	NULL	2000-04-23

Note neste exemplo, que no comando INSERT, não foi necessário fornecer o valor da coluna num\_contato. Automaticamente o banco de dados verifica o maior valor existente e insere o valor seguinte para a coluna com propriedade de auto-incremento.

### 12.7.5 Usando múltiplas tabelas

É muito comum necessitarmos de informações que somente são obtidas cruzando informações entre tabelas. Por exemplo: Qual é o email pessoal dos responsáveis por cada empresa ? Qual é o dia de aniversário dos responsáveis pela empresa ? Quantos contatos existem por estado (unidade de federação) ? Na verdade, podemos criar inúmeras consultas, cruzando informações entre tabelas através de uma operação denominada junção [join]. Como exemplo, vamos obter o email dos responsáveis por empresas:

```
mysql> SELECT B.NOME_FANTASIA, A.NOME, A.EMAIL
-> FROM PESSOA A, EMPRESA B
-> WHERE A.CPF = B.RESPONSAVEL;
```

NOME_FANTASIA	NOME	EMAIL
Mabel	João da Silva	joao@uol.com.br
Arisco	Maria Nunes	maria@bol.com.br

Note:

- É utilizado as letras 'A' e 'B' como apelidos [alias] para as tabelas PESSOA e EMPRESA, respectivamente.
- Você pode referenciar colunas de tabelas simplesmente indicando o alias da tabela, 'ponto' e o nome da coluna. Por exemplo, B.NOME\_FANTASIA é a coluna NOME\_FANTASIA da tabela EMPRESA.

- Todo join tem uma condição de junção, especificada na parte de 'WHERE' da cláusula SELECT.

Neste caso, a condição de junção é que o responsável da empresa tenha o mesmo número de cpf da tabela de pessoas.

## 12.8 Alterando linhas

Para se alterar alguma linha, é utilizado o comando update, com a seguinte sintaxe:

```
UPDATE
SET =
WHERE =
```

Basicamente, é necessário apenas especificar a tabela em que se quer fazer a atualização; o campo e seu novo valor; e finalmente, a condição de filtragem. Por exemplo:

```
mysql> select cpf, nome
-> from pessoa;
+-----+-----+
| cpf | nome |
+-----+-----+
| 1111 | Joao da Silva |
| 2222 | Maria Nunes |
| 3333 | Jose da Silva |
| 4444 | Ana Souza |
+-----+-----+
mysql> update pessoa
-> set nome="Joao da Silva Pereira"
-> where cpf="1111";
mysql> select cpf, nome
-> from pessoa;
+-----+-----+
| cpf | nome |
+-----+-----+
| 1111 | Joao da Silva Pereira |
| 2222 | Maria Nunes |
| 3333 | Jose Pereira |
| 4444 | Ana Souza |
+-----+-----+
```

## 12.9 Deletando linhas

Para se alterar alguma linha, é utilizado o comando update, com a seguinte sintaxe:

```
DELETE FROM
WHERE =
```

Basicamente, é necessário apenas especificar a tabela em que se quer fazer a deleção; e a condição de filtragem. Deve se ter cuidado com este comando, pois o simples comando DELETE sem a cláusula WHERE, significa a deleção de todas as linhas da tabela, Por exemplo:

```
mysql> select cpf, nome
-> from pessoa;
+-----+-----+
| cpf | nome |
+-----+-----+
| 1111 | Joao da Silva |
| 2222 | Maria Nunes |
| 3333 | Jose da Silva |
| 4444 | Ana Souza |
+-----+-----+
mysql> delete from pessoa
-> where cpf="1111";
mysql> select cpf, nome
-> from pessoa;
+-----+-----+
| cpf | nome |
+-----+-----+
| 2222 | Maria Nunes |
| 3333 | Jose Pereira |
| 4444 | Ana Souza |
+-----+-----+
```

```
mysql> delete from pessoa;
mysql> select cpf, nome
-> from pessoa;
Empty set (0.00 sec)
```

## 12.10 Exemplos:

### Exemplo 1

Visualizar os salários dos empregados acrescidos de 10%.

```
SQL>SELECT last_name, salary * 1.10 FROM s_emp;
```

LAST_NAME	SALARY*1.10
Velasquez	2750
Ngao	1595
Nagayama	1540
Quick-To-See	1595
Ropeburn	1705
Urguhart	1320
Menchu	1375

### Exemplo 2

Visualizar os salários dos empregados acrescidos de R\$ 100,00.

```
SQL>SELECT last_name, salary + 100 FROM s_emp;
```

LAST_NAME	SALARY+100
Velasquez	2600
Ngao	1550
Nagayama	1500
Quick-To-See	1550
Ropeburn	1650
Urguhart	1300
Menchu	1350

### Exemplo 3

Exibir o nome do empregado, número e nome do departamento.

```
SQL>SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp, s_dept
WHERE s_emp.dept_id = s_dept.id;
```

LAST_NAME	DEPT_ID	NAME
Velasquez	50	Administration
Ngao	41	Operations
Nagayama	31	Sales
Quick-To-See	10	Finance
Ropeburn	50	Administration
Urguhart	41	Operations

Menchu	42	Operations
Biri	43	Operations
Catchpole	44	Operations
Havel	45	Operations
Magee	31	Sales
Giljum	32	Sales
Sedeghi	33	Sales

#### Exemplo 4

Exibir os funcionários cujo código de departamento seja igual ao código de departamento da funcionária 'Roberta'.

```
SQL> SELECT first_name, dept_id
      FROM s_emp
      WHERE dept_id =
            (SELECT dept_id
             FROM s_emp
             WHERE first_name = 'Roberta');
```

FIRST_NAME	DEPT_ID
Roberta	42
Akira	42
Vikram	42

#### Exemplo 5

Exibir o sobrenome dos funcionários que tenham o mesmo cargo do Smith.

```
SQL> SELECT first_name, title
      FROM s_emp
      WHERE title =
            (SELECT title
             FROM s_emp
             WHERE last_name = 'Smith');
```

LAST_NAME	TITLE
Maduro	Stock Clerk
Smith	Stock Clerk
Nozaki	Stock Clerk
Patel	Stock Clerk
Newman	Stock Clerk
Markarian	Stock Clerk
Chang	Stock Clerk
Patel	Stock Clerk
Dancs	Stock Clerk
Schwartz	Stock Clerk

10 rows selected.

## 12.10 Operadores:

### Operadores de Comparação Lógica

Operador	Significado
=	Igual a
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a

### Operadores Comparação SQL

Operador	Significado
BETWEEN...AND...	Entre dois valores (inclusive)
IN( <i>list</i> )	Satisfaz todas de uma lista de valores
LIKE	Satisfaz um padrão de caracter
IS NULL	É um valor nulo

### Operadores Lógicos

Operador	Significado
AND	Se ambas condições componentes retornarem TRUE, o resultado é TRUE.
OR	Se uma condições componentes retornarem TRUE, o resultado é TRUE.
NOT	Retorna a condição oposta.

### Operadores Lógicos de Negação

Operador	Significado
!=	Diferente de (VAX,UNIX,PC)
^=	Diferente de(IBM)
:=	
<>	Diferente de(todos os S.O.)
<	Menor que
<=	Menor ou igual a

### Operadores Comparação SQL

Operador	Significado
NOT BETWEEN...AND...	Não entre dois valores especificados
NOT IN( <i>list</i> )	Não na lista de valores especificados
NOT LIKE	Não como cadeia de comparação
IS NOT NULL	Não é um valor nulo