



O Portal do conhecimento

<http://apostilando.com>

## **Tutorial MySQL**

## Table of Contents

<u>Projeto Linux</u> .....	1
<u>Tutorial de MySQL</u> .....	3
<u>Conectando e desconectando do servidor</u> .....	6
<u>Entrando e perguntando</u> .....	18
<u>Exemplos de perguntas comuns</u> .....	22
<u>A fila da coluna com o número máximo</u> .....	24
<u>Máximo da coluna: por grupo e por valores</u> .....	29
<u>Usando chaves estrangeiras</u> .....	34
<u>Pesquisando em duas chaves</u> .....	36
<u>Criando e usando um banco de dados</u> .....	40
<u>Selecionando um banco de dados</u> .....	43
<u>Criando uma tabela</u> .....	49
<u>Carregando dados dentro uma tabela</u> .....	57

## Table of Contents

<u>Recuperando informação de uma tabela.....</u>	66
<u>Selecionando colunas particulares.....</u>	71
<u>Classificando filas.....</u>	76
<u>Cálculos de Data.....</u>	86
<u>Trabalhando com valores NULOS.....</u>	89
<u>Unindo Modelo.....</u>	99
<u>Contando filas.....</u>	107
<u>Usando mais que uma tabela.....</u>	118
<u>Usando MySQL em modo lote.....</u>	title

# Tutorial de MySQL

Esta apostila fornece uma introdução ao MySQL por mostrar ao usuário como criar

e gerenciar um banco de dados. MySQL é um programa interativo que permite você conectar

com um servidor de MySQL, perguntar e ver os resultados. Também veremos MySQL utilizado

em modo de lote: você coloca suas perguntas em um arquivo de antemão, e MySQL executa os

conteúdos do arquivo. Ambos os caminhos de usar o MySQL são descritos aqui.

Para ver uma lista de opções fornecidas por MySQL, invoque-o com `-help`:

```
shell> mysql --help
```

Esta apostila assume que o MySQL está instalado em sua máquina, e que um servidor de MySQL

está disponível para que possa conectar.

A apostila descreve o processo inteiro de como criar e usar um banco de dados.

## Conectando e desconectando do servidor

Para conectar ao servidor, você usualmente necessitará fornecer um usuário ao MySQL e,

mais provavelmente, uma senha. Se o servidor continua numa máquina que não tem onde se

registrar, também necessitará especificar um hostname. Sabendo os parâmetros próprios,

você poderá se conectar:

```
shell> mysql -h host -u user -p
```

```
Enter password: *****
```

O \*\*\*\*\* representa sua senha; entre com a senha quando MySQL exibir Enter

password: prompt.

Se daquela estação, você deveria ver alguma informação de introdução seguida

por uma mysql> prompt:

```
shell> mysql -h host -u user -p
```

```
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
```

Tipo "help" para ajuda.

```
mysql>
```

O prompt diz que você está pronto para entrar com os comandos.

Algumas instalações do MySQL permite aos usuários conectar com o servidor e continuar

como anfitrião local. Se isto é o caso em sua máquina, você deveria ser capaz de

conectar com o servidor ao invocar o MySQL sem quaisquer opções:

```
shell> mysql
```

Depois que está conectado, você pode desconectar a qualquer momento, é só digitar QUIT

```
no mysql> prompt:
```

```
mysql> QUIT
```

Bye

Você também pode desconectar por control-D.

Na maioria dos exemplos nas seguintes seções, assumem que você está conectado ao servidor.

Eles indicam isto por: mysql> prompt.



## Entrando e perguntando

Neste ponto, é mais importante descobrir como emitir perguntas do que criar tabelas,

carregar e recuperar dados. Esta seção descreve os princípios básicos de como entrar

com os comandos, usando várias perguntas. Você pode experimentar familiarizando com os

trabalhos do MySQL.

Aqui é um comando simples que pergunta ao servidor o número de sua versão e a data corrente.

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

```
+-----+-----+
| version() | CURRENT_DATE |
+-----+-----+
| 3.22.20a-log | 1999-03-19 |
```

```
+-----+-----+  
1 row in set (0.01 sec)  
mysql>
```

Esta pergunta ilustra várias coisas sobre MySQL:

\* Um comando normalmente consiste de uma declaração do SQL seguida por um ponto-e-vírgula.

(Há algumas exceções onde um ponto-e-vírgula não são necessário.

QUIT, mencionado anteriormente, é um deles.)

\* Quando você emite um comando, MySQL envia ao servidor para execução e exibe os resultados,

então exibe mysql>, para indicar que está pronto para outro comando.

\* MySQL exibe a saída da pergunta como uma tabela (filas e colunas).

A primeira fila contem rótulos às colunas. As demais filas são os resultados da pergunta.

Normalmente, os rótulos da coluna são os nomes das colunas que traz das tabelas do banco

de dados. Se você está recuperando o valor de uma expressão, em vez de uma coluna de tabela,

MySQL rotula a coluna usando a expressão que foi usada.

\* MySQL mostra quantas filas foram exibidas, e quanto tempo a pergunta levou para ser

executada, dando uma idéia ruda de performance do servidor. Esses valores são imprecisos

porque eles representam tempo de relógio de parede (não CPU ou tempo de máquina),

e porque eles são afetados por certos tipos de fatores. Palavras chaves podem ser inseridas

em qualquer caixa de entrada . As seguintes perguntas são equivalente:

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

```
mysql> select version(), current_date;
```

```
mysql> SeLeCt vErSiOn(), current_DATE;
```

Aqui é outra pergunta. Isto demonstra que você pode usar MySQL como uma simples calculadora:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
```

```
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
|    0.707107 |      25 |
+-----+-----+
```

Os comandos mostrados têm estado relativamente em declarações de linhas únicas e curtas.

Você pode até mesmo entrar com declarações múltiplas em uma única linha.

Somente termine cada uma com um ponto-e-vírgula.

```
mysql> SELECT VERSION(); SELECT NOW();
```

```
+-----+  
| version()      |  
+-----+  
| 3.22.20a-log  |  
+-----+
```

```
+-----+  
| NOW()          |  
+-----+  
| 1999-03-19 00:15:33 |  
+-----+
```

Um comando dado todo em uma única linha, assim como comandos compridos que requerem várias

linhas, não tem nenhum problema. MySQL determina que sua declaração termina por um

ponto-e-vírgula, e não, o fim de uma linha de entrada.

Aqui é uma simples declaração de linha múltipla:

```
mysql> SELECT
```

```
    -> USER()
```

```
    -> ,
```

```
    -> CURRENT_DATE;
```

```
+-----+-----+
| USER()          | CURRENT_DATE |
+-----+-----+
| joesmith@localhost | 1999-03-18   |
+-----+-----+
```

Neste exemplo, note como o prompt muda de `mysql>` para `->` depois que entra com a pergunta na

primeira linha de uma linha múltipla. MySQL indica que não tem uma declaração completa e

está esperando o resto. O prompt é seu amigo, porque ele fornece retorno valioso.

Se você usa aquele retorno, você sempre estará atento do que MySQL está esperando.

Se você decide não executar mais nenhum comando que está no processo de entrada, cancele

isto, digitando `\c`

```
mysql> SELECT
```

```
    -> USER()
```

```
    -> \c
```

```
mysql>
```

Note também aqui, o prompt. Isto muda o retorno para `mysql>` depois que você digita `\c`,

MySQL indica o retorno e que está pronto para um novo comando.

A seguinte tabela mostra cada um dos prompts que pode ser visto e resume que estado

o MySQL está:

Prompt

Significado

mysql>

Pronto para um novo comando

->

Esperando para próxima linha de linha múltipla de comando

'>

Esperando para próxima linha, colecionando um fio que começa com uma citação única ( "'")

">

Esperando para próxima linha, colecionando um fio que começa com uma citação duplicada ( """)



Linhas múltiplas de declarações comuns ocorrem por acaso, quando você pretende emitir

um comando em uma única linha, mas esquece o ponto-e-vírgula. Neste caso, MySQL espera

para mais entrada:

```
mysql> SELECT USER()
```

```
->
```

Se isto acontecer (você pensa que entrou com uma declaração mas a unicamente

a resposta é um -> prompt), mas provavelmente, MySQL está esperando o ponto-e-vírgula.

Entre com um ponto-e-vírgula para completar a declaração, e MySQL executará:

```
mysql> SELECT USER();
```

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

O '>' e '>' prompts ocorrem durante coleção de fio. No MySQL, pode escrever fios

circundados por um "'" ou "" caracteres (por exemplo, 'hello' ou "adeus"), e MySQL

deixa que você entre com os fios que transpõem linhas múltiplas. Quando você vê um '>' ou

> prompt, este meio que tem de entrar com uma linha contendo um fio que começa com

caracteres: "'" ou "", não tem ainda que entrar com a citação que termina o fio.

Seria bom se está realmente entrando com uma linha múltipla de fio, mas como provavelmente

é isso? Mais freqüentemente, o '>' e '>' prompts indicam que você descuidou e deixou fora um

caracter de citação. Por exemplo:

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;  
"  
>
```

Se você ao entrar com a declaração SELECT, ele acessa, volta e se você ficar esperando

o resultado e nada acontecer, ao invés de admirar, note a pista fornecida pelo '>' prompt.

Isto diz que MySQL está esperando ver o final de um fio.

Neste ponto, o que deve ser feito? É simples, é só cancelar o comando. Entretanto,

não pode ser somente \c, porque MySQL interpreta isto, como a separação do fio que está

coleccionando. Ao invés, de entrar somente com o fechamento de caracter (assim MySQL saberá que

você terminou o fio), entre com "\c:

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;
```

```
"> "\c
```

```
mysql>
```

As mudanças do prompt retornam para `mysql>`, indicando que MySQL está pronto para um novo

comando.

Isto é importante, para saber o que `'>` e `">` prompts expressam, porque se você entrar

incorretamente com um fio, mais além, você digita a vontade e parece que MySQL ignorou,

incluindo uma linha contendo `QUIT`, isto pode ser completamente confuso, se você não sabe

o que necessita para fornecer a citação de término antes que possa cancelar o comando corrente.

## Exemplos de perguntas comuns

Aqui segue exemplos de como resolver alguns dos problemas mais comuns do MySQL.

Alguns dos exemplos usam a tabela de compras, coloque os preços de cada artigo (número de item)

de cada negociante. Supondo que cada negociante tem um preço fixo por artigo, então

(item, negociante) é uma chave primária aos registros.

Você pode criar a tabela de exemplo como:

```
CREATE TABLE shop (  
  article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,  
  dealer CHAR(20) DEFAULT '' NOT NULL,  
  price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,  
  PRIMARY KEY(article, dealer));
```

```
INSERT INTO shop VALUES  
(1, 'A', 3.45), (1, 'B', 3.99), (2, 'A', 10.99), (3, 'B', 1.45), (3, 'C', 1.69),  
(3, 'D', 1.25), (4, 'D', 19.95);
```

Assim os dados de exemplo estarão:

```
SELECT * FROM shop
```

```
+-----+-----+-----+  
| article | dealer | price |  
+-----+-----+-----+  
|    0001 | A      |  3.45 |  
|    0001 | B      |  3.99 |  
|    0002 | A      | 10.99 |  
|    0003 | B      |  1.45 |  
|    0003 | C      |  1.69 |
```

```
| 0003 | D | 1.25 |  
| 0004 | D | 19.95 |  
+-----+-----+-----+
```

O valor máximo de uma coluna

"Qual é o artigo que tem o preço mais alto?"

```
SELECT MAX(article) AS article FROM shop
```

```
+-----+  
| article |  
+-----+  
|      4 |  
+-----+
```





## A fila da coluna com o número máximo

"Encontre o número do negociantes, e avalie quem tem o artigo mais caro."

No ANSI SQL isto é facilmente feito com um sub-query:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop)
```

No MySQL (ainda não faz uma sub-seleção) somente faz isto em dois passos:

1. Obtem o valor máximo e avalia a tabela com uma declaração SELECT.
2. Usando este valor compila a pergunta real:

```
SELECT article, dealer, price
```

```
FROM shop  
WHERE price=19.95
```

Outra solução está em classificar todas filas decrescentes por preço e unicamente obter

uma fila usando o MySQL cláusula de LIMIT:

```
SELECT article, dealer, price  
FROM shop  
ORDER BY price DESC  
LIMIT 1
```

Note: Se há vários artigos caros, a solução de LIMIT mostra unicamente um deles.

## Máximo da coluna: por grupo e por valores

"Qual é o preço mais alto por artigo?"

```
SELECT article, MAX(price) AS price
FROM shop
GROUP BY article
```

```
+-----+-----+
| article | price |
+-----+-----+
| 0001 | 3.99 |
| 0002 | 10.99 |
| 0003 | 1.69 |
| 0004 | 19.95 |
```

```
+-----+-----+
```

As filas com grupos de campos de valor máximo

"Para cada artigo, encontre o(s) negociante(s) com o preço mais caro."

No ANSI SQL, pode fazer isto com um sub-query:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
```

No MySQL é melhor ser feito em vários passos:

1. Obtenha a lista de artigo e preço máximo.
2. Para cada artigo obtenha as filas correspondentes que têm o preço máximo armazenado.

Isto pode ser facilmente feito com uma tabela temporária:

```
CREATE TEMPORARY TABLE tmp (  
    article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,  
    price DOUBLE(16,2) DEFAULT '0.00' NOT NULL);  
  
LOCK TABLES article read;  
  
INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;
```

```
SELECT article, dealer, price FROM shop, tmp

WHERE shop.article=tmp.article AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

Se você usar uma tabela TEMPORÁRIA, você deve também fechar o "tmp" tabela.

"Isto pode ser feito com uma pergunta única?"

Sim, mas pode usar um truque ineficiente que é chamado de "MAX-CONCAT trick":

```
SELECT article,
           SUBSTRING( MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 7) AS dealer,
           0.00+LEFT(      MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 6) AS price
```

```
FROM shop  
GROUP BY article;
```

article	dealer	price
0001	B	3.99
0002	A	10.99
0003	C	1.69
0004	D	19.95

## Usando chaves estrangeiras

Você não necessita de chaves estrangeiras para unir 2 tabelas.

