Guia de Consulta Rápida

MySQL 5

Juliano Niederauer
Rubens Prates

Novatec Editora

Editor: Rubens Prates

ISBN: 85-7522-081-0

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 São Paulo SP – Brasil
Tel.: +55 11 6959-6529
Fax: +55 11 6950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br

Introdução

Objetivos

Este guia foi criado para orientar os programadores e administradores de bancos de dados que desejam conhecer as funcionalidades do SGBD (Sistema de Gerência de Bancos de Dados) MySQL. Para isso, serão apresentados os principais recursos desse software, incluindo a referência dos comandos SQL, operadores, funções, tipos de dados, programas clientes e procedimentos de configuração do SGBD.

O guia também apresenta a descrição das funções de conectividade entre o MySQL e a linguagem PHP, utilizada para a criação de páginas dinâmicas na Web.

O que é MySQL?

MySQL é um SGBD relacional que utiliza a linguagem padrão SQL (*Structured Query Language*) e é largamente utilizado em aplicações para a Internet. É o mais popular entre os bancos de dados com código-fonte aberto. Há mais de cinco milhões de instalações do MySQL no mundo todo, inclusive em sites com alto volume de dados e de tráfego, como *Associated Press, Google, NASA, Sabre Holdings* e *Suzuki*.

O MySQL á uma alternativa atrativa porque, mesmo possuindo uma tecnologia complexa de banco de dados, seu custo é bastante baixo. Tem como destaque suas características de velocidade, escalabilidade e confiabilidade, o que vem fazendo com que ele seja adotado por departamentos de TI (Tecnologia da Informação), desenvolvedores Web e vendedores de pacotes de softwares.

A seguir são listadas algumas vantagens do MySQL:

- número ilimitado de utilização por usuários simultâneos;
- capacidade de manipulação de tabelas com mais de 50.000.000 de registros;
- alta velocidade de execução de comandos;
- fácil e eficiente controle de privilégios de usuários.

Para o desenvolvimento de sites dinâmicos, o MySQL forma uma excelente dupla com a linguagem PHP, tanto para websites pequenos como para grandes portais.

Download e instalação

Para efetuar o download do MySQL, acesse o site http://www.mysql.com, entre na seção de produtos, escolha "Database Server" e faça o download do arquivo de instalação para o seu sistema operacional. Se você pretende instalar o MySQL no Linux, uma boa opção é fazer o download do pacote *rpm*, que é de fácil instalação.

No caso do Windows, ao terminar o download do arquivo, execute-o para inciar a instalação, e siga as instruções que irão aparecer na tela. Durante a instalação, você poderá optar se deseja que o servidor MySQL seja iniciado automaticamente na inicialização do sistema. É bom você marcar essa opção, para não ter que iniciar manualmente o servidor cada vez que a máquina for reiniciada.

Qualquer dúvida, consulte a documentação contida no subdiretório "Docs".

Extensão MySQL para o PHP

Se você pretende usar o MySQL com a linguagem PHP, será necessário habilitar a extensão *mysql* (ou *mysqli*) no arquivo de configuração do PHP, chamado `php.ini`. Isso é feito através da seguinte linha:

```
extension=php_mysql.dll
```

Se a sua versão do MySQL for a 4.1 ou superior, você terá que usar a extensão "mysqli":

```
extension=php_mysqli.dll
```

Normalmente essas linhas já existem no seu arquivo `php.ini`, mas estão comentadas com uma vírgula na frente. Nesse caso, basta remover a vírgula. Em seguida, reinicie o servidor Web para que as alterações tenham efeito.

Novidades do MySQL 5.0

As características listadas a seguir devem fazer parte da versão 5.0 do MySQL, sendo que algumas delas, como os procedimentos armazenados (*stored procedures*), já estão incluídas desde a versão alpha. Outras estão apenas parcialmente implementadas, como por exemplo os cursores.

Views

As visões (views) permitem que os usuários acessem um conjunto de tabelas como se fosse uma única tabela. Ou seja, seria como uma tabela virtual.

Stored Procedures e Triggers

Os procedimentos armazenados (*stored procedures*) são conjuntos de comandos SQL que podem ser armazenados no servidor, para posteriormente serem chamados pelos clientes. Estão implementados com base no padrão SQL:2003.

Os gatilhos (*triggers*) são objetos que podem ser associados às tabelas do banco de dados, e são ativados na ocorrência de determinados eventos nessas tabelas.

Outras funcionalidades e melhorias

- Suporte elementar aos cursores. Os cursores simples são suportados dentro dos procedimentos armazenados e funções, e utilizam a mesma sintaxe do SQL embutido (*embedded SQL*).
- Especificação explícita do uso de índices RTREE para criação tabelas MyISAM. No MySQL 4.1, os índices RTREE são usados apenas internamente para dados geométricos.
- Linhas de tamanho dinâmico para tabelas MEMORY. A nova versão também possui um melhor suporte a esse tipo de tabelas.
- Suporte a dicionário de dados / INFORMATION_SCHEMA.
- Linhas de tamanho dinâmico e manipulação mais rápida das mesmas, devido à redução no número de cópias.

Para ver uma descrição mais detalhada de todas as melhorias e novas funcionalidades, assim como as mudanças planejadas para versões futuras, consulte a seção "News" do manual do MySQL.

Operações básicas do MySQL

Criando um banco de dados

No MySQL, para criar um banco de dados utilizamos o utilitário *mysql*. Trata-se do programa cliente, que é executado pela própria linha de comando (tanto do Windows como do Linux). Se você não tiver a permissão para a criação de um banco de dados, deve solicitar ao seu administrador para que ele crie sua base de dados inicial. Caso contrário, execute as seguintes linhas para criar um banco de dados chamado `bdteste`:

```
mysql
    > CREATE DATABASE bdteste;
    > USE bdteste;
```

Veja que, primeiramente executamos o programa cliente mysql, para depois escolher o banco de dados que será utilizado, e isso é feito por meio do comando **USE**.

Se a criação do banco de dados foi feita por seu administrador, provavelmente você terá um nome de usuário e uma senha para acesso à sua base de dados. Para acessá-la, você deverá digitar o seguinte comando:

```
mysql -u username -p
```

Por exemplo, supondo que no Linux o *mysql* esteja localizado em `/usr/bin` e você queira se conectar com o usuário *root*, bastaria digitar:

```
/usr/bin/mysql -u root -p
```

A opção `-u` indica que o parâmetro seguinte é o nome de usuário utilizado para o acesso, e a opção `-p` indica que será digitado um password (senha) para conexão. Após a digitação da senha, você já estará conectado ao banco de dados criado, podendo inserir e manipular livremente seus dados.

Criando tabelas

Após a criação do banco de dados, podemos acessá-lo (com o utilitário **mysql**) e começar a criar as tabelas que armazenarão os dados. Isso é feito por meio do comando **CREATE TABLE**, cuja sintaxe mais básica é a seguinte:

```
CREATE TABLE <nome_tabela> (
    <nome_campo> tipo_de_dado [NULL | NOT NULL]
    [DEFAULT valor_padrão], ...
    );
```

No tópico "Comandos SQL" deste guia, você encontra uma descrição completa do comando, incluindo chaves primárias e outras opções. Veja o significado de cada uma das partes desse comando:

Parte	Descrição
nome_tabela	Representa o nome da tabela que será criada. Não pode haver nomes de tabelas repetidos.
nome_campo	Representa o nome pelo qual o campo será referenciado.
tipo_de_dado	Deve ser substituído por um dos tipos apresentados no tópico anterior.
NULL | **NOT NULL**	Define se o campo pode aceitar valores nulos ou não.
DEFAULT	Define um valor-padrão para inserções na tabela. Esse valor será utilizado se nenhum valor para este campo for informado.