O MySQL não faz a checagem de certificar que as chaves da tabela são referências e isto

não é feito automaticamente apagando as filas da tabela com uma definição de chave

estrangeira. Se você usa as chaves normais, ele trabalhará perfeitamente.

```
CREATE TABLE persons (  
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    name CHAR(60) NOT NULL,  
    PRIMARY KEY (id)  
);
```



```
CREATE TABLE shirts (  
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,  
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,  
    owner SMALLINT UNSIGNED NOT NULL REFERENCES persons,  
    PRIMARY KEY (id)  
);
```

```
INSERT INTO persons VALUES (NULL, 'Antonio Paz');
```

```
INSERT INTO shirts VALUES  
(NULL, 'polo', 'blue', LAST_INSERT_ID()),  
(NULL, 'dress', 'white', LAST_INSERT_ID()),  
(NULL, 't-shirt', 'blue', LAST_INSERT_ID());
```

```
INSERT INTO persons VALUES (NULL, 'Lilliana Angelovska');
```

```
INSERT INTO shirts VALUES  
(NULL, 'dress', 'orange', LAST_INSERT_ID()),  
(NULL, 'polo', 'red', LAST_INSERT_ID()),  
(NULL, 'dress', 'blue', LAST_INSERT_ID()),  
(NULL, 't-shirt', 'white', LAST_INSERT_ID());
```

```
SELECT * FROM persons;
```

```
+----+-----+  
| id | name                |  
+----+-----+  
| 1  | Antonio Paz        |  
| 2  | Lilliana Angelovska |  
+----+-----+
```

```
SELECT * FROM shirts;
```

```
+-----+-----+-----+-----+
| id | style  | color | owner |
+-----+-----+-----+-----+
| 1  | polo   | blue  | 1     |
| 2  | dress  | white | 1     |
| 3  | t-shirt| blue  | 1     |
| 4  | dress  | orange| 2     |
| 5  | polo   | red   | 2     |
| 6  | dress  | blue  | 2     |
| 7  | t-shirt| white | 2     |
+-----+-----+-----+-----+
```

```
SELECT s.* FROM persons p, shirts s
WHERE p.name LIKE 'Lilliana%'
AND s.owner = p.id
```

```
AND s.color <> 'white';
```

```
+-----+-----+-----+-----+
| id | style | color | owner |
+-----+-----+-----+-----+
|  4 | dress | orange |    2 |
|  5 | polo  | red    |    2 |
|  6 | dress | blue   |    2 |
+-----+-----+-----+-----+
```

## Pesquisando em duas chaves

MySQL ainda não faz, pesquisa com duas chaves diferentes combinadas com OR

(Pesquisando com uma chave OR em diferentes partes é bem melhor):

```
SELECT field1_index, field2_index FROM test_table WHERE field1_index = '1'
```

```
OR field2_index = '1'
```

No momento, você pode resolver isto com muita eficiência, usando uma tabela TEMPORÁRIA;

Este tipo de otimização é também muito boa se você está usando muitas perguntas

complicadas onde o servidor do SQL faz as otimizações no pedido errado.

```
CREATE TEMPORARY TABLE tmp
```

```
SELECT field1_index, field2_index FROM test_table WHERE field1_index =  
'1';
```

```
INSERT INTO tmp
```

```
SELECT field1_index, field2_index FROM test_table WHERE field2_index =  
'1';
```

```
SELECT * from tmp;
```

```
DROP TABLE tmp;
```

Acima está o caminho para resolver a pergunta com efeito de união de duas perguntas.

## Criando e usando um banco de dados

Agora que você já sabe como entrar com os comandos, isto é, como acessar um banco de dados.

Suponha que você tenha vários animais de estimação em sua casa e tem vontade de

guardar vários tipos de informações sobre eles. Você pode fazer isto, criando tabelas

para guardar seus dados e carregá-las com informações desejada. Então você pode responder

várias perguntas de diferentes tipos sobre seus animais recuperando os dados das tabelas.

Esta seção mostra:

- \* Como criar um banco de dados
- \* Como criar uma tabela
- \* Como carregar dados dentro a tabela
- \* Como recuperar dados da tabela em vários caminhos
- \* Como usar tabelas múltiplas

A vantagem de um banco de dados é simples, pense em situações do mundo real em que um banco

de dados pôde ser utilizado. Por exemplo, um banco de dados pode ser utilizado por um

fazendeiro para guardar dados da criação, ou por um médico para guardar dados de registros

dos pacientes.

Use a declaração SHOW para descobrir que bancos de dados existem no servidor:

```
mysql> SHOW DATABASES;
```

```
+-----+  
| Database |  
+-----+  
| mysql    |  
| test     |  
| tmp      |
```



+-----+

A lista de bancos de dados está provavelmente diferente em sua máquina, mas o MySQL

testa os bancos de dados que estão provavelmente dentro dele. O banco de dados mysql é

requerido porque isto descreve os privilégios de acesso do usuário. O banco de dados test

é freqüentemente fornecido como um workspace aos usuários para tentar coisas fora.

Se o banco de dados de test existe, tenta acessar:

```
mysql> USE test
```

```
Database changed
```

Note que USE, QUIT, não requerem um ponto-e-vírgula. (Você pode terminar tais declarações

com um ponto-e-vírgula) A declaração USE também é especial em outro caminho: isto deve-se

a dar em uma única linha.

Você pode usar o banco de dados test com os exemplos que seguem, mas se você criar algo no

banco de dados, pode ser removido por qualquer um que tenha acesso. Por esta razão, você

deveria pedir ao seu administrador do MySQL, permissão para usar um banco de dados.

Suponha que você deseja chamar seu menagerie. O administrador necessita executar um comando:

```
mysql> GRANT ALL ON menagerie.* TO your_mysql_name;
```

onde your\_mysql\_name é o nome do usuário do MySQL designado a você.

## Selecionando um banco de dados

Se o administrador criar para você seu banco de dados com suas permissões, você pode

começar a usá-lo. De outro modo, você mesmo necessita criá-lo.

```
mysql> CREATE DATABASE menagerie;
```

Sob Unix, nomes de banco de dados são caso sensível (diferente de palavras chaves do SQL),

assim você deve sempre referir para seu banco de dados como `menagerie`, não como `Menagerie`,

`MENAGERIE` ou alguma outra variante. Isto também é vale para nomes de tabela. (Sob o Windows,

esta restrição não aplica, embora você deva referir-se a bancos de dados e tabelas usando o

mesmo lettercase por toda pergunta dada.)

Criando um banco de dados, você não o seleciona para uso, deve fazer isto

explicitamente.

Para fazer o menagerie do banco de dados corrente, use este comando:

```
mysql> USE menagerie
```

```
Database changed
```

Seu banco de dados necessita ser criado uma unicamente vez, mas deve selecioná-lo o

uso a cada vez que começar uma sessão do MySQL. Você pode fazer isto ao usar uma declaração

de USE como mostrada acima. Alternativamente, você pode selecionar o banco de dados

na linha de comando, quando você invocar MySQL. Somente vai especificar seu nome depois

de quaisquer parâmetros de conexão que poderá ser necessário fornecer. Por exemplo:

```
shell> mysql -h host -u user -p menagerie
```

```
Enter password: *****
```

Note que `menagerie` não é sua senha no comando. Se você deseja fornecer sua senha na

linha de comando depois da opção `-p`, você deve fazer não intervindo espaço (e.g., como

`-pmypassword`, não como `-p mypassword`). Entretanto, colocar sua senha na linha de comando não

é recomendada, porque impede ações que expõe a outros usuários registrados em sua máquina.

## Criando uma tabela

Criar um banco de dados é a parte mais fácil, mas neste ponto ele está vazio,

com SHOW TABLES mostrará:

```
mysql> SHOW TABLES;
```

```
Empty set (0.00 sec)
```

A parte mais dura é decidir que estrutura o seu banco de dados deverá ter: que tabelas

você necessitará, e que colunas deve ter em cada uma delas.

Você desejará uma tabela que contém um registro para cada um de seus animais de estimação.

Isto pode ser chamado de tabela de animal de estimação, e isto deveria conter, no mínimo,

o nome de cada animal. O nome por si mesmo não está muito interessante, a tabela deveria

conter outra informação. Por exemplo, se mais de uma pessoa em sua família tem animais de

estimação, você pôde desejar listar o dono de cada animal. Pode registrar alguma informação

básica, tal como, espécie e sexo.

Como sobre idade? É interessante, mas isto, não é uma boa coisa para armazenar em um banco

de dados. Ao invés da idade, será melhor armazenar um valor fixo, como a data de nascimento.