Por exemplo, imagine que precisamos de uma tabela para armazenar as seguintes informações sobre os produtos de uma loja virtual:

Código

Nome

Descrição

Preço

Categoria

Para criar essa tabela com o comando **CREATE TABLE**, faríamos da seguinte forma:

```
CREATE TABLE produtos (
    codigo_produto smallint NOT NULL,
    nome_produto varchar(80) NOT NULL,
    descricao_produto text,
    preco float NOT NULL,
    cod_categoria smallint NOT NULL,
    primary key(codigo_produto)
    );
```

Os campos seguidos da cláusula **NOT NULL** não aceitarão valores nulos no momento da inserção no banco de dados. Quanto aos tipos de dados (**int**, **float**, **char** etc.) aceitos pelo MySQL, consulte o tópico "Tipos de colunas" deste guia.

Para visualizar o nome das tabelas criadas no MySQL, digitamos:

```
SHOW TABLES;
```

E para visualizar a estrutura de determinada tabela, basta digitar:

```
DESC <nome_tabela>;
```

Você verá uma tela com os nomes dos campos, o tipo, se aceitam valores **NULL**, entre outras características da tabela. Por exemplo:

```
mysql> desc produtos;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| codigo_produto  | smallint(6) | NO   | PRI |         |       |
| nome_produto    | varchar(80) | NO   |     |         |       |
| descricao_produto | text      | YES  |     | NULL    |       |
| preco           | float       | NO   |     |         |       |
| cod_categoria   | smallint(6) | NO   |     |         |       |
+-----------------+-------------+------+-----+---------+-------+
5 rows in set (0.12 sec)
```

Inserindo dados

Para incluir um ou mais registros em uma tabela, utilizamos o comando **INSERT** da SQL. Existem duas sintaxes possíveis para esse comando:

```
INSERT INTO <nome_tabela> VALUES (valor1, valor2, ..., valorn);
```

ou

```
INSERT INTO <nome_tabela> (nome_campo1, ..., nome_campox)
    VALUES (valor1, valor2, ..., valorx);
```

Na primeira variação, os valores digitados no lugar de *valor1*, *valor2*, ..., *valorn* serão incluídos na mesma ordem em que foram definidos os campos no momento da criação da tabela.

Na segunda variação do comando insert, os valores serão inseridos na ordem especificada entre parênteses, logo após o nome da tabela. Os campos não listados receberão o valor **NULL** ou o valor-padrão (caso exista um).

Os valores numéricos não devem ser delimitados por aspas. Observe o exemplo a seguir, em que utilizamos o comando **INSERT** para fazer a inclusão de um produto na tabela produtos:

```
INSERT INTO produtos VALUES (1, 'Refrigerador Tabajara Plus',
    'Excelente refrigerador, com 30 anos de garantia.', 1500, 1);
```

Após a execução do comando, deve aparecer uma mensagem do tipo:

```
Query OK, 1 row affected (0.75 sec)
```

Note que na tabela `produtos` existe um campo (`descricao_produto`) que não foi declarado com a cláusula **`NOT NULL`**. Nesse caso, podemos utilizar a segunda variação do comando **`INSERT`**:

```
INSERT INTO produtos (codigo_produto,nome_produto, preco,
    cod_categoria) VALUES (2, 'Fogão 16 bocas Tabajara', 549.90, 1);
```

Assim, o campo `descricao_produto` ficará com o valor **`NULL`**.

Observação: se tivéssemos um campo do tipo **`date`**, o formato de entrada dependeria da configuração do SGBD. O formato mais usado é o *aaaa-mm-dd*, delimitado por aspas simples (por exemplo: '2005-11-24').

Alterando registros ou tabelas

O comando **`UPDATE`** realiza alterações em um ou mais registros de determinada tabela. Sua sintaxe é a seguinte:

```
UPDATE <nome_tabela>
    SET campo1=valor1 [, campo2=valor2, ..., campon=valorn]
    [WHERE <condições>];
```

Se a cláusula **`WHERE`** não for utilizada, a alteração será efetuada em todos os registros da tabela. Se quisermos, por exemplo, alterar o preço de um produto, podemos utilizar o seguinte comando:

```
UPDATE produtos SET preco=1700 WHERE codigo_produto=1;
```

O outro tipo de alteração que podemos fazer em uma tabela é em sua estrutura, utilizando o comando **`ALTER TABLE`**. Podem ser usadas as seguintes sintaxes:

```
ALTER TABLE <nome_tabela> ADD <nome_campo> tipo_de_dado;
    ALTER TABLE <nome_tabela> RENAME TO <novo_nome_tabela>;
```

Para adicionar, por exemplo, o campo fabricante à nossa tabela de produtos, devemos executar a seguinte linha:

```
ALTER TABLE produtos ADD fabricante varchar(50);
```

O campo fabricante será incluído como o último campo da tabela produtos.

Excluindo registros ou tabelas

O comando **`DELETE`** exclui um ou mais registros de determinada tabela. Sua sintaxe é a seguinte:

```
    DELETE FROM <nome_tabela> [WHERE <condições>];
```

Se a cláusula **`WHERE`** não for utilizada, todos os registros da tabela serão excluídos. Por exemplo, para excluir o produto de código 2, podemos digitar o comando a seguir:

```
DELETE FROM produtos WHERE codigo_produto=2;
```

Para excluir todos os produtos existentes na tabela, basta digitar:

```
DELETE FROM produtos;
```

Para excluir uma tabela inteira (dados e estrutura) do banco de dados, usamos o comando **`DROP TABLE`**. Sua sintaxe é a seguinte:

```
    DROP TABLE <nome_tabela1> [,<nome_tabela2>, ...];
```

Por exemplo, para excluir a tabela de produtos, basta digitar:

```
DROP TABLE produtos;
```

Existe também o comando **DROP DATABASE**, que serve para excluir um banco de dados inteiro. Porém, para executar esse comando é necessário que o administrador do sistema lhe dê essa permissão.

Executando consultas

Com o comando **SELECT** podemos executar diversos tipos de consultas sobre as tabelas de um banco de dados. Sua sintaxe mais básica é a seguinte:

```
SELECT <lista_de_campos> FROM <lista_de_tabelas>
  [WHERE <condições>];
```

Consulte a sintaxe completa no tópico "Comandos SQL" deste guia.

Se *<lista_de_campos>* for substituída por um asterisco (*), serão retornados todos os campos existentes na(s) tabela(s) em uso. Se a cláusula **WHERE** for omitida, serão mostrados todos os registros das tabelas determinadas em *<lista_de_tabelas>*.

Por exemplo, para selecionar os campos código, nome e preço da tabela de produtos, o comando seria:

```
SELECT codigo_produto, nome_produto, preco FROM produtos;
```

O resultado seria semelhante ao apresentado na figura a seguir:

```
mysql> SELECT codigo_produto, nome_produto, preco FROM produtos;
+----------------+---------------------------+--------+
| codigo_produto | nome_produto              | preco  |
+----------------+---------------------------+--------+
|              1 | Refrigerador Tabajara Plus |  1500 |
|              2 | Fogão 16 bocas Tabajara    | 549.9 |
+----------------+---------------------------+--------+
2 rows in set (0.00 sec)
```

Agora vamos ver um exemplo usando a cláusula **WHERE** para determinar quais registros devem ser retornados. Se quisermos, por exemplo, obter o nome dos produtos que custam mais de R$600,00, executamos o seguinte comando:

```
SELECT nome_produto FROM produtos WHERE preco > 600;
```

Se houvesse mais de uma condição a ser analisada na cláusula **WHERE**, utilizaríamos os operadores lógicos **AND** e **OR** entre elas. Por exemplo:

```
SELECT codigo_produto, nome_produto FROM produtos
    WHERE cod_categoria=2 AND preco<100;
```

A linguagem SQL nos oferece um operador de grande utilidade, que é o **LIKE**. Com ele, podemos descobrir, por exemplo, todos os produtos que começam com a letra "F", ou todos que terminam com a letra "A", ou ainda todos que contenham a letra "O". Por exemplo:

```
SELECT * FROM produtos WHERE nome_produto LIKE 'Fogão%';
```

Esse comando retorna todos os produtos cujo nome começa com "*Fogão*". O símbolo % representa uma seqüência de caracteres. Poderíamos usar também o símbolo _, que representa um caractere qualquer.