Então, sempre que você necessitar da idade, você pode calcular isto como a diferença entre

a data corrente e a data de nascimento. MySQL fornece funções aritmética de data.

Armazenando a data de nascimento em vez da idade teremos outras vantagens:

\* Você pode usar o banco de dados para tarefas, tal como gerar lembretes para os próximos

aniversários do animal de estimação. (Se você pensa que este tipo de pergunta é tola, note

que é a mesma pergunta que pôde ser feita, no contexto de um banco de dados de negócios,

identifica clientes que fazem aniversário.)

\* Você pode calcular a idade em relação a datas que não são corrente. Por exemplo,

se você armazena data da morte, você pode facilmente calcular a idade que animal de

estimação estava, quando morreu.

Você pode provavelmente pensar de outros tipos de informação que poderia ser útil na

tabela do animal de estimação, mas são suficientes agora: nome, dono, espécie, sexo,

nascimento e morte.

Use a declaração CREATE TABLE para especificar o layout de sua tabela:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),  
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

VARCHAR é uma boa escolha ao nome, dono e de espécie porque os valores da coluna variam



em comprimento. Os comprimentos daquelas colunas não são do mesmo tamanho. Você pode

escolher qualquer comprimento de 1 até 255, qualquer que pareça razoável. (Se fez uma

escolha pobre e mais tarde você necessita um campo mais longo, MySQL fornece uma declaração

de ALTER TABLE .)

O sexo do animal pode ser representado com uma variedade de caminhos, por exemplo,

"m" e "f", ou talvez "masculino" e "feminino". É simples é só usar os caracteres únicos

"m" e "f".

O uso de dados tipo data para ao nascimento e morte é honestamente uma escolha óbvia.

Agora que você criou uma tabela, SHOW TABLE deveria produzir alguma saída:

```
mysql> SHOW TABLES;
```

```
+-----+
```

```
| Tables in menagerie |
+-----+
| pet                |
+-----+
```

Para verificar que sua tabela foi criada no caminho que você criou, use a declaração

DESCRIBE:

```
mysql> DESCRIBE pet;
```

```
+-----+-----+-----+-----+-----+-----+
| Field  | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name   | varchar(20)  | YES  |     | NULL    |       |
| owner  | varchar(20)  | YES  |     | NULL    |       |
| species| varchar(20)  | YES  |     | NULL    |       |
```

sex	char(1)	YES		NULL		
birth	date	YES		NULL		
death	date	YES		NULL		
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+

Você pode usar DESCRIBE a qualquer hora, por exemplo, se você esquecer os nomes das

colunas em sua tabela.

## Carregando dados dentro uma tabela

Depois de ter criado sua tabela, você necessita inserir os dados. As declarações: LOAD

DATA e INSERT são úteis.

Suponha que os registros do animal de estimação podem ser descritos como mostrados a baixo.

(Observe que MySQL espera datas em formato: YYYY-MM-DD; pode ser diferente do que

é utilizado.)

Nome

Dono

Espécie

Sexo

Nascimento

Morte

Fofo

Harold

gato

f

1993-02-04

Claws

Gwen

gato

m

1994-03-17

Buffy

Harold

cão

f

1989-05-13

Presa

Benny

cão

m

1990-08-27

Bowser

Diane

cão

m

1998-08-31

1995-07-29

Chirpy

Gwen

pássaro

f

1998-09-11

Whistler

Gwen

pássaro

1997-12-09

Magro

Benny

cobra

m

1996-04-29

Se você está começando com uma tabela vazia, um caminho fácil para preencher é criar

um arquivo de texto contendo uma fila para cada um de seus animais, então carregue os

conteúdo do arquivo dentro da tabela com uma declaração única.

Você pode criar um arquivo de texto "pet.txt" contendo um registro por linha, com

valores separados por tabulações, conforme as colunas foram listadas na declaração de

CREATE TABLE. Para valores desconhecidos (tal como sexos desconhecidos, ou datas

de morte para animais que estão ainda vivos), você pode usar valores NULL. Para

representá-los em seu arquivo de texto, use \N. Por exemplo, o registro do pássaro:

Whistler

Gwen



Pássaro

\N

1997-12-09

\N

Para carregar o arquivo de texto "pet.txt" dentro da tabela de animal de estimação,

use este comando:

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

Você pode especificar o separador de valor da coluna e a terminação da linha

explicitamente na declaração de LOAD DATA, se você desejar, mas os defaults são tabulações

e mudança de linha. Essas declarações são suficientes para ler o arquivo "pet.txt"

propriamente.

Cada vez que você desejar inserir um novo registro, a declaração INSERT é útil.

Em seu formulário, você fornece valores para cada coluna, na forma em que as colunas

foram listadas na declaração de CREATE TABLE. Suponha Diane obtem um novo hamster

especificado Puffball. Você podia inserir um novo registro usando uma declaração INSERT:

```
mysql> INSERT INTO pet
```

```
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Note que o fio e valores de data são especificados como citados. Com INSERT, você

também pode inserir NULL para representar um valor desconhecido. Você não usa \N,

conforme faz com LOAD DATA.

Deste exemplo, você deveria ser capaz de ver que pode ser muito mais trabalhoso

inserir seus registros inicialmente usando várias declarações de INSERT em vez de

uma única declaração de LOAD DATA.

## Recuperando informação de uma tabela

A declaração `SELECT` é utilizada para puxar informação de uma tabela. O formulário

geral da declaração é:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

`what_to_select` indica o que você deseja ver. Isto pode ser uma lista de colunas, ou

\* para para indicar todas colunas.

`which_table` indica a tabela de que você deseja recuperar os dados. A cláusula `WHERE` é

opcional.

Se for apresentado, `conditions_to_satisfy` especifica as condições que as

filas devem  
satisfazer para recuperação qualificada.

Selecionando todos os dados

A forma simples de recuperar tudo de uma tabela é SELECT:

```
mysql> SELECT * FROM pet;
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner | species | Sex   | birth   | death   |

```

```

+-----+-----+-----+-----+-----+-----+
| Fluffy | Harold | cat   | f   | 1993-02-04 | NULL |
| Claws  | Gwen  | cat   | m   | 1994-03-17 | NULL |
| Buffy  | Harold | dog   | f   | 1989-05-13 | NULL |
| Fang   | Benny | dog   | m   | 1990-08-27 | NULL |
| Bowser | Diane | dog   | m   | 1998-08-31 | 1995-07-29 |
| Chirpy | Gwen  | bird  | f   | 1998-09-11 | NULL |
| Whistler | Gwen | bird  | NULL | 1997-12-09 | NULL |
| Slim   | Benny | snake | m   | 1996-04-29 | NULL |
| Puffball | Diane | hamster | f   | 1999-03-30 | NULL |
+-----+-----+-----+-----+-----+-----+

```

Este comando `SELECT` é útil se você desejar rever sua tabela inteira. Isto acontece

quando a saída revela um erro em seu arquivo de dados: Bowser parece nascer depois

que ele morreu! Consultando seus papéis do pedigree originais, você encontra que o

ano de nascimento correto é 1989, não 1998.

Existe no mínimo um par de caminhos para fixar:

\* Edite o arquivo "pet.txt" para corrigir o erro, então esvazie a tabela e recarregue

usando DELETE LOAD DATA:

```
mysql> DELETE FROM pet;
```

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

Entretanto, se você fez isto, você também deve dar uma reentrada no registro do Puffball.

\* Fixe unicamente o registro errôneo com uma declaração de UPDATE :

```
mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Bowser";
```

Como mostrado acima, isto é fácil para recuperar uma tabela inteira.

Selecionando filas particulares

Você pode selecionar unicamente filas particulares de sua tabela. Por exemplo, se

you deseja verificar a mudança que você fez na data de nascimento do Bowser, selecione

o registro do Bowser:

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Bowser | Diane | dog      | m    | 1989-08-31 | 1995-07-29 |
```



```
+-----+-----+-----+-----+-----+-----+
```

A saída confirma que agora o ano está corretamente registrado como 1989 e não 1998.