Agora vamos ver como realizar outros tipos de consultas. Por exemplo, para contar quantos registros existem em nossa tabela de produtos, utilizamos a função **COUNT**:

```
SELECT COUNT(*) FROM produtos;
```

Para somar os valores de determinado campo de uma tabela, utilizamos a função **SUM**. Vamos somar os preços de todos os produtos de nossa loja, executando o seguinte comando:

```
SELECT SUM(preco) FROM produtos;
```

Para descobrir quanto custa o produto mais caro da loja, podemos utilizar a função **MAX**:

```
SELECT MAX(preco) FROM produtos;
```

Se ao invés da função **MAX** fosse utilizada a função **MIN**, seria retornado o preço do produto mais barato da loja.

A linguagem SQL nos oferece ainda a opção **GROUP BY**, que nos permite agrupar os resultados de uma consulta. Se quisermos, por exemplo, saber quantos produtos existem em cada uma das categorias, podemos fazer da seguinte forma:

```
SELECT cod_categoria,COUNT(*)
    FROM produtos GROUP BY cod_categoria;
```

O comando **GROUP BY** também pode ser utilizado com a opção **HAVING**, que seleciona alguns registros retornados pelo **GROUP BY**. Para retornar, por exemplo, somente as categorias que possuem 1 único produto cadastrado, digitamos o seguinte comando:

```
SELECT cod_categoria,COUNT(*) FROM produtos GROUP BY cod_
    categoria HAVING COUNT(*)=1;
```

Ordenando e limitando os resultados

Para ordenar os registros retornados por uma consulta, utilizamos a cláusula **ORDER BY**, seguida pelo nome dos campos a serem ordenados. Os campos serão avaliados da esquerda para a direita. Um exemplo seria mostrar em ordem alfabética o nome de todas os produtos da loja:

```
mysql> SELECT nome_produto FROM produtos ORDER BY nome_produto;
+-------------------------+
| nome_produto            |
+-------------------------+
| Fogão 16 bocas Tabajara |
| Refrigerador Tabajara Plus |
+-------------------------+
2 rows in set (0.00 sec)
```

Podemos também fazer uma ordenação decrescente dos resultados, e para isso basta utilizar a opção **DESC** após o nome do campo:

```
SELECT nome_produto FROM produtos ORDER BY nome_produto DESC;
```

Podemos também utilizar a cláusula **LIMIT** para determinar o número máximo de registros que uma consulta pode retornar. Se, por exemplo, um usuário fizer uma busca pela palavra "CD", e houver mais de 500 CDs na loja, obviamente não mostraremos os 500 CDs na mesma tela, pois a página ficaria muito grande. Poderíamos, então, determinar que devem ser retornados no máximo 10 CDs, utilizando o seguinte comando:

```
SELECT * FROM produtos
    WHERE nome_produto LIKE 'CD %'  LIMIT 10;
```

Dessa forma exibiríamos apenas os 10 primeiros CDs ao usuário, e colocaríamos um link para a próxima página, em que seriam mostrados os próximos 10. É claro que, para obtermos 10 registros por vez, devemos utilizar a opção **ORDER BY**. Assim, evitamos que os registros apareçam em uma ordem diferente a cada chamada. Por exemplo:

```
SELECT * FROM produtos WHERE nome_produto LIKE 'CD %'
    ORDER BY nome_produto LIMIT 10;
```

O parâmetro **OFFSET** pode ser utilizado em conjunto com o **LIMIT**, para determinar a partir de qual registro a consulta deve retornar. Portanto, para retornar os próximos 10 registros, utilizamos o seguinte comando:

```
SELECT * FROM produtos WHERE nome_produto LIKE 'CD %'
    ORDER BY nome_produto LIMIT 10 OFFSET 10;
```

Então bastaria incrementar o parâmetro **OFFSET** de 10 em 10 para obter todos os produtos que contêm a palavra "CD".

Numeração automática de campos

Em algumas ocasiões, precisamos criar em uma tabela um campo com numeração automática (sequencial). No MySQL, isso pode ser feito com a propriedade de auto-incremento, da seguinte forma.

```
CREATE TABLE produto (
    codigo int NOT NULL AUTO_INCREMENT,
    nome varchar(70) NOT NULL,
    primary key(codigo)
    );
```

Para inserir registros nessa tabela, faríamos do seguinte modo:

```
INSERT INTO produto (nome) VALUES ('Raquete de Tênis');
    INSERT INTO produto (nome) VALUES ('Taco de Baseball');
```

Nesse exemplo, o produto "Raquete de Tênis" ficará com o código 1, e o produto "Taco de Baseball" ficará com o código 2.

Comentários no código SQL

Se você criar um arquivo de script SQL, existe a possibilidade de comentar alguns trechos, para evitar que determinados comandos sejam executados quando o script for chamado. Os caracteres #, -- e /* */ são utilizados para indicar comentários no código SQL:

```
#   comentário para o fim da linha
    --  comentário para o fim da linha. Requer pelo menos um espaço após
        --
    /* comentário de uma única linha */
    /*
    este comentário ocupa
    mais de uma linha
    */
```

Existem algumas limitações no uso de /*...*/:

- Caracteres como apóstrofo e aspas indicam o início ou o fim de um string mesmo dentro do comentário.
- Ponto e vírgula indicam o final do comando. Tudo que estiver após o ponto e virgula é interpretado como um novo comando.

Tipo especial de comentário

A partir do MySQL 3.23 é possível desabilitar recursos específicos do MySQL, utilizando a forma especial de comentário /*! ... */ dentro de um comando. O sinal de ! indica que o MySQL deve interpretar e executar o código dentro dos comentários, enquanto outros servidores ignoram o que estiver entre /* e */. Por exemplo:

```
SELECT /*! STRAIGHT_JOIN */ coluna
    FROM tabela1, tabela2 WHERE ...
```

No exemplo anterior somente o MySQL considerará a opção **STRAIGHT_JOIN**.

Também é possível desabilitar recursos recursos de versões antigas do MySQL. Se você adiciona o número da versão após o sinal '!', a sintaxe será executada somente se a versão do MySQL for igual ou mais recente que o número da versão utilizada:

```
CREATE /*!32302 TEMPORARY */ TABLE (a int);
```

O comando anterior significa que o MySQL utilizará a palavra-chave **TEMPORARY** se a versão for 3.23.02 ou mais recente.

Nomes de identificadores

Nomes de bancos de dados, tabelas, índices, colunas e alias seguem as mesmas regras no **MySQL**.

Identificador	Tam. máx	Caracteres permitidos
Banco de dados	64	Qualquer caractere permitido em um nome de diretório, exceto '/', '\' ou '.'
Tabela	64	Qualquer caractere permitido em um nome de arquivo, exceto '/', '\' ou '.'
Coluna	64	Todos os caracteres
Índice	64	Todos os caracteres
Alias	255	Todos os caracteres

As regras para nomes são as seguintes:

- Um nome pode conter caracteres alfanuméricos do character set corrente e também os caracteres "_" e "$". O character set default é ISO-8859-1 (Latin1); que pode ser alterado recompilando o MySQL.
- Um nome pode iniciar com qualquer caractere válido, inclusive um número, mas não somente números.
- O caractere "." (ponto) não pode ser utilizado dentro do nome pois é utilizado nas referências de colunas, que podem ser feitas das seguintes formas:

Referência	Significado
coluna	Nome da coluna que é utilizada dentro de um comando select, update, etc.
'*coluna*'	Nome de uma coluna delimitada com apóstrofe indica que o nome da coluna é uma palavra reservada ou contêm caracteres especiais.
tabela.*coluna*	Nome da coluna da tabela especificada do banco de dados corrente.
bd.*tabela*.*coluna*	Nome da coluna da tabela e banco de dados especificados.