Comparações de fio são normalmente casos insensíveis, assim você pode especificar o

nome como "browser", "BROWSER", etc. A pergunta é a mesma.

Você pode especificar condições em qualquer coluna, não somente no nome. Por exemplo,

se você deseja saber que animais nasceram depois 1998, teste a coluna de nascimento:

```
mysql> SELECT * FROM pet WHERE birth >= "1998-1-1";
```

```
+-----+-----+-----+-----+-----+-----+
| name      | owner  | species | sex   | birth      | death  |
+-----+-----+-----+-----+-----+-----+
| Chirpy    | Gwen  | bird    | f     | 1998-09-11 | NULL   |
| Puffball  | Diane | hamster | f     | 1999-03-30 | NULL   |
```

```
+-----+-----+-----+-----+-----+-----+
```

Você pode combinar condições, por exemplo, para localizar os cães femininos:

```
mysql> SELECT * FROM pet WHERE species = "dog" AND sex = "f";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death   |
+-----+-----+-----+-----+-----+-----+
| Buffy  | Harold | dog     | f     | 1989-05-13 | NULL    |
+-----+-----+-----+-----+-----+-----+
```

A pergunta precedente usa operador lógico AND. Existe também o operador OR:

```
mysql> SELECT * FROM pet WHERE species = "snake" OR species = "bird";
```

```
+-----+-----+-----+-----+-----+-----+
```

name	owner	species	sex	birth	death
Chirpy	Gwen	bird	f	1998-09-11	NULL
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL

```
mysql> SELECT * FROM pet WHERE (species = "cat" AND sex = "m")
      -> OR (species = "dog" AND sex = "f");
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

AND e OR podem ser misturados. Isto é uma boa idéia para usar parênteses para indicar

que condições deveriam ser agrupados:

## Selecionando colunas particulares

Se você não deseja ver filas inteiras de sua tabela, somente especifique as colunas

em que está interessado, separadas por vírgulas. Por exemplo, se você deseja saber

quando seus animais nasceram, selecionam o nome e colunas de nascimento:

```
mysql> SELECT name, birth FROM pet;
```

```
+-----+-----+
| name   | birth   |
+-----+-----+
| Fluffy | 1993-02-04 |
| Claws  | 1994-03-17 |
| Buffy  | 1989-05-13 |
| Fang   | 1990-08-27 |
| Bowser | 1989-08-31 |
```

```
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |

| Puffball | 1999-03-30 |
+-----+-----+
```

Para descobrir quem possui animais de estimação, use esta pergunta:

```
mysql> SELECT owner FROM pet;
```

```
+-----+
| owner |
+-----+
| Harold |
| Gwen   |
| Harold |
```

```
| Benny |  
| Diane |  
| Gwen  |  
| Gwen  |  
| Benny |  
| Diane |  
+-----+
```

Entretanto, note que a pergunta simplesmente recupera o campo do dono de cada registro,

e algum deles aparece mais de uma vez. Para diminuir a saída, recuperar e registrar

somente uma vez use a palavra chave DISTINCT:

```
mysql> SELECT DISTINCT owner FROM pet;
```

```
+-----+  
| owner |
```

```
+-----+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+-----+
```

Você pode usar a cláusula WHERE para combinar a seleção da fila com seleção da coluna.

Por exemplo, para obter datas de nascimento de cães e gatos unicamente:

```
mysql> SELECT name, species, birth FROM pet
      -> WHERE species = "dog" OR species = "cat";
```

```
+-----+-----+-----+
| name   | species | birth   |
+-----+-----+-----+
```



```
| Fluffy | cat      | 1993-02-04 |
| Claws  | cat      | 1994-03-17 |
| Buffy  | dog      | 1989-05-13 |
| Fang   | dog      | 1990-08-27 |
| Bowser | dog      | 1989-08-31 |
+-----+-----+-----+
```

## Classificando filas

Você pode notar nos exemplos, que as filas de resultado são exibidas sem nenhum

modo particular. Entretanto, isto é mais fácil examinar a saída da pergunta quando

as filas são classificadas em algum caminho significativo. Para classificar um resultado,

use ORDER BY.

Aqui são os aniversários dos animais, classificados por data:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

```
+-----+-----+
| name   | birth   |
+-----+-----+
| Buffy  | 1989-05-13 |
```

```
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+-----+-----+
```

Para classificar um pedido em ordem decrescente, use a palavra chave DESC com o nome

da coluna:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
```

```
+-----+-----+
```

name	birth
Puffball	1999-03-30
Chirpy	1998-09-11
Whistler	1997-12-09
Slim	1996-04-29
Claws	1994-03-17
Fluffy	1993-02-04
Fang	1990-08-27
Bowser	1989-08-31
Buffy	1989-05-13

Você pode classificar colunas múltiplas. Por exemplo, para classificar por tipo de

animal e por data de nascimento, o tipo de animal deve ser classificado pelo o

mais jovem, use a seguinte query:

```
mysql> SELECT name, species, birth FROM pet ORDER BY species, birth DESC;
```

```
+-----+-----+-----+
| name   | species | birth   |
+-----+-----+-----+
| Chirpy | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws  | cat     | 1994-03-17 |
| Fluffy | cat     | 1993-02-04 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
| Buffy  | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
```

```
| Slim      | snake    | 1996-04-29 |  
+-----+-----+-----+
```

Note que a palavra chave DESC aplica unicamente à coluna (nascimento);  
valores de

espécie são classificados por ordem crescente.

## Cálculos de Data

MySQL fornece várias funções que pode fazer cálculos em datas, por exemplo, para

calcular idades ou extrair partes das datas.

Para determinar quantos anos tem seus animais de estimação, ele pega a diferença entre

a data de nascimento e a data corrente. Converte as duas datas para dias, tomam a

diferença, e divide por 365 (o número de dias em um ano):

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 FROM pet;
```

```
+-----+-----+
| name      | (TO_DAYS(NOW())-TO_DAYS(birth))/365 |
+-----+-----+
| Fluffy    | 6.15 |
| Claws     | 5.04 |
```

Buffy		9.88	
Fang		8.59	
Bowser		9.58	
Chirpy		0.55	
Whistler		1.30	
Slim		2.92	
Puffball		0.00	
+-----+-----+-----+-----+-----+			

Há algumas coisas que podem ser melhoradas. Primeiro, o resultado pode ser examinado

mais facilmente se as filas que forem apresentadas em alguma ordem. Segundo, o

título da coluna de idade não está significativo.

O primeiro problema pode ser tratado por inserindo a cláusula ORDER BY ao nome,

para classificar a saída por nome. Para lidar com o título da coluna, fornece um



nome à coluna de modo que um rótulo diferente apareça na saída (isto é chamado de apelido

da coluna):

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 AS age
```

```
-> FROM pet ORDER BY name;
```

```
+-----+-----+
| name   | age  |
+-----+-----+
| Bowser | 9.58 |
| Buffy  | 9.88 |
| Chirpy | 0.55 |
| Claws  | 5.04 |
| Fang   | 8.59 |
| Fluffy | 6.15 |
| Puffball | 0.00 |
```

```
| Slim      | 2.92 |  
| Whistler  | 1.30 |  
+-----+-----+
```