Uso de maiúsculas e minúsculas

No MySQL, bancos de dados e tabelas correspondem a diretórios e arquivos dentro desses diretórios. Em razão disso, no Unix é feita distinção entre minúsculas e maiúsculas nos nomes de tabelas e bancos de dados, e no Win32 não é feita distinção.

Nos nomes de colunas não é feita distinção, em todos os casos.

Comandos SQL

Este tópico contém uma referência dos comandos básicos da SQL (*Structured Query Language* – Linguagem de Consulta Estruturada), a linguagem padrão para manipulação de bancos de dados no MySQL. Para cada comando é apresentada sua descrição e sintaxe, assim como seus principais parâmetros.

ALTER DATABASE >4.1.1

Permite alterar as características de um banco de dados. Essas características ficam armazenadas no arquivo "*db.opt*", localizado no diretório do banco de dados. Para executar esse comando, você precisa ter o privilégio de alteração (**ALTER**).

```
ALTER {DATABASE | SCHEMA} nome_bd ação [,ação] ...
```

O parâmetro *ação* pode ser:

```
    [DEFAULT] CHARACTER SET conj_caracteres
  | [DEFAULT] COLLATE comp_caracteres
```

Parâmetro	Significado
conj_caracteres	Conjunto de caracteres (charset) usado no banco de dados (ex: latin1, utf8 etc.). Você pode usar o comando **SHOW CHARACTER SET** para visualizar as opções disponíveis.
comp_caracteres	Conjunto de regras (*collation*) para comparação dos caracteres de um conjunto (charset). Você pode usar o comando **SHOW COLLATION** para visualizar as opções disponíveis.

ALTER PROCEDURE e ALTER FUNCTION >5.0.3

Pode ser usado para renomear um procedimento armazenado (*stored procedure*) ou função, além de alterar suas características.

```
ALTER {PROCEDURE | FUNCTION} nome [característica ...]
```

característica:

```
NAME novo_nome
  | SQL SECURITY {DEFINER | INVOKER}
  | COMMENT 'comentário'
```

ALTER SCHEMA >5.0.2

Permite alterar as características de um banco de dados. Para executar esse comando, você precisa ter o privilégio de alteração (**ALTER**).

```
ALTER {DATABASE | SCHEMA} nome_bd ação [,ação] ...
```

O parâmetro *ação* pode ser:

```
[DEFAULT] CHARACTER SET conj_caracteres
  | [DEFAULT] COLLATE comp_caracteres
```

Para uma descrição detalhada dos parâmetros, veja o comando **ALTER DATABASE**, cuja sintaxe é a mesma.

ALTER TABLE

Modifica os atributos de um campo ou adiciona um novo campo em uma determinada tabela em um banco de dados selecionado.

```
ALTER [IGNORE] TABLE tabela  ação [, ação...]
```

Parâmetro	Significado
IGNORE	Utiliza somente a primeira linha quando encontrar duplicidade em uma chave **UNIQUE**. As demais serão eliminadas. Se a opção não for especificada, será emitida uma mensagem de erro e nenhuma alteração será feita.
tabela	Nome da tabela a ser alterada.
ação	Ação ou ações a serem efetuadas na tabela.

Ações do comando ALTER TABLE

ADD COLUMN

Adiciona uma coluna à tabela.

```
ALTER [IGNORE] TABLE tabela
  ADD [COLUMN] decl_coluna [FIRST | AFTER nome_coluna]
  | ADD [COLUMN] (decl_coluna,...)
```

Parâmetro	Significado
decl_coluna	Declaração da coluna; mesmo formato usado no comando **CREATE TABLE**.
FIRST	Adiciona a nova coluna na primeira posição da tabela.
AFTER	Adiciona a nova coluna após a coluna especificada.