Para classificar a saída por idade em vez de nome, somente use a cláusula ORDER BY:

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 AS age  
-> FROM pet ORDER BY age;
```

```
+-----+-----+  
| name    | age  |  
+-----+-----+  
| Puffball | 0.00 |  
| Chirpy   | 0.55 |  
| Whistler | 1.30 |  
| Slim     | 2.92 |
```

```
| Claws      | 5.04 |
| Fluffy     | 6.15 |
| Fang       | 8.59 |
| Bowser     | 9.58 |
| Buffy      | 9.88 |
+-----+-----+
```

Uma pergunta parecida pode ser utilizada para determinar a idade dos animais

quando morreram. Você determina os animais que já morreram, checa, não importando se

o valor de morte é NULL e para valores não NULOS, computam a diferença entre valores

de morte e valores de nascimento:

```
mysql> SELECT name, birth, death, (TO_DAYS(death)-TO_DAYS(birth))/365 AS
age
```

```
-> FROM pet WHERE death IS NOT NULL ORDER BY age;
```

```
+-----+-----+-----+-----+
| name   | birth   | death   | age   |
+-----+-----+-----+-----+
| Bowser | 1989-08-31 | 1995-07-29 | 5.91 |
+-----+-----+-----+-----+
```

A pergunta usa morte IS NOT NULL em vez de morte != NULL porque NULL é um valor especial.

Se você deseja saber que animais têm aniversários no próximo mês? Para este tipo de

cálculo, ano e dia são irrelevantes, você simplesmente deseja extrair o mês da

coluna de nascimento. MySQL fornece várias funções de extração de partes da data,

tal como YEAR(), MONTH() and DAYOFMONTH(). MONTH() é a função apropriada aqui. Para ver

como isto trabalha, faz uma pergunta simples que exhibe o valor de ambos data de

nascimento e mês(nascimento):

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

```
+-----+-----+-----+
| name   | birth   | MONTH(birth) |
+-----+-----+-----+
| Fluffy | 1993-02-04 |      2 |
| Claws  | 1994-03-17 |      3 |
| Buffy  | 1989-05-13 |      5 |
| Fang   | 1990-08-27 |      8 |
| Bowser | 1989-08-31 |      8 |
| Chirpy | 1998-09-11 |      9 |
| Whistler | 1997-12-09 |     12 |
| Slim   | 1996-04-29 |      4 |
| Puffball | 1999-03-30 |      3 |
+-----+-----+-----+
```

Descobrir animais com aniversários no próximo mês é fácil, também.  
Suponha o mês

corrente é abril. Então o valor de mês é 4 e você espera animais nascidos  
no mês de

maio (mês 5):

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

```
+-----+-----+
| name  | birth      |
+-----+-----+
| Buffy | 1989-05-13 |
+-----+-----+
```

Há uma pequena complicação se o mês corrente é dezembro. Você não faz  
somente a soma

do número do mês (12) e espera animais produzidos em mês 13, porque não  
há tal mês.

Você espera animais produzidos em janeiro (mês 1).

Você pode até mesmo escrever a pergunta de modo que isto trabalhe com nenhuma matéria

do mês corrente. Aquele caminho que você não tem que usar um número de mês particular

na pergunta. `DATE_ADD()` permite você somar um intervalo do tempo para uma data dada.

Se você soma um mês ao valor de `NOW()`, então o mês desfaz-se `MONTH()`, o resultado

produzido é o mês em que esperávamos os aniversários:

```
mysql> SELECT name, birth FROM pet
      -> WHERE MONTH(birth) = MONTH(DATE_ADD(NOW(), INTERVAL 1 MONTH));
```

Um caminho diferente para efetuar a mesma tarefa é somar 1 para obter o próximo mês

depois o mês corrente:

```
mysql> SELECT name, birth FROM pet
```

```
-> WHERE MONTH(birth) = MOD(MONTH(NOW()), 12) + 1;
```

Note que MÊS volta um número entre 1 e 12. e MOD(alguma coisa,12) volta um número

entre 0 e 11. Assim a adição tem que estar depois de MOD() de outra maneira nós

poderíamos ir de Novembro (11) para Janeiro (1).



## Trabalhando com valores NULOS

O valor NULL pode ser surpreendente até você se acostumar a isto. Conceitualmente,

valores NULL são valores desconhecidos e isto é tratado diferentemente dos outros valores.

Para testar NULL, você não pode usar os operadores de comparação de aritmética tal

como =, < ou !=. Para demonstrar isto, tenta a seguinte pergunta:

```
mysql> SELECT 1 = NULL, 1 != NULL, 1 < NULL, 1 > NULL;
```

```
+-----+-----+-----+-----+
| 1 = NULL | 1 != NULL | 1 < NULL | 1 > NULL |
+-----+-----+-----+-----+
|      NULL |      NULL |      NULL |      NULL |
+-----+-----+-----+-----+
```

Claramente você obtém nenhum resultado significativo destas comparações.

Use os operadores IS NULL e IS NOT NULL:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
```

```
+-----+-----+
| 1 IS NULL | 1 IS NOT NULL |
+-----+-----+
|          0 |              1 |
+-----+-----+
```

No MySQL, 0 são valores falsos e 1 são valores verdadeiros.

Este tratamento especial de NULL é por que isto foi necessário para determinar

se os animais não estão mortos, usando death IS NOT NULL ao invés de death != NULL.



## Unindo Modelo

MySQL fornece um modelo do SQL padronizado unindo assim um formulário de modelo

baseado em expressões regulares parecidas com aquelas utilizadas por utilitários de

Unix tal como vi, grep e sed.

Modelo do SQL permite você usar "\_" para unir qualquer caracter único, e "%" para unir

um número arbitrário de caracteres (incluindo zero caracteres). No MySQL, modelos

do SQL são caso insensível por default. Alguns exemplos são mostrados abaixo. Note

que você não usa = ou != quando você usa modelos do SQL; use o GOSTADO DE ou NÃO

operadores de comparação IGUAIS ao invés.

Para encontrar nomes começam com "b":

```
mysql> SELECT * FROM pet WHERE name LIKE "b%";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Buffy  | Harold | dog     | f     | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m     | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+-----+
```

Para encontrar nomes finalizando com "%fy":

```
mysql> SELECT * FROM pet WHERE name LIKE "%fy";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Fluffy | Harold | cat     | f     | 1993-02-04 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

## Tutorial MySQL

```
| Buffy | Harold | dog      | f      | 1989-05-13 | NULL |
+-----+-----+-----+-----+-----+-----+
```

Para encontrar nomes contendo uns "w":

```
mysql> SELECT * FROM pet WHERE name LIKE "%w%";
```

```
+-----+-----+-----+-----+-----+-----+
| name      | owner | species | sex   | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Claws     | Gwen  | cat     | m     | 1994-03-17 | NULL       |
| Bowser    | Diane | dog     | m     | 1989-08-31 | 1995-07-29 |
| Whistler  | Gwen  | bird    | NULL  | 1997-12-09 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

Para encontrar nomes contendo exatamente cinco caracteres, usam os caracteres de modelo "\_":

```
mysql> SELECT * FROM pet WHERE name LIKE "_____";
```

```
+-----+-----+-----+-----+-----+-----+
| name  | owner | species | sex  | birth      | death |
+-----+-----+-----+-----+-----+-----+
| Claws | Gwen  | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-----+-----+-----+-----+-----+-----+
```

O outro tipo de modelo fornecido por MySQL usa expressões regulares. Quando você

testa este tipo de modelo, usa os operadores REGEXP e NOT REGEXP (ou RLIKE e NOT RLIKE,

que são sinônimos).

Algumas características expressões regulares são:

\* "." casa qualquer caracter único.

\* Uma classe de caracter "[...]" une qualquer caracter dentro dos suportes.

Por exemplo, "[abc]" une "a", "b" ou "c". Para especificar uma série de caracteres,

usam um traço. "[a-z]" une qualquer letra minúscula, ao passo que "[0-9]" une

qualquer dígito.

\* "\*" une zero ou mais solicitações. Por exemplo, "x\*" une qualquer número de "x"

caracteres, "[0-9]\*" une qualquer número de dígitos, e ".\*" une qualquer número de algo.

\* Expressões Regulares são casos sensíveis, mas você pode usar uma classe de caracter

para unir ambos lettercases. Por Exemplo, "[aA]" une letra minúscula ou maiúscula "a"

e "[a-zA-Z]" une qualquer letra em um caso ou outro.

\* O modelo ocorre em qualquer parte na existência do valor testado (modelos do SQL

unem unicamente os valores inteiro).



\* Para ancorar um modelo de modo que isto deve unir o começo ou término da existência

do valor testado, use "^" no começo ou "\$" no final do modelo.

Para trabalho de expressões regulares, as perguntas LIKE e REGEXP podem ser usadas abaixo:

Para encontrar nomes que começam com "b", para unir o começo do nome use "^" e "[bB]" para unir letra minúscula com outra maiúscula "b":

```
mysql> SELECT * FROM pet WHERE name REGEXP "^[bB]";
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29

Para encontrar nomes finalizando com "fy" use e "\$" para unir no final do nome:

```
mysql> SELECT * FROM pet WHERE name REGEXP "fy$";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death |
+-----+-----+-----+-----+-----+-----+
| Fluffy | Harold | cat     | f     | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f     | 1989-05-13 | NULL  |
+-----+-----+-----+-----+-----+-----+
```

Para encontrar nomes contendo "w" use "[wW]" e para unir letra minúscula com outra maiúscula "w":

```
mysql> SELECT * FROM pet WHERE name REGEXP "[wW]";
```

```

+-----+-----+-----+-----+-----+-----+
| name   | owner | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Claws  | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen | bird    | NULL | 1997-12-09 | NULL       |
+-----+-----+-----+-----+-----+-----+

```

Um modelo de expressão regular ocorre em qualquer parte do valor, isto não

é necessário uma pergunta prévia para colocar um wildcard no lado um do outro

para obter a união do valor inteiro, poderia ser utilizado um modelo do SQL.

Para encontrar nomes contendo exatamente cinco caracteres, use "^" e "\$" para unir

o começo e término do nome, e cinco expressões de ".":

```
mysql> SELECT * FROM pet WHERE name REGEXP "^.....$";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death   |
+-----+-----+-----+-----+-----+-----+
| Claws  | Gwen   | cat     | m    | 1994-03-17 | NULL    |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL    |
+-----+-----+-----+-----+-----+-----+
```

Você também podia escrever a pergunta prévia usando "{n}":

```
mysql> SELECT * FROM pet WHERE name REGEXP "^.{5}$";
```

```
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death   |
+-----+-----+-----+-----+-----+-----+
| Claws  | Gwen   | cat     | m    | 1994-03-17 | NULL    |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL    |
+-----+-----+-----+-----+-----+-----+
```

+-----+-----+-----+-----+-----+-----+

## Contando filas

Bancos de Dados são freqüentemente utilizados para responder a pergunta, "quantas vezes

faz um certo tipo de dados ocorrer em uma tabela?" Por exemplo, você pôde querer

saber quantos animais de estimação você tem, ou quantos animais de estimação cada

dono tem, ou você pôde desejar desempenhar várias espécies de censos em seus animais.

Contando o número total de animais você faz a mesma pergunta de "quantas filas

tem na tabela de animal de estimação?" porque há um registro, por cada animal de estimação.

A função COUNT() conta o número de resultados não NULOS, assim a pergunta para

contar seus de animais é:

```
mysql> SELECT COUNT(*) FROM pet;
```

```
+-----+
| COUNT(*) |
+-----+
|         9 |
+-----+
```

Mais, você recuperou os nomes das pessoas que possuíam animais de estimação. Você

pode usar COUNT() se você deseja descobrir quantos animais de estimação cada dono tem:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
```

```
+-----+-----+
| owner  | COUNT(*) |
+-----+-----+
| Benny  |         2 |
| Diane  |         2 |
```

```
| Gwen      |          3 |  
| Harold   |          2 |  
+-----+-----+
```

Note o uso de GROUP BY para agrupar juntamente todos os registros de cada dono.

Sem isto, você obtém uma mensagem de erro:

```
mysql> SELECT owner, COUNT(owner) FROM pet;  
  
ERROR 1140 at line 1: Mixing of GROUP columns (MIN(),MAX(),COUNT()...) with no GROUP columns is illegal if there is no GROUP BY clause
```

COUNT() e AGROUP BY são úteis para caracterizar seus dados em vários caminhos.

Os seguintes exemplos mostram caminhos diferentes para desempenhar operações de

censo animais.



Número de animais por espécie:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
```

```
+-----+-----+
| species | COUNT(*) |
+-----+-----+
| bird   |        2 |
| cat    |        2 |
| dog    |        3 |
| hamster|        1 |
| snake  |        1 |
+-----+-----+
```

Número de animais por sexo:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
```

```
+-----+-----+
| sex   | COUNT(*) |
+-----+-----+
| NULL  |         1 |
| f     |         4 |
| m     |         4 |
+-----+-----+
```

(Nesta saída, NULL indica "sexo desconhecido.")

Número de animais por combinação de espécie e sexo:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
```

```
+-----+-----+-----+
| species | sex   | COUNT(*) |
+-----+-----+-----+
```

```

+-----+-----+-----+
| bird   | NULL |      1 |
| bird   | f     |      1 |
| cat    | f     |      1 |
| cat    | m     |      1 |
| dog    | f     |      1 |
| dog    | m     |      2 |
| hamster| f     |      1 |
| snake  | m     |      1 |
+-----+-----+-----+

```

Para uma tabela inteira você usa COUNT(). Por exemplo, a pergunta prévia, quando

desempenhada somente nos cães e gatos:

```

mysql> SELECT species, sex, COUNT(*) FROM pet
      -> WHERE species = "dog" OR species = "cat"

```

```

-> GROUP BY species, sex;
+-----+-----+-----+
| species | sex  | COUNT(*) |
+-----+-----+-----+
| cat     | f    | 1        |
| cat     | m    | 1        |
| dog     | f    | 1        |
| dog     | m    | 2        |
+-----+-----+-----+

```

Ou, se você desejar o número de animais por sexo, que contenha valores conhecidos para

o sexo dos animais:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
```

```

-> WHERE sex IS NOT NULL

```

```
-> GROUP BY species, sex;
```

```
+-----+-----+-----+
| species | sex  | COUNT(*) |
+-----+-----+-----+
| bird    | f    | 1        |
| cat     | f    | 1        |
| cat     | m    | 1        |
| dog     | f    | 1        |
| dog     | m    | 2        |
| hamster | f    | 1        |
| snake   | m    | 1        |
+-----+-----+-----+
```

## Usando mais que uma tabela

A tabela de animal de estimação guarda informações dos animais de estimação que você

tem. Se você deseja registrar outra informação sobre eles, tal como eventos em suas

vidas, visitas ao veterinário, você necessita de outra tabela.

\* Isto necessita conter o animal de estimação assim você sabe que evento cada animal

participou.

\* Isto necessita uma data assim você sabe quando o evento ocorreu.

\* Isto necessita um campo para descrever o evento.