ADD INDEX

Adiciona um índice à tabela.

```
ALTER [IGNORE] TABLE tabela
  ADD INDEX [índice] [tipo_índice] (coluna,...)
```

ADD PRIMARY KEY

Adiciona uma chave primária à tabela.

```
ALTER [IGNORE] TABLE tabela
  ADD [CONSTRAINT [símbolo]]
  PRIMARY KEY [tipo_índice] (coluna,...)
```

ADD UNIQUE

Adiciona um índice sem duplicidade à tabela.

```
ALTER [IGNORE] TABLE tabela
  ADD [CONSTRAINT [símbolo]]
  UNIQUE [índice] [tipo_índice] (coluna,...)
```

ADD FULLTEXT/ ADD SPATIAL[>4.1]

Adiciona um índice para colunas do tipo texto ou para colunas espaciais (ex: tipos geométricos).

```
ALTER [IGNORE] TABLE tabela
  ADD [FULLTEXT|SPATIAL] [índice] (coluna,...)
```

ADD FOREIGN KEY[>4.0]

Adiciona uma chave estrangeira à tabela.

```
ALTER [IGNORE] TABLE tabela
  ADD [CONSTRAINT [símbolo]]
  FOREIGN KEY [índice] (coluna,...)
  [definição_referência]
```

ALTER COLUMN

Altera (**SET**) ou elimina (**DROP**) o valor padrão de uma coluna.

```
ALTER [IGNORE] TABLE tabela
  ALTER [COLUMN] coluna
  {SET DEFAULT literal | DROP DEFAULT}
```

CHANGE COLUMN

Altera o nome e/ou a declaração de uma coluna.

```
ALTER [IGNORE] TABLE tabela
  CHANGE [COLUMN] coluna  decl_coluna
  [FIRST|AFTER nome_coluna]
```

Parâmetro	Significado
coluna	Nome atual da coluna.
decl_coluna	Declaração da nova coluna (mesmo formato usado no comando **CREATE TABLE**).
FIRST[>4.0.1]	Posiciona a coluna no início da tabela.
AFTER[>4.0.1]	Posiciona a coluna após a especificada por *nome_coluna*.

DISABLE KEYS[>4.0]

Indica que o MySQL deve parar de atualizar os índices que não são únicos (uniques).

```
ALTER [IGNORE] TABLE tabela
  DISABLE KEYS
```

DROP COLUMN

Remove uma coluna da tabela.

```
ALTER [IGNORE] TABLE tabela
  DROP [COLUMN] coluna
```

DROP INDEX

Remove um índice da tabela.

```
ALTER [IGNORE] TABLE tabela
  DROP INDEX índice
```

DROP PRIMARY KEY

Remove a chave primária da tabela.

```
ALTER [IGNORE] TABLE tabela
  DROP PRIMARY KEY
```

DROP FOREIGN KEY[>4.0.13]

Remove uma chave estrangeira da tabela especifica.

```
ALTER [IGNORE] TABLE tabela
  DROP FOREIGN KEY símbolo_fk
```

ENABLE KEYS[>4.0]

Usado para recriar os índices perdidos devido ao uso da opção **DISABLE KEYS**.

```
ALTER [IGNORE] TABLE tabela
  ENABLE KEYS
```

MODIFY COLUMN

Altera a declaração de uma coluna. Similar a **CHANGE COLUMN**, exceto que não permite alterar o nome da coluna.

```
ALTER [IGNORE] TABLE tabela
  MODIFY [COLUMN] decl_coluna
  [FIRST | AFTER nome_coluna]
```

Parâmetro	Significado
decl_coluna	Declaração da coluna (mesmo formato usado no comando **CREATE TABLE**).
FIRST[>4.0.1]	Posiciona a coluna no início da tabela.
AFTER[>4.0.1]	Posiciona a coluna após a especificada por *nome_coluna*.