\* Se você deseja ser capaz de classificar por cada evento.

Dado essas considerações, crie a tabela com a declaração CREATE TABLE:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
```

```
-> type VARCHAR(15), remark VARCHAR(255));
```

Como com a tabela do animal de estimação, é mais fácil carregar os registros iniciais

do que criar um arquivo de texto com tabulação delimitado, contendo a informação:

Fofo

1995-05-15

litter

4 kittens, 3 fêmea, 1 masculino

Buffy

1993-06-23

litter

5 puppies, 2 fêmea, 3 masculino

Buffy

1994-06-19

litter

3 puppies, 3 fêmea

Chirpy

1999-03-21

vet

necessitado beak endireitado

Magro

1997-08-03

vet

quebrado rib

Bowser

1991-10-12

kennel

Presa

1991-10-12

kennel



Presa

1998-08-28

aniversário

Deu ele um novo brinquedo mascado

Claws

1998-03-17

aniversário

Deu ele um novo colarinho de pulga

Whistler

1998-12-09

aniversário

Primeiro aniversário

Carregue os registros:

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

Baseado no que você já aprendeu nas perguntas, continuando na tabela de animal de

estimação, agora você é capaz de desempenhar recuperações nos registros da tabela de

evento; os princípios são os mesmos. Mas quando a tabela de evento é por si mesmo

insuficiente para responder as perguntas você pôde perguntar?

Suponha que deseja descobrir as idades de cada animal de estimação quando eles tiveram

suas crias. A tabela de evento indica quando isto ocorreu, mas para calcular a idade

da mãe, você necessita a data de nascimento que está armazenado na tabela de animal de

estimação, você necessita de ambas tabelas:

```
mysql> SELECT pet.name, (TO_DAYS(date) - TO_DAYS(birth))/365 AS age,  
remark
```

```
-> FROM pet, event
```

```
-> WHERE pet.name = event.name AND type = "litter";
```

```
+-----+-----+-----+
| name   | age   | remark                               |
+-----+-----+-----+
| Fluffy | 2.27  | 4 kittens, 3 female, 1 male       |
| Buffy  | 4.12  | 5 puppies, 2 female, 3 male       |
| Buffy  | 5.10  | 3 puppies, 3 female                |
+-----+-----+-----+
```

Há várias coisas para notar sobre esta pergunta:

\* A cláusula FROM lista duas tabelas porque a pergunta necessita puxar informação de ambas.

\* Quando combina informação de tabelas múltiplas, você necessita registrar em uma

tabela que pode ser a união do outro registro. Isto é fácil porque ambos tem uma

coluna de nome. A cláusula WHERE pergunta para as duas tabelas os valores do nome.

\* A coluna de nome ocorre em ambas tabelas, você deve ser específico sobre que tabela

você está se referindo à coluna.

Você tem duas tabelas diferentes para uni-las. Às vezes isto é útil para unir

uma tabela para si mesmo, se você deseja comparar registros em uma tabela com

outros registros naquela mesma tabela. Por exemplo, para encontrar o sexo entre seus animais de estimação, você pode unir a tabela de animal de estimação

com si mesmo para juntar os masculinos e as fêmeas da mesma espécie:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
```

```
-> FROM pet AS p1, pet AS p2
```

```
-> WHERE p1.species = p2.species AND p1.sex = "f" AND p2.sex = "m";
```

```
+-----+-----+-----+-----+-----+
| name   | sex   | name   | sex   | species |
+-----+-----+-----+-----+-----+
| Fluffy | f     | Claws  | m     | cat     |
| Buffy  | f     | Fang   | m     | dog     |
| Buffy  | f     | Bowser | m     | dog     |
+-----+-----+-----+-----+-----+
```

Obtendo informação sobre bancos de dados e tabelas

Se você esquecer o nome de um banco de dados ou tabela, ou a estrutura de uma tabela

(como suas colunas são chamadas). MySQL resolve este problema através de várias

declarações que fornecem informação sobre os bancos de dados e tabelas.

Você já viu `SHOW DATABASES`, que lista os bancos de dados administrados pelo servidor.

Para descobrir qual o banco de dados corrente foi selecionado, usa a função `DATABASE()`:

```
mysql> SELECT DATABASE();
```

```
+-----+  
| DATABASE() |  
+-----+  
| menagerie |  
+-----+
```

Se você não tem um banco de dados selecionado ainda, o resultado é vazio.

Para descobrir que tabelas o banco de dados corrente contem ou quando você não está

seguro sobre o nome de uma tabela, use este comando:

```
mysql> SHOW TABLES;
```

```
+-----+
| Tables in menagerie |
+-----+
| event                |
| pet                  |
+-----+
```

Se você deseja descobrir sobre a estrutura de uma tabela, o comando DESCRIBE é útil;

isto exibe informação sobre cada coluna de uma tabela:

```
mysql> DESCRIBE pet;
```

```
+-----+-----+-----+-----+-----+-----+
| Field  | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
```

name	varchar(20)	YES		NULL		
owner	varchar(20)	YES		NULL		
species	varchar(20)	YES		NULL		
sex	char(1)	YES		NULL		
birth	date	YES		NULL		
death	date	YES		NULL		

Field indica o nome de coluna, Type é o tipo de dados, Null indica que não importa se

a coluna pode conter valores NULOS, Key indica que não importa se a coluna é ordenada

e Default especifica valor de default da coluna.

Se ordenou uma tabela, SHOW INDEX FROM tbl\_name produzimos informação sobre elas.



## Usando MySQL em modo lote

Você e o MySQL utilizaram entrar com as perguntas e ver os resultados. Você também

pode também executar MySQL em modo lote. Para fazer isto, coloque os comandos que

you deseja num arquivo, então peça ao MySQL para ler a entrada do arquivo:

```
shell> mysql < batch-file
```

Se você necessita especificar parâmetros de conexão na linha de comando, use:

```
shell> mysql -h host -u user -p < batch-file
```

```
Enter password: *****
```

Quando usa este caminho no MySQL, você está criando um arquivo de escrita, e o executa.

Por que usar uma escrita? Aqui são umas das poucas razões:

- \* Se você corre um `repeatedly` de pergunta, fazendo isto numa escrita permite a você

evitar o retying a cada tempo que você executar.

- \* Você pode gerar novas perguntas de arquivos existentes que são parecidos, copiando

e editando arquivos de escrita.

- \* Modo Lote pode também ser útil enquanto você está desenvolvendo uma pergunta,

particularmente por comandos de linha múltiplas ou declarações múltiplas de seqüências

de comandos.

- \* Se você tem uma pergunta que produz uma saída, você pode fazer que a saída ocorra

através de um pager:

```
shell> mysql < batch-file | more
```

\* Você pode pegar a saída em um arquivo para ser processado:

```
shell> mysql < batch-file > mysql.out
```

\* Você pode distribuir sua escrita para outras pessoas.

\* Algumas situações não levam em conta o uso interativo, por exemplo, quando você faz

uma pergunta de um trabalho. Neste caso, você deve usar modo lote.

O formato da saída de default é diferente (mais conciso) quando você pergunta ao MySQL

em modo de lote, usa isto interativamente. Por exemplo, a saída de espécie

```
+-----+  
| species |  
+-----+  
| bird   |  
| cat    |  
| dog    |  
| hamster|  
| snake  |  
+-----+
```

Quando vista em modo de lote:

```
cobra  
hamster  
cão  
gato  
pássaro
```

espécie

Se você deseja obter a saída interativa formatada em modo lote, use `mysql -t`. Para ecoar

à saída os comandos que são executados, use `mysql -vvv`.

Apostila de MySQL

1

